UNIVERSITÀ DEGLI STUDI DI TRIESTE
Sede Amministrativa del Dottorato di Ricerca

XXIX CICLO - DOTTORATO IN
INGEGNERIA E ARCHITETTURA

Genetic Programming Techniques for Regular Expression inference from Examples

Settore scientifico-disciplinare ING-INF/05

DOTTORANDO
Fabiano Tarlao

COORDINATORE DOTTORATO DI RICERCA
Chiar.mo Prof. **Diego Micheli**
Università degli Studi di Trieste
FIRMA: .....

RELATORE
Chiar.mo Prof. **Alberto Bartoli**
Università degli Studi di Trieste
FIRMA: .....

CORRELATORE
Chiar.mo Prof. **Eric Medvet**
Università degli Studi di Trieste
FIRMA: .....

Anno Accademico 2015/2016

Contents

Abstract

In the recent years, Machine Learning techniques have emerged as a new way to obtain solutions for a given problem, the novelty of the Machine Learning approach lies in the ability to automatically learn solutions by only looking at the observations of phenomena or examples of the expected behaviour. Machine learning methods are, in other words, able to generate models, rules or programs starting from a descriptive set of data for the given problem. Besides, Machine Learning techniques may be adopted when the problem exceeds the human ability to find out a solution and are, at the present time, a viable solution also in fields that were previously dominated by the human intelligence: language translation, image recognition, car driving, sentiment analysis, computer programming and also arts and creativity. At present time the Machine Learning tools are often a cost-effective alternative to employing human experts.

In this thesis we will describe the work developed at the Machine Learning Lab[1] at University of Trieste, consisting in novel Machine Learning techniques aimed at the solution of real world problems of practical interest: automatic synthesis of regular expressions for text extraction and text classification tasks; an approach for the continuous reauthentication of web users; design of algorithms for author verification for text documents; author profiling for text messages; automatic generation of fake textual reviews. Among them the main contribution of this thesis is the design and implementation of new algorithms for the automatic generation of regular expressions for text extraction, based solely on examples of the desired behavior [21, 23, 31]. This is a long-standing problem where we aim at generating a regular expression that generalizes the extraction behavior represented by some examples, i.e., strings annotated by a user with the desired portions to be extracted. The proposed algorithms are based on an evolutionary approach called Genetic Programming (GP), that is an evolutionary computing paradigm which implements an heuristic search, in a space of candidate solution, in a way that mimic the natural evolution process.

The results demonstrate that our new algorithms have higher effectiveness than previous proposals and demonstrate that our algorithms are able to generate regular expressions in a way that is competitive with human experts both in terms of effectiveness and generation time [24, 33]. Thanks to these achievements, the proposed method has been awarded with the Silver Medal at the 13th Annual "Humies" Award[2], an international competition that establishes the state of the art in genetic and evolutionary computation and is open to human-competitive results that are "equal to or better than the most recent human-created solution to a long-standing problem". The result of our research has been also released as an opensource framework[3] and as a web application demo[4] where users are free to provide text extraction examples to the application and obtain the corresponding regular expression.

Later in this thesis we will extend our work on automatic generation of regular expressions for text extraction from examples in order to operate in an Active learning scenario. In this scenario the user is not required to annotate all the examples at once but the Active learning tool interacts with the user in order to assist him during the annotation of the extractions in examples. We will propose our Active learning method [22, 26] that is based on our previous GP algorithms and the results will demonstrate that our

[1] http://machinelearning.inginf.units.it/
[2] http://gecco-2016.sigevo.org/index.html/Humies
[3] https://github.com/MaLeLabTs/RegexGenerator
[4] http://regex.inginf.units.it/

active learning tool reduces the user annotation effort while providing comparable effectiveness for the generated regular expressions.

Moreover, in this thesis we will consider two applications of the proposed regular expressions generator, adapted in order to cope with text categorization problems that are different from text extraction: (i) the Regex Golf game and (ii) the identification of Genic Interactions in sentences. The Regex Golf is a game where the player should write he shortest regular expression that accepts the strings in a positive set and does not accept strings in a negative set. We will show that our GP algorithm is able to play this game effectively and we will demonstrate that our algorithm is competitive with human players [20]. In the second case, we will consider the problem of automatically identifying sentences that contain interactions between genes and proteins inside a text document [30]. Our proposal requires solely a dictionary of genes and proteins and a small set of sample sentences in natural language. The proposed method generates a model in form of regular expressions that represents the relevant syntax patterns in terms of standard part-of-speech annotations. We will assess our approach on realistic datasets and show an accuracy that is sufficiently high to be of practical interest and that is in line with significant baseline methods.

The following contributions leave the field of the Genetic Programming algorithms and will propose solutions based on other Machine Learning methodologies, ranging from Grammatical Evolution to Support Vector Machines and Random Forests to Recurrent Neural Networks.

We will propose a methodology for *predicting* the accuracy of the text extractor [25] that may be inferred with the proposed GP method. We will employ several prediction techniques and the results suggest that reliable predictions for tasks of practical complexity may indeed be obtained quickly and without actually generating the entity extractor. Later, we will approach the problem of the automatic text extraction from another perspective and we will propose a novel learning algorithm that is able to generate a string similarity function tailored to problems of syntax-based entity extraction from unstructured text streams [27]. The proposed algorithm, based on an evolutionary paradigm named Grammatical Evolution, takes in input pairs of strings along with an indication of whether they adhere or not adhere to the same syntactic pattern. The results suggest that the proposed approach is indeed feasible and that the learned similarity function is more effective than the Levenshtein distance and the Jaccard similarity index. Hence, we will propose a system for continuous reauthentication of web users based on the observed mouse dynamics [144]; the key feature of our proposal is that no specific software needs to be installed on client machines. We obtain accuracy in the order of 97%, which is aligned with earlier proposals. Then, we will approach the user authentication problem [14], this task consists in determining if an unknown document was authored by the same author of a set of documents with the same author. Our methods has been submitted to the 2015 PAN competition and achieved the first position in the final rank for the Spanish language. Hence, we will approach the user profiling problem [19], this task consists in predicting some attributes of an author–i.e gender, age– analyzing a set of his/her Twitter tweets. We consider several sets of stylometric and content features, and different decision algorithms. Finally, we will investigate the feasibility of two tools capable of generating (i) fake reviews for a given scientific paper automatically [28] and (ii) fake consumer reviews for a restaurant automatically [29]. We experimentally assessed our methods on human subjects and the results highlight the ability of our methods to produce reviews that often look credible and may subvert the human decision.

Riassunto

Negli ultimi anni le tecniche di Machine Learning si sono affermate come un nuovo modo per trovare soluzioni ad un dato problema, la novità dell'approccio Machine Learning sta nella capacità di determinare le soluzioni dalle sole osservazioni di un fenomeno o da esempi del comportamento desiderato. In altre parole, i metodi di Machine Learning sono capaci di generare regole o programmi a partire da un insieme di dati descrittivi per un dato problema. Inoltre, i metodi di Machine Learning possono essere adottati quando il problema supera l'umana capacità di determinare una soluzione e sono, al momento, una soluzione praticabile anche in campi precedentemente dominati dall'intelligenza umana: traduzione, riconoscimento di immagini, guida di veicoli, analisi emotiva di testi (sentiment analysis), programmazione di computer e anche arte e creatività. Al giorno d'oggi gli strumenti di Machine Learning risultano spesso un'alternativa conveniente all'impiego di esperti umani.

In questa tesi descriveremo il lavoro svolto presso il Machine Learning Lab[5] dell'Università degli Studi di Trieste e che consiste in nuove tecniche di Machine Learning volte alla soluzione di problemi reali e di interesse pratico: la sintesi automatica di espressioni regolari finalizzate alla estrazione di testo o alla classificazione di testo; un approccio per la ri-autenticazione continua di utenti web; l'ideazione di algoritmi per la verifica dell'autore di documenti testuali; la profilazione di autore di messaggi testuali; la generazione automatica di false recensioni testuali. Fra queste, il contributo principale della tesi è il progetto e la realizzazione di nuovi algoritmi per la generazione automatica di espressioni regolari per l'estrazione di testo a partire da soli esempi del comportamento desiderato [21, 23, 31]. Questo è un problema di lunga data nel quale si desidera generare una espressione regolare che generalizzi il comportamento di estrazione rappresentato in alcuni esempi–i.e. stringhe annotate dall'utente evidenziando le porzioni da estrarre. Gli algoritmi proposti sono basati su un approccio evolutivo denominato Programmazione Genetica (GP), si tratta di un paradigma del calcolo evolutivo che implementa una ricerca euristica all'interno di uno spazio di soluzioni candidate, in una maniera che imita il processo di evoluzione naturale.

I risultati ottenuti dimostrano che il nostro algoritmo raggiunge un'efficacia migliore delle proposte precedenti e dimostrano che il nostro algoritmo è in grado di generare espressioni regolari tali da renderlo competitivo con gli umani esperti, sia in termini di efficacia che di tempo per la generazione [24,33]. Grazie a questi risultati il metodo proposto è stato premiato con la medaglia d'argento alla 13a competizione annuale "Humies"[6], una competizione internazionale che stabilisce lo stato dell'arte nel calcolo genetico ed evolutivo, aperta a risultati che sono competitivi con l'uomo ovvero che sono "uguali o migliori delle più recenti soluzioni create dall'uomo per un problema di lunga data". Il risultato della ricerca è stato anche rilasciato come framework opensource[7] e come una applicazione web demo[8] dove gli utenti possono fornire esempi testuali di estrazioni all'applicazione e ottenere la corrispondente espressione regolare.

[5] http://machinelearning.inginf.units.it/
[6] http://gecco-2016.sigevo.org/index.html/Humies
[7] https://github.com/MaLeLabTs/RegexGenerator
[8] http://regex.inginf.units.it/

Nel prosieguo della tesi illustreremo il nostro lavoro sulla generazione automatica di espressioni regolari per l'estrazione di testo da esempi in uno scenario Active Learning. In questo scenario non è richiesto all'utente di annotare in anticipo e completamente tutti gli esempi ma lo strumento Active Learning interagisce con l'utente, assistendolo durante la annotazione delle estrazioni negli esempi. Proporremo un metodo Active Learning [22,26] che è basato sui nostri precedenti algoritmi GP e i risultati dimostreranno che il nostro strumento active learning riduce lo sforzo di annotazione dell'utente pur mantenendo un'efficacia delle espressioni regolari comparabile.

Inoltre, in questa tesi considereremo due applicazioni dei proposti generatori di espressioni regolari, ma adattati in modo da far fronte a problemi di categorizzazione di testo invece che di estrazione di testo: (i) il gioco Regex Golf e (ii) l'identificazione di interazioni geniche in sequenze. Il Regex Golf è un gioco dove il giocatore deve scrivere l'espressione regolare più corta che accetti le stringhe in un insieme positivo e non accetti le stringhe in un insieme negativo. Mostreremo che il nostro algoritmo GP è effettivamente capace di affrontare questo gioco e dimostreremo che il nostro algoritmo è competitivo con i giocatori umani [20]. Nella seconda applicazione considereremo il problema della identificazione automatica all'interno di documenti testuali di frasi che contengono interazioni tra geni e proteine [30]. La nostra proposta richiede solamente un dizionario di geni e proteine e un piccolo campione di frasi in linguaggio naturale. Il metodo proposto genera un modello nella forma di espressione regolare che rappresenta i pattern sintattici rilevanti nella forma di annotazioni standard part-of-speech(POS). Verificheremo il nostro approccio su dei dataset realistici e dimostreremo una accuratezza che è in linea con importanti metodi di riferimento e sufficientemente alta da essere di interesse pratico.

I contributi che seguono, abbandonano il campo della Programmazione Genetica e propongono soluzioni basate su altre metodologie di Machine Learning, che spaziano dalla Grammatical Evolution alle Support Vector Machine e dalle Random Forests alle Recurrent Neural Networks.

Proporremo un metodo per la *predizione* della accuratezza per gli estrattori di testo [25] che vengono generati dai metodi GP precedentemente proposti. Utilizzeremo diverse tecniche di predizione e i risultati suggeriranno che predizioni affidabili, per problemi di complessità pratica, possono essere effettivamente ottenuti, velocemente e senza in realtà generare l'estrattore di entità testuali. Nel prosieguo affronteremo il problema della estrazione automatica di testo da un'altra prospettiva e proporremo un nuovo algoritmo in grado di generare una funzione di similarità tra stringhe ottimizzata per l'estrazione di entità da un flusso di testo non strutturato [27]. Di seguito proporremo un sistema per la ri-autenticazione continua degli utenti web basata sulle dinamiche osservate del mouse [144]; la caratteristica chiave della nostra proposta è che non necessita l'installazione di alcun software sulle macchine client. Abbiamo ottenuto un'accuratezza dell'ordine del 97% che è allineato con le proposte precedenti. Successivamente, affronteremo il problema della verifica dell'autore [14], questo problema consiste nel determinare se un documento sconosciuto è stato scritto dallo stesso autore di un insieme di documenti noti di uno stesso autore. Il nostro metodo è stato inviato alla competizione PAN 2015 e ha conseguito il primo posto nella graduatoria finale per la lingua spagnola. Affronteremo quindi il problema di profilazione di un utente [19], questo problema consiste nel predire alcune caratteristiche di un autore–i.e sesso, età– analizzando un insieme dei suoi tweets di Twitter. Consideriamo molti insiemi di feature stilometriche e di contenuto, e differenti algoritmi di decisione. Infine, investigheremo la fattibilità di due strumenti capaci di generare automaticamente (i) false review per un dato articolo scientifico [28] e (ii) false recensioni di clienti per un ristorante [29]. Abbiamo verificato sperimentalmente i nostri metodi su soggetti umani che hanno evidenziato l'abilità del nostro metodo di produrre recensioni che risultano spesso credibili e in grado di sovvertire la decisione finale del lettore umano.

Chapter 1

Introduction

The machine learning methods allow the computer to automatically generate programs starting from a set of data that represents the expected behaviour, in other words machine learning methods can automatically infer knowledge only by looking at examples of the expected behaviour or observations of phenomena. In recent years, the machine learning techniques have evolved in a way that are currently able to perform some tasks with the same effectiveness and greater reliability than the human being, for example, machine learning techniques are now employed in applications like language translation, fraud detection, drug design, recommender system, medical diagnoses, assisted and autonomous car driving, image and face recognition. In fact machine learning tools have proven to be a cost-effective and reliable alternative to human intelligence on a broad set of problems.

An effective machine learning technique is Genetic Programming (GP) which is a framework inspired by biological evolution [120]. With GP, a solution for a problem is represented as a computer program that is usually encoded as a tree structure–or abstract syntax tree, at start, the GP algorithm randomly generates a *population* of such computer programs, composed only by a predefined set of building blocks, each program is called an *individual*. Given a set of examples of the expected program behaviour, the GP algorithm assess the effectiveness of each individual by evaluating a *fitness function* which returns a number that represents the ability of the individual to solve the problem of interest. Hence, individuals are randomly selected from the current population and recombined through certain genetic operators called "crossover" and "mutation" in order to generate new ones, these new individuals are inserted in a new population, a key point of this process is the individual selection that favors individuals that exhibit a better fitness. The population obtained at each iteration of this process is called *generation*, and this process is iterated until a satisfactory solution is found or until some termination criterion is met.

In this thesis we will describe the work developed at the Machine Learning Lab[1] at University of Trieste, consisting in novel machine learning techniques aimed at the solution of real world problems of practical interest. The main contribution of this thesis is the application of GP for the automatic inference of regular expressions both for text extraction [20, 21, 22, 23, 26, 30, 31] and classification tasks. Our proposals improve over existing state-of-the art and the results shown in this thesis demonstrate that our GP tools are able to solve these complex problems with the same effectiveness than human experts and in a comparable amount of time [24, 33]; these results have been published in international conferences and journals and have been awarded with the Silver Medal at the 13th Annual "Humies" Award[2], an international competition that establishes the state of the art in genetic and evolutionary computation.

[1] http://machinelearning.inginf.units.it/

[2] http://gecco-2016.sigevo.org/index.html/Humies

1.1 Thesis outline

A large class of entity extraction tasks from text that is either semistructured or fully unstructured may be addressed by regular expressions, because in many practical cases the relevant entities follow an underlying syntactical pattern and this pattern may be described by a regular expression. Writing a regular expression capable of guaranteeing high precision and recall for a given extraction task is tedious, difficult and requires specific technical skills.

In chapter 2 we consider the long-standing problem of synthesizing such expressions automatically, based solely on examples of the desired behavior. We present the design and implementation of a system capable of addressing extraction tasks of realistic complexity [21, 23, 31]. Our system is based on an evolutionary procedure carefully tailored to the specific needs of regular expression generation by examples. The procedure executes a search driven by a multiobjective optimization strategy aimed at simultaneously improving multiple performance indexes of candidate solutions while at the same time ensuring an adequate exploration of the huge solution space. We assess our proposal experimentally in great depth, on a number of challenging datasets. The accuracy of the obtained solutions seems to be adequate for practical usage and improves over earlier proposals significantly. Most importantly, our results are highly competitive even with respect to human operators. A prototype is available as a web application at http://regex.inginf.units.it.

As mentioned, building a regular expression involves a considerable amount of skill, expertise and creativity, in chapter 3 we investigate whether the GP algorithm, proposed in chapter 2, may effectively surrogate these qualities and construct automatically regular expressions, for tasks of realistic complexity, in a way that is competitive with expert human users [24, 33]. We performed a large scale experiment involving more than 1700 users on 10 challenging tasks [33]. We compared the solutions constructed by these users to those constructed by our tool based on Genetic Programming. This large-scale experiment confirmed the preliminary results of chapter 2 the quality of automatically-constructed solutions turned out to be similar to the quality of those constructed by the most skilled user group and the time for automatic construction was similar to the time required by human users.

Hence in chapter 4 we study the automatic generation of regular expressions for text extraction from examples in an active learning scenario in which the user annotates only one desired extraction and then merely answers extraction queries generated by the system. The resulting framework [22, 26] is attractive because it is the system, not the user, which digs out the data in search of the samples most suitable to the specific learning task. We base our proposals on our state-of-the-art GP learner—described in chapter 2—based on Genetic Programming and we assess them experimentally on a number of challenging tasks of realistic complexity. The results indicate that active learning is indeed a viable framework in this application domain and may thus significantly decrease the amount of costly annotation effort required.

In chapter 5 we propose a methodology for predicting the accuracy of the extractor that may be inferred with the methods proposed in chapter 2. We propose several prediction techniques and analyze experimentally our proposals in great depth [25], with reference to the extractors consisting of regular expressions. The results suggest that reliable predictions for tasks of practical complexity may indeed be obtained quickly and without actually generating the entity extractor.

In the following two chapters we apply our, GP based, regex inference engine at two text classification problems. In chapter 6 we develop an artificial player [20] for the Regex Golf game; Regex golf has recently emerged as a specific kind of code golf, i.e., unstructured and informal programming competitions aimed at writing the shortest code solving a particular problem. A problem in regex golf consists in writing the shortest regular expression which matches all the strings in a given list and does not match any of the strings in another given list. The regular expression is expected to follow the syntax of a specified programming language, e.g., Javascript or PHP. We propose a *regex golf player* internally based on a variant of the Genetic Programming regex inference engine described in the paper [16] that is an ancestor of the one we describe in chapter 2. We assess experimentally our player on a popular regex golf challenge consisting of 16 problems and compare our results against those of a recently proposed algorithm—the only one we are aware of. Our player obtains scores which improve over the baseline and are highly competitive also with respect to human players. The time for generating a solution is usually in the order

of tens minutes, which is arguably comparable to the time required by human players.

In chapter 7 we look at the development of techniques for automatic relation extraction from unstructured text, which is a problem attracting growing interest. The biomedical domain, in particular, is a sector that may greatly benefit from those techniques due to the huge and ever increasing amount of scientific publications describing observed phenomena of potential clinical interest. In our work, we consider the problem of automatically identifying sentences that contain interactions between genes and proteins, based solely on a dictionary of genes and proteins and a small set of sample sentences in natural language. We propose an evolutionary technique for learning a classifier that is capable of detecting the desired sentences within scientific publications with high accuracy. The key feature of our proposal [30], that is internally based on Genetic Programming, is the construction of a model of the relevant syntax patterns in terms of standard part-of-speech annotations. The model consists of a set of regular expressions that are learned automatically despite the large alphabet size involved. We assess our approach on two realistic datasets and obtain 74% accuracy, a value sufficiently high to be of practical interest and that is in line with significant baseline methods.

Hence, in chapter 8 we approach the syntax-based entity extraction from examples from a different perspective and we propose a similarity learning algorithm tailored to problems of syntax-based entity extraction from unstructured text streams. Several research efforts have shown that a similarity function synthesized from examples may capture an application-specific similarity criterion in a way that fits the application needs more effectively than a generic distance definition. The proposed algorithm takes in input pairs of strings along with an indication of whether they adhere or not adhere to the same syntactic pattern. Our approach [27] is based on Grammatical Evolution and explores systematically a similarity definition space including all functions that may be expressed with a specialized, simple language that we have defined for this purpose. We assessed our proposal on patterns that are a subset of ones employed in chapter 2. The results suggest that the proposed approach is indeed feasible and that the learned similarity function is more effective than the traditional Levenshtein distance and the Jaccard similarity index.

In the final chapters we leave entity extraction and text classification problems to face different challenges like (i) the continuous reauthentication of web users based on the observed mouse movements, (ii) the authentication of the author for text documents, (iii) the prediction of attributes–i.e: age, gender–for an author of Twitter messages, (iv) the automatic generation of fake scientific reviews and (v) the automatic generation of fake restaurant reviews

In chapter 9 we propose a system for continuous reauthentication of web users based on the observed mouse dynamics [144]. Key feature of our proposal is that no specific software needs to be installed on client machines, which allows to easily integrate continuous reauthentication capabilities into the existing infrastructure of large organizations. We assess our proposal with real data from 24 users, collected during normal working activity for several working days. We obtain accuracy in the order of 97%, which is aligned with earlier proposals requiring instrumentation of client workstations for intercepting all mouse activity—quite a strong requirement for large organizations. Our proposal may constitute an effective layer for a defense-in-depth strategy in several key scenarios: web applications hosted in the cloud, where users authenticate with standard mechanisms; organizations which allow local users to access external web applications, and enterprise applications hosted in local servers or private cloud facilities.

In chapter 10 we describe an approach for the author identification task [14]. The task consists in determining if an unknown document was authored by the same author of a set of documents with the same author. We propose a machine learning approach based on a number of different features that characterize documents from widely different points of view. We construct non-overlapping groups of homogeneous features, use a random forest regressor for each features group, and combine the output of all regressors by their arithmetic mean. We train a different regressor for each language. Our approach achieved the first position in the final rank for the Spanish language in the 2015 PAN competition [196].

In chapter 11 we describe an approach for the author profiling task [19]. The task consists in predicting some attributes of an author analyzing a set of his/her Twitter tweets. We consider several sets of stylometric and content features, and different decision algorithms: we use a different combination of features and decision algorithm for each language-attribute pair, hence treating it as an individual problem.

Hence, in chapter 12 we investigate the feasibility of a tool capable of generating fake reviews for a given scientific paper automatically [28]. While a tool of this kind cannot possibly deceive any rigorous editorial procedure, it could nevertheless find a role in several questionable scenarios and magnify the scale of scholarly frauds. Peer review is widely viewed as an essential step for ensuring scientific quality of a work and is a cornerstone of scholarly publishing. On the other hand, the actors involved in the publishing process are often driven by incentives which may, and increasingly do, undermine the quality of published work, especially in the presence of unethical conduits. A key feature of our tool is that it is built upon a small knowledge base, which is very important in our context due to the difficulty of finding large amounts of scientific reviews. We experimentally assessed our method with 16 human subjects and we presented to these subjects a mix of genuine and machine generated reviews and we measured the ability of our proposal to actually deceive subjects judgment. The results highlight the ability of our method to produce reviews that often look credible and may subvert the decision. Moreover, we like to emphasize that the scientific article [28] that describes this work, has also attracted much interest on magazines[3].

In the last chapter 13 we explore the feasibility of a tool capable of generating fake reviews for a restaurant automatically [29]. Consumer reviews are an important information resource for people and a fundamental part of everyday decision-making. Product reviews have an economical relevance which may attract malicious people to commit a review fraud, by writing false reviews. In this work, we investigate the possibility of generating hundreds of false restaurant reviews automatically and very quickly. We propose and evaluate a method for automatic generation of restaurant reviews tailored to the desired rating and restaurant category. A key feature of our work is the experimental evaluation which involves human users. We assessed the ability of our method to actually deceive users by presenting to them sets of reviews including a mix of genuine reviews and of machine-generated reviews. Users were not aware of the aim of the evaluation and the existence of machine-generated reviews. As it turns out, it is feasible to automatically generate realistic reviews which can manipulate the opinion of the user.

[3] https://www.timeshighereducation.com/news/robot-written-reviews-fool-academics, http://retractionwatch.com/2016/09/02/weve-seen-computer-generated-fake-papers-get-published-now-we-have-computer-generated-fake-peer-reviews/,http://www.powerlineblog.com/archives/2016/09/academic-absurdity-of-the-week-fake-peer-reviews.php?,https://www.insidehighered.com/news/2016/09/22/many-academics-are-fooled-robot-written-peer-reviews

1.2 Publication list

- [31] A. Bartoli, A. D. Lorenzo, E. Medvet, and F. Tarlao. Data quality challenge: Toward a tool for string processing by examples. *Journal of Data and Information Quality (JDIQ)*, 6(4):13, 2015

- [23] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Inference of regular expressions for text extraction from examples. *IEEE Transactions on Knowledge and Data Engineering*, 28(5):1217–1230, 2016

- [25] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Predicting the effectiveness of pattern-based entity extractor inference. *Applied Soft Computing*, 46:398–406, 2016

- [33] A. Bartoli, E. Medvet, A. D. Lorenzo, and F. Tarlao. Can a machine replace humans in building regular expressions? a case study. *IEEE Intelligent Systems*, PP(99):1–1, 2016

- [26] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Regex-based entity extraction with active learning and genetic programming. *ACM SIGAPP Applied Computing Review*, 16(2):7–15, 2016

- [29] A. Bartoli, A. De Lorenzo, E. Medvet, F. Tarlao, and D. Morello. "best dinner ever!!!": Automatic generation of restaurant reviews with lstm-rnn. In *2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. IEEE, 2016

- [20] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Playing regex golf with genetic programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 1063–1070. ACM, 2014

- [144] E. Medvet, A. Bartoli, F. Boem, and F. Tarlao. Continuous and non-intrusive reauthentication of web sessions based on mouse dynamics. In *Availability, Reliability and Security (ARES), 2014 Ninth International Conference on*, pages 166–171. IEEE, 2014

- [21] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Learning text patterns using separate-and-conquer genetic programming. In *European Conference on Genetic Programming*, pages 16–27. Springer, 2015

- [14] A. Bartoli, A. Dagri, A. De Lorenzo, E. Medvet, and F. Tarlao. An Author Verification Approach Based on Differential Features—Notebook for PAN at CLEF 2015. In Cappellato et al. [171]

- [30] A. Bartoli, A. De Lorenzo, E. Medvet, F. Tarlao, and M. Virgolin. Evolutionary learning of syntax patterns for genic interaction extraction. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1183–1190. ACM, 2015

- [19] A. Bartoli, A. De Lorenzo, A. Laderchi, E. Medvet, and F. Tarlao. An author profiling approach based on language-dependent content and stylometric features. In *Proceedings of CLEF*, 2015

- [22] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Active learning approaches for learning regular expressions with genetic programming. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 97–102. ACM, 2016

- [24] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. On the automatic construction of regular expressions from examples (gp vs. humans 1-0). In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 155–156. ACM, 2016

- [28] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Your paper has been accepted, rejected, or whatever: Automatic generation of scientific paper reviews. In *International Conference on Availability, Reliability, and Security*, pages 19–28. Springer, 2016

- [27] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Syntactical similarity learning by means of grammatical evolution. In *International Conference on Parallel Problem Solving from Nature*, pages 260–269. Springer, 2016

Chapter 2

Inference of Regular Expressions for Text Extraction

2.1 Overview

A regular expression is a means for specifying string patterns concisely. Such a specification may be used by a specialized engine for extracting the strings matching the specification from a data stream. Regular expressions are a long-established technique for a large variety of application domains, including text processing, and continue to be a routinely used tool due to their expressiveness and flexibility. A large class of entity *extraction* tasks, in particular, may be addressed by regular expressions, because in many practical cases the relevant entities follow an underlying syntactical pattern and this pattern may be described by a regular expression. However, the construction of regular expressions capable of guaranteeing high precision and high recall for a given extraction task is tedious, difficult and requires specific technical skills.

In this chapter, we consider the problem of synthesizing a regular expression *automatically*, based solely on *examples* of the desired behavior. This problem has attracted considerable interest, since a long time and from different research communities. A wealth of research efforts considered *classification* problems in *formal* languages [45, 77, 78, 84, 117, 199]—those results are not immediately useful for text extraction. Essentially, the problem considered by those efforts consisted in inferring an acceptor for a regular language based on positive and negative sample strings, i.e., of strings described by the language and of strings not described by the language. Learning of *deterministic finite automata (DFA)* from examples was also a very active area, especially because of competitions that resulted in several important insights and algorithms, e.g. [60, 126]. Such research, however, usually considered problems that were not inspired by any real world application [60] and the applicability of the corresponding learning algorithms to other application domains is still largely unexplored [40]. For example, the so-called Abbadingo competition was highly influential in this area and considered short sequences of binary symbols, with training data drawn uniformly from the input space. Settings of this sort do not fit the needs of practical text processing applications, which have to cope with much longer sequences of symbols, from a much larger alphabet, not drawn uniformly from the space of all possible sequences. Furthermore, regular expressions used in modern programming languages allow specifying more various extraction tasks than those which can be specified using a DFA.

A text extraction problem was addressed by researchers from IBM Almaden and the University of Michigan, which developed a procedure for improving an initial regular expression to be provided by the user based on examples of the desired functioning [133]. The cited work is perhaps the first one addressing entity extraction from real text of non trivial size and complexity: the entities to be extracted included software names, email addresses and phone numbers while the datasets were unstructured and composed of many thousands of lines. A later proposal by researchers from IBM India and Chennai Mathematical Institute still required an initial regular expression but was more robust toward initial expressions of

modest accuracy and noisy datasets [10]. Refinement of a given regular expression was also considered by an IBM Research group, which advocated involvement of a human operator for providing feedback during the process [152]. The need of an initial solution was removed by researchers from SAP AG that demonstrated the practical feasibility of inferring a regular expression from scratch, based solely on a set of examples derived from enterprise data, such as, e.g., a product catalog or historical invoices [44]. A more recent proposal of ours has obtained further significant improvements in this area, in terms of precision and recall of the generated solutions as well as in terms of smaller amount of training data required [15, 17]. Regular expressions for text extraction tasks of practical complexity may now be obtained in a few minutes, based solely on a few tens of examples of the desired behavior.

In this work we present a system that aims at improving the state-of-the-art in this area. Our proposal is internally based on *Genetic Programming (GP)*, an evolutionary computing paradigm which implements a heuristic search in a space of candidate solutions [120]. We execute a search driven by a *multiobjective optimization* strategy aimed at simultaneously improving multiple performance indexes of candidate solutions while at the same time ensuring an adequate exploration of the huge solution space. Our proposal is a significant improvement and redesign of the approach in [17], resulting in a system that generates solutions of much better accuracy. The improvements include: (a) a radically different way of quantifying the quality of candidate solutions; (b) inclusion, in the starting points of the search, of candidate solutions built based on an analysis of the training data, rather than being fully random; (c) a strategy for restricting the solution space by defining potentially useful "building blocks" based on an analysis of the training data; and (d) a simple mechanism for enforcing structural diversity of candidate solutions.

Furthermore, the redesign features several novel properties which greatly broaden the scope of extraction tasks that may be addressed effectively:

- *Support for the or operator*. In many cases learning a single pattern capable of describing all the entities to be extracted may be very difficult—e.g., dates may be expressed in a myriad of different formats. Our system is able to address such scenarios by generating several regular expressions that are all joined together with *or* operators to form a single, larger regular expression. We implement this functionality by means of a *separate-and-conquer* procedure [13, 91, 163]. Once a candidate regular expression provides adequate accuracy on a subset of the examples, the expression is inserted into the set of final solutions and the learning process continues on a smaller set of examples including only those not yet solved adequately [21]. The key point is that the system is able to realize automatically how many regular expressions are needed.

- *Context-dependent extraction*. It is often the case that a text snippet must or must not be extracted depending on the text surrounding the snippet—e.g., an email address might have to be extracted only when following a Reply-To: header name. Modern regular expression engines provide several constructs for addressing these needs but actually taking advantage of those constructs is very challenging: the more the available constructs, the larger the search space. Our system is able to generate regular expressions which exploit *lookaround* operators effectively, i.e., operators specifying constraints on the text that precedes or follows the text to be extracted.

- *No constraints on the size of training examples*. We place no constraints on the size of training examples: the training data may consist of either a single, potentially very large, file with an annotation of all the desired extractions, or of a set of lines with zero or more extractions in each one. This seemingly minor detail may in fact be quite important in practice: the cited work [17] was not able to exploit training examples including multiple extractions correctly (this point will be discussed in detail later), thus the training data had to be segmented in units containing at most one extraction and in such a way that desired extractions did not span across adjacent units. The need for such a tricky operation is now removed. Accommodating the possibility of multiple extractions in each training example has required significant changes in the search strategy internally used by the system.

We assess our proposal experimentally in great depth, on a number of challenging datasets of realistic complexity and with a very small portion of the dataset available for learning. We compare precision and

recall of the regular expressions generated by our system to significant baseline methods proposed earlier in the literature. The results indicate a clear superiority of our proposal and the obtained accuracy values seem to be adequate for practical usage. Our results are highly competitive also with respect to a pool of more than 70 human operators, both in terms of accuracy and of time required for building a regular expression. Indeed, we are not aware of any proposal for automatic generation of regular expressions in which human operators were used as a baseline.

We made publicly available the source code of our system (https://github.com/MaLeLabTs/RegexGenerator) and deployed an implementation as a web app (http://regex.inginf.units.it).

2.2 Related work

In this section we discuss further proposals that, beyond those already discussed in the overview, may be useful to place our work in perspective with respect to the existing literature. As pointed out by [133], the learning of regular expressions for information extraction prior to the cited work focused on scenarios characterized by alphabet sizes much smaller than those found in natural language text. Rather than attempting to infer patterns over the text to be extracted, the usual approach consisted on learning patterns over *tokens* generated with various text processing techniques, e.g., POS tagging, morphological analysis, gazetteer matching [61, 177, 209].

An attempt at learning regular expressions over real text was proposed in [55]. The cited work considered reduced forms of regular expressions (a small subset of POSIX rules) and, most importantly, considered a simple classification problem consisting in the detection of HTML lines with a link to other web documents. Text classification and text extraction are related but different problems, though. The former assumes an input stream segmented in units to be processed one at a time; one has to detect whether the given input unit contains at least one interesting substring. The latter requires instead the ability to identify, in the (possibly very long) input stream, the boundaries of all the relevant substrings, if any. Furthermore, text extraction usually requires the ability to identify a context for the desired extraction, that is, a given sequence of characters may or may not have to be extracted depending on its surroundings. Interestingly, the approach in [17] was developed for extraction but delivered better results than in [55] also in classification.

Further proposals for addressing classification problems have been developed but tailored to very specific scenarios, recent examples include email spam campaigns [168, 210] and clinical symptoms [48].

There have been other proposals for regular expression learning aimed at information extraction from real text, specifically web documents [12]. The cited work provides an accuracy in URL extraction from real web documents that is quite low—the reported value for F-measure being 27% (on datasets that are not public). In this respect, it is useful to observe that the latest proposal [17] obtained accuracy well above 90% in the 12 datasets considered; moreover, two of those datasets were used also in [44, 133] and in those cases it obtained similar or much better accuracy with a training set smaller by an order of magnitude.

The problem of learning a regular expression by examples of the desired extraction behavior could be seen as a very specific problem in the broader category of *programming by examples*, where a program in a given programming language is to be synthesized based on a set of input-output pairs [190]. In particular, the problem is an under-specified task [62] in the sense that there may usually be many different solutions whose behavior on the training data is identical while their behavior on unseen data is different. The cited work considers the generation of regular expressions for classification tasks on phone numbers, dates, email addresses and URLs—tasks that are considered to be tricky even for expert developers and to lack an easy-to-formalize specification. It advocates the writing of solutions by several expert developers based on some examples, an assessment of their behavior on unseen data made in *crowd-sourcing*, and an evolutionary optimization of the available solutions based on the feedback from the crowd. Our proposal generates a regular expression in a fully automatic way. Furthermore, we assess our work on datasets that are orders of magnitude larger than those considered in [62] and on tasks that seems fair to define much more challenging. Of course, we make these observations in the attempt of clarifying our proposal

and by no means we intend to criticize the cited work: besides, the cited work investigates the possibility of crowd-sourcing difficult programming tasks and is *not* meant to propose a method for the automatic generation of regular expressions from examples. It is useful to observe, though, that the authors of the cited work were not aware of any approach suitable for learning regular expressions capable of handling the large alphabet sizes occurring in real-world text files, while such functionality was demonstrated in [15, 17, 44].

As pointed out above, learning a program from examples of the desired behavior is an intrinsically under-specified task—there might be many different solutions with identical behavior over the examples. Furthermore, in practice, there is usually not even any guarantee that a solution which perfectly fits all the examples actually exists. The common approach for addressing this issue, which is also our approach, aims at an heuristic balance between generalization and overfitting: we attempt to infer from the examples what is the actual desired behavior, without insisting on obtaining perfect accuracy on the training set. It may be worth mentioning that coding challenges exist (and occasionally become quite popular in programming forums) which are instead aimed at overfitting a list of examples [20, 157]. The challenge[1] consists in writing the shortest regular expression that matches all strings in a given list and does not match any string in another given list. Our proposal is not meant to address these scenarios. From the point of view of our discussion, scenarios of this sort differ from text extraction in several crucial ways. First, they are a classification problem rather than an extraction problem. Second, they place no requirements on how strings not listed in the problem specification should be classified—e.g., strings in the problem specification followed or preceded by additional characters. Text extraction requires instead a form of generalization, i.e., the ability of inducing a general pattern from the provided examples.

Finally, we mention a recent proposal for information extraction from examples [129]. The cited work describes a powerful and sophisticated framework for extracting multiple different fields automatically in semi-structured documents. As such, the framework encompasses a much broader scenario than our work. A tool implementing this framework has been publicly released as part of Windows Powershell[2]. The tool does not generate a regular expression; instead, it generates a program in a specified algebra of string processing operators that is to be executed by a dedicated engine. We decided to include this tool in our experimental evaluation in order to obtain further insights into our results.

2.3 Scenario

We are concerned with the task of generating a regular expression which can generalize the extraction behavior represented by some examples, i.e., by strings annotated with the desired portions to be extracted. In this section we define the problem statement in detail along with the notation which will be used hereafter.

We focus on the regular expression implementation which is provided by the Java standard libraries. A deep comparison of different flavours of regular expressions is beyond the scope of our work [89], yet it is worth to mention that Java regular expressions provide more constructs than POSIX extended regular expressions (ERE)—e.g., lookarounds (see Section 2.4.1)—which allow to define patterns in a more compact form.

2.3.1 Definitions

A *snippet* x_s of a string s is a substring of s, identified by the starting and ending index in s. For readability, we refer to snippets using their textual content followed by their starting index as subscript—e.g., ex_5, extra_5 and $\mathsf{traction}_7$, are three different snippets of the string $\mathsf{text\_extraction}$. We denote by $\mathcal{X}_s$ the set of all the snippets of s. Let $x_s, x'_s \in \mathcal{X}_s$. A total order is defined among snippets in $\mathcal{X}_s$ based on their starting index: x_s *precedes* x'_s if the starting index of the former is strictly lower than the starting index of the latter. We say that x_s is a *supersnippet* of x'_s if the indexes interval of x_s strictly contains the indexes

[1] https://www.google.it/search?q=regex+golf

[2] Windows Management Framework 5.0 Preview, November 2014.

interval of x'_s: in this case, x'_s is a *subsnippet* of x_s. Finally, we say that x_s *overlaps* x'_s if the intersection of their index intervals is not empty. For instance, ex_1, ex_5, extra_5 and $\mathsf{traction}_7$, are snippets of the string $\mathsf{text\_extraction}$: extra_5 is a supersnippet of ex_5 (but not of ex_1), extra_5 precedes and overlaps $\mathsf{traction}_7$.

A regular expression r applied on a string s deterministically extracts zero, one or more snippets. We denote the (possibly empty) set of such snippets, that we call *extractions*, by $[\mathcal{X}_s]_r$.

2.3.2 Problem statement

The problem input consists of a set of *examples*, where an example (s, X_s) is a string s associated with a (possibly empty) set of non-overlapping snippets $X_s \subset \mathcal{X}_s$. String s may be, e.g., a text line, or an email message, or a log file and so on. Set X_s represents the *desired extractions* from s, whereas snippets in $\mathcal{X}_s \setminus X_s$ are *not* to be extracted.

Intuitively, the problem consists in learning a regular expression $\hat{r}$ whose extraction behavior is consistent with the provided examples—$\hat{r}$ should extract from each string s only the desired extractions X_s. Furthermore, $\hat{r}$ should capture the *pattern* describing the extractions, thereby *generalizing* beyond the provided examples. In other words, the examples constitute an incomplete specification of the extraction behavior of an ideal and unknown regular expression $r^\star$. The learning algorithm should aim at inferring the extraction behavior of $r^\star$ rather than merely obtaining from the example strings exactly the desired extractions. We formalize this intuition as follows.

Let E and $E^\star$ be two different sets of examples, both representing the extraction behavior of a target regular expression $r^\star$. The problem consists in learning, from *only* the examples in E, a regular expression $\hat{r}$ which maximizes its F-measure on $E^\star$, i.e., the harmonic mean of precision and recall w.r.t. the desired extractions from the examples in $E^\star$:

$$\mathrm{Prec}(\hat{r}, E^\star) := \frac{\sum_{(s,X_s)\in E^\star} |[\mathcal{X}_s]_{\hat{r}} \cap X_s|}{\sum_{(s,X_s)\in E^\star} |[\mathcal{X}_s]_{\hat{r}}|}$$

$$\mathrm{Rec}(\hat{r}, E^\star) := \frac{\sum_{(s,X_s)\in E^\star} |[\mathcal{X}_s]_{\hat{r}} \cap X_s|}{\sum_{(s,X_s)\in E^\star} |X_s|}$$

The greater the F-measure of $\hat{r}$ on $E^\star$, the more similar the extraction behaviour of $\hat{r}$ and $r^\star$.

We call the pair of sets of examples $(E, E^\star)$ a *problem instance*. In our experimental evaluation we built several problem instances starting from quite complex target expressions $r^\star$ and strings consisting of real world datasets (e.g., logs, HTML lines, Twitter posts, and alike). Of course, in a practical deployment of the system set $E^\star$ is not available because the target expression $r^\star$ is not known.

Observations on the problem statement

We point out that characterizing the features of a problem instance which may impact the quality of a generated solution is beyond the scope of our work. Assessing the difficulty of a given problem instance, either in general or when solved by a specific approach, is an important theoretical and practical problem. Several communities have long started addressing this specific issue, e.g., in information retrieval [66, 104] or in pattern classification [54, 110]. Obtaining practically useful indications, though, is still a largely open problem, in particular, in evolutionary computing [96] as well as in more general search heuristics [191, 192].

A notable class of problem instances is the one which we call *with context*. Intuitively, these are the problem instances in which a given sequence of characters is the textual content of snippet to be extracted and also the textual content of a snippet which is not to be extracted. For example, consider a problem instance with the two examples $(\mathsf{I\_have\_12\_dogs}, \varnothing)$ and $(\mathsf{Today\_is\_7\text{-}12\text{-}11}, \{12_{11}\})$. This problem instance is with context because the sequence of characters 12 is not to be extracted from the first example but is to be extracted from the second example. The discriminant between the two cases is in the portion of the string surrounding the sequence 12, that is, in its context. Of course, similar scenarios could occur with respect to sequences of characters in the same example rather than in different examples—e.g., assuming

an email message is an example, one might want to extract only the email addresses following a Reply-To: header name.

2.4 Our approach

Our approach is based on *Genetic Programming* (GP) [120]. GP is an evolutionary computing paradigm in which candidate solutions for a target problem, called *individuals*, are encoded as trees. A problem-dependent numerical function, called *fitness*, must be defined in order to quantify the ability of each individual to solve the target problem. This function is usually implemented by computing a performance index of the individual on a predefined set of problem instances, called the *learning set*. A GP execution consists of an heuristic and stochastic search in the solution space, looking for a solution with optimal fitness. To this end, an initial population of individuals is built, usually at random, and an iterative procedure is performed which consists in (i) building new individuals from existing ones using genetic operators (usually *crossover* and *mutation*), (ii) adding new individuals to the population, and (iii) discarding worst individuals. The procedure is repeated a predefined number of times or until a predefined condition is met (e.g., a solution with perfect fitness is found).

We carefully adapted the general framework outlined above to the specific problem of regular expression generation from examples. Our GP procedure is built upon our earlier proposal [17]—the numerous improvements were listed in the introduction. We describe this procedure in detail in the next sections: encoding of regular expressions as trees (Section 2.4.1), fitness definition (Section 2.4.1), construction of the initial population and its evolution for exploring the solution space (Section 2.4.1). Next, we describe our separate-and-conquer strategy (Section 2.4.1) and the overall organization of GP searches (Section 2.4.2).

2.4.1 GP search

We designed a *GP search* which takes a *training set* $\mathcal{T}$ as input and outputs a regular expression $\hat{r}$. The training set is composed of tuples (s, X_s^d, X_s^u), the components of each tuple being as follows: (i) a string s; (ii) a set of snippets X_s^d representing the desired extractions from s; (iii) a set of snippets X_s^u representing the undesired extractions from s, i.e., no snippet of s overlapping a snippet in X_s^u should be extracted. The training set $\mathcal{T}$ must be constructed such that $\forall s \in \mathcal{T}$ (i) $X_s^d \cap X_s^u = \varnothing$, and, (ii) snippets in $X_s^d \cup X_s^u$ must not overlap each other. The goal of a GP search is to generate a regular expression r such that $\forall s \in \mathcal{T}, X_s^d = [\mathcal{X}_s]_r$. We recall that, from a broader point of view, the generated regular expression r should generalize beyond the examples in $\mathcal{T}$ (see Section 2.3.2).

Tree representation

In our proposal an individual is a tree which represents a regular expression r. Each node in a tree is associated with a *label*, which is a string representing basic components of a regular expressions that are available to the GP search (discussed in detail below). Labels of non-leaf nodes include the placeholder symbol •: each children of a node is associated with an occurrence of symbol • in the label of that node. The regular expression represented by a tree is the string constructed by means of a depth-first post-order visit of the tree. In detail, we execute a *string transformation* of the root node of that tree. The string transformation of a node is a string obtained from the node label where each • symbol is replaced by the string transformation of the associated child. Figure 2.1 shows two examples of tree representations of regular expressions.

Available labels are divided in two sets: a set of predefined labels which represent regular expression constructs, and a set of $\mathcal{T}$-dependent labels constructed as described below. In other words, unlike the previous work in [17], the GP search explores a space composed of candidate solutions assembled from general regular expression constructs and from components constructed before starting the GP search by analyzing the provided examples. The rationale for $\mathcal{T}$-dependent labels consists in attempting to shrink the size of the solution space by identifying those sequences of characters which occurs often in

the desired extractions (or "around" them) and making these sequences available to the GP search as unbreakable building blocks. For instance, in the task of generating a regular expression for extracting URLs, the string http could be an useful such block.

Predefined labels are the following: character classes (\d, \w), predefined ranges (a-z, A-Z), digits (0, . . . , 1), predefined characters (\␣, :, ␣, ;, ␣, =, ", ', \\, /, \?, \!, \}, \{, \(, \), \[, \], <, >, @, #, ␣), concatenator (••), set of (un)possible matches ([•], [^•]), possessive quantifiers (•*+, •++, •?+, •{•,•}+), non-capturing group ((?:•)), and lookarounds ((?<=•), (?<!•), (?=•), (?!•)). We include possessive quantifiers and we do not include greedy and lazy quantifiers[3] because greedy and lazy quantifiers have worst-case exponential complexity, which results in execution times for fitness evaluation too long to be practical [17]. Unlike the previous work in [17], we include lookarounds for addressing problem instances with context, i.e., scenarios where a given sequence of characters has or has not to be extracted depending on its surroundings (Section 2.3.2). Lookaround is a shorthand for regular expression constructs which allow defining constraints on the text that either precedes or follows the snippet to be extracted, in the form of text that must or must not be present (see [89] for details). For instance, the regular expression r = (?<=\d\d-\d\d-)\d++ contains a lookaround operator, the *positive lookbehind* operator, that specifies which text must precede the snippet to be extracted. Given the string s = born:␣02-03-1979,␣graduated:␣21-07-04,␣age:␣35, the set of extractions $[\mathcal{X}_s]_r$ contains 1979_{12} and 04_{37}, but not 35_{44}. Some notable regular expression implementations (namely JavaScript) does not work with lookbehind.

Note that the mere addition of one or more regular expression operators does not necessarily broaden the scope of the system. The more the available constructs, the larger the search space: a system with too many operators to choose from may end up generating poor solutions. In fact, during our early redesign with the proposal in [17] we were unable to exploit lookaround operators effectively.

The set of $\mathcal{T}$-dependent labels contains *token* labels and *partial range* labels.

Token labels are generated as follows. A multiset T^d of candidate tokens is built by applying the regular expression \w+|\s+|[^\w\s]+ to each desired extraction in $\mathcal{T}$: that is, T^d contains all the extractions obtained by that regular expression on each element of $\bigcup_{(s,X_s^d,X_s^u)\in\mathcal{T}} X_s^d$. Then, the occurrency rate of each candidate token is computed as its multiplicity in T^d divided by $|\bigcup_{(s,X_s^d,X_s^u)\in\mathcal{T}} X_s^d|$. Finally, candidate tokens with an occurrency rate which is greater than 80% are retained as token labels. The same procedure is executed with respect to candidate tokens obtained from undesired extractions (only in tuples (s, X_s^d, X_s^u) for which $X_s^d \neq \varnothing$).

Partial range labels are obtained as the largest intervals of alphanumeric characters whose elements occur in the desired extractions (i.e., in $\bigcup_{(s,X_s^d,X_s^u)\in\mathcal{T}} X_s^d$). For instance, a-c and l-n are two partial ranges labels obtained from the strings cabin and male.

Fitness

The fitness definition, i.e., how to quantify the quality of a candidate solution for the problem being solved, is a fundamental design decision in GP. Several practical applications are based on a *multiobjective* approach, where the quality of a candidate solution is assessed by means of *two* fitness indexes: one for quantifying performance, the other for quantifying a complexity index of the solution, typically its size. Such an approach has proven to be very effective at preventing *bloat*, i.e., the proliferation of candidate solutions that grow bigger in size without any corresponding improvement in performance [71].

We developed a fitness definition in which the performance of the solution is taken into account by *two* performance indexes (differently from the single one used in [17]): one considers examples at the level of full extractions; the other considers instead each example as a character sequence where each character, specified by its value *and* position, is to be classified between extracted vs. non extracted. The aim of the latter is to rewards small improvements at the character level in the extraction behavior, even when they do not result in new full snippets correctly extracted. The same aim motivated the fitness definition of [17], but here we accomodate a scenario with multiple extractions for each example, which was not

[3] Greedy quantifiers: •* •+ •? •{•,•}. Lazy quantifiers: •*? •+? •?? •{•,•}?

tailored by the cited paper. We couple the two performance indexes with length of the regular expression, thereby resulting in three fitness indexes.

Our fitness definition requires comparing the actual extractions generated by a regular expression to the desired extractions. To this end, we define two operators over sets of snippets. Let X_s and X'_s be two sets of snippets of s. The *snippet set difference* $X_s \ominus X'_s$ is the set composed of each snippet in $\mathcal{X}_s$ which satisfies the following conditions: (i) is a subsnippet of, or is equal to, one or more snippets in X_s, (ii) does not overlap any snippet in X'_s, (iii) is not a subsnippet of any snippet which meets the two previous conditions. For instance, consider string $s = \mathsf{I\_said\_I\_wrote\_a\_ShortPaper}$ and the sets of snippets $X_s = \{\mathsf{I}_0, \mathsf{I}_7, \mathsf{ShortPaper}_{17}\}$, and $X'_s = \{\mathsf{I}_0, \mathsf{Pap}_{22}\}$. It will be $X_s \ominus X'_s = \{\mathsf{I}_7, \mathsf{Short}_{17}, \mathsf{er}_{25}\}$. The *snippet set intersection* $X_s \sqcap X'_s$ is defined in the same way except that condition ii requires to be a subsnippet of, or to be equal to, one or more snippets in X'_s. In the previous example it will be $X_s \sqcap X'_s = \{\mathsf{I}_0, \mathsf{Pap}_{22}\}$.

Each individual r is associated with a fitness tuple $f(r) := (\text{Prec}(r, \mathcal{T}), \text{Acc}(r, \mathcal{T}), \ell(r))$. The first component $\text{Prec}(r, \mathcal{T})$ of the fitness is the precision on the tuples in $\mathcal{T}$:

$$\text{Prec}(r, \mathcal{T}) := \frac{\sum_{(s, X_s^d, X_s^u) \in \mathcal{T}} \left| [\mathcal{X}_s]_r \sqcap X_s^d \right|}{\sum_{(s, X_s^d, X_s^u) \in \mathcal{T}} \left| [\mathcal{X}_s]_r \sqcap (X_s^d \cup X_s^u) \right|}$$

The second component $\text{Acc}(r, \mathcal{T})$ is the average of the True Positive Character Rate (TPCR) and True Negative Character Rate (TNCR):

$$\text{TPCR}(r, \mathcal{T}) := \frac{\sum_{(s, X_s^d, X_s^u) \in \mathcal{T}} \left\| [\mathcal{X}_s]_r \sqcap X_s^d \right\|}{\sum_{(s, X_s^d, X_s^u) \in \mathcal{T}} \left\| X_s^d \right\|}$$

$$\text{TNCR}(r, \mathcal{T}) := \frac{\sum_{(s, X_s^d, X_s^u) \in \mathcal{T}} \left\| (\{s_0\} \ominus [\mathcal{X}_s]_r) \sqcap X_s^u \right\|}{\sum_{(s, X_s^d, X_s^u) \in \mathcal{T}} \left\| X_s^u \right\|}$$

where $\|X\|$ is the sum of the length of all the snippets in X and s_0 is the snippet consisting of the whole string s.

Finally, the latter component $\ell(r)$ is the length of the regular expression r (this index has to be minimized, unlike the other two indexes which have to be maximized).

We rank individuals based on their fitness tuples as follows. An individual a *Pareto-dominates* another individual b if a is better than b on at least one fitness element and not worse on the other elements. An individual belongs to the ith *frontier* if and only if it is Pareto-dominated only by individuals belonging to jth frontier, with $j < i$ (individuals in the first frontier are not Pareto-dominated by any other individual). Based on these definitions, we first sort individuals based on the Pareto frontier they belong to. Second, we establish a total order among individuals belonging to the same Pareto frontier based on a lexicographic ordering among fitness indexes.

Initialization and evolution

Our GP search operates on a fixed-size population of n_{pop} individuals. We build the initial population basing on the training set $\mathcal{T}$, unlike the usual approach in GP which consists of building the entire population at random (as in [17]). We generate 4 individuals from each snippet in each example, all generated so as to extract that snippet. The rationale is to provide a sort of good starting point and useful genetic material for the search.

In detail, for each snippet x_s in $\bigcup_{(s, X_s^d, X_s^u) \in \mathcal{T}} X_s^d$, we generate 4 individuals as described below—Figure 2.1 shows an example of the procedure applied to a single snippet.

a) A tree is generated from the textual content of x_s using, whenever possible and with decreasing priority, (i) nodes with token label to represent the corresponding tokens, (ii) nodes with the label `\d` to represent digits, (iii) subtrees corresponding to `[a-zA-Z]` to represent alphabetic characters, (iv) nodes with predefined characters labels to represent corresponding characters, and (v) nodes with the label `.` for all other characters.

b) A tree is generated as in a, then subtrees composed only of nodes with two labels, one being the concatenator •• and the other a generic label l, are replaced by the subtree corresponding to l++.

c) Two snippets x_s^{behind} and x_s^{ahead} are considered such that their length is at most $10\ell(x_s)$ and they immediately precede (x_s^{behind}) or succeed (x_s^{ahead}) x_s in the corresponding tuple—they are not considered if x_s stays at the beginning or at the end of the string, respectively. Then, a tree for each snippet x_s^{behind}, x_s, and x_s^{ahead} is built as in a. Finally, a tree is built such that it corresponds to the concatenation of (i) a lookbehind node whose child is the tree obtained from x_s^{behind}, (ii) the tree obtained from x_s, and (iii) a lookahead node whose child is the tree obtained from x_s^{ahead}.

d) A tree is generated as in c and then modified as in b, by compacting subtrees of repeated leaf nodes. For lookbehind trees, subtrees are replaced by $l\{1,m\}$+, rather than l++, where l is the repeated label and m is the number of its occurrences in the subtree. This change is made to accommodate a limitation of common regular expression libraries which do not allow for ++ and *+ to occur within lookbehinds.

$s_1 =$ height:␣174.5cm $\quad s_2 =$ height:␣n.a.

$x_{s_1} = 174.5_8$

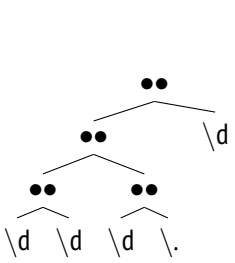
$r_a =$ \d\d\d\.\d

$r_b =$ \d++\.\d

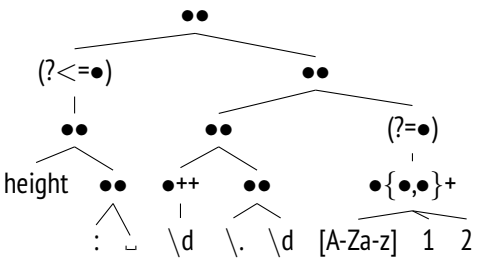
$r_c =$ (?<=height:␣)\d\d\d\.\d(?=[A-Za-z][A-Za-z])

$r_d =$ (?<=height:␣)\d++\.\d(?=[A-Za-z]{1,2}+)

(a) Individuals generated for x_s.



(b) Tree for r_a.



(c) Tree for r_d.

Figure 2.1: Example of the population initialization from a training set of 2 examples: 4 individuals r_a, r_b, r_c, r_d are generated from the only desired extraction x_{s_1}. The trees corresponding to 2 of them are shown: note that, in r_d, height is a token label (see Section 2.4.1). The subscript of the individuals (a–d) corresponds to the specific points described in Section 2.4.1.

If the number of individuals generated from the training set $\mathcal{T}$ is greater than n_{pop}, exceeding individuals are removed at random; otherwise, missing individuals are generated at random with a Ramped half-and-hald method [120], each one with a tree depth chosen randomly within the interval 2–15. Whenever an individual is generated whose string transformation is not a valid regular expression, it is discarded and a new one is generated.

Once the initial population is built, it is evolved iteratively as follows. At each iteration (called *generation*), n_{pop} new individuals are generated: 80% by *crossover* of pairs of individuals of the current population, 10% by *mutation* of individuals of the current population, 10% generated randomly with a Ramped half-and-half method. Crossover is a genetic operator which takes two individuals and outputs two new individuals that are identical to the input individuals except for two randomly selected subtrees that are swapped. Mutation is a genetic operator which takes an individual and outputs a new individual identical to the input inividual except for a randomly selected subtree that is replaced by a new randomly generated subtree. The choice of an individual (or a pair of individuals) to undergo mutation (or crossover) is made with a *tournament selection*: 7 individuals are randomly picked in the current population and the

one with the best fitness is selected. A new population is built from the resulting $2n_{\text{pop}}$ individuals, by retaining only the best n_{pop} of them.

The above procedure includes a *genotypic diversity enforcement* criterion, which was not present in [17], (a very similar mechanism is used in [128]): whenever an individual r_1 is generated whose string transformation is the same as one of an existing individual r_2 (i.e., one in the current population or one previously generated in the current iteration), r_1 is discarded and a new one is generated.

The iterative procedure is repeated until one of the two following conditions is met: (i) a number of n_{gen} iterations have been performed, or (ii) the fitness tuple of the best individual has remained unchanged for n_{stop} consecutive iterations. The string transformation of the best individual of the population at the end of the last iteration is the outcome of the GP search.

Separate-and-conquer

A problem instance may include desired extractions which are structurally very different from each other. For example, dates may be expressed in a myriad of different formats and learning a single pattern capable of expressing all these formats may be very difficult. While problem instance of this sort could be theoretically addressed by including the or operator $|$ among the building blocks available to the GP search for building candidate solutions, in practice such a design choice is ineffective. As it turned out from our analyses, that we omit for brevity, inclusion of the or operator generally leads to poor solutions, probably because of the much increased size of the solution space along with the difficulty of figuring out when such operator is actually needed and at which exact point of a candidate solution.

To address this important practical problem we designed a *separate-and-conquer* search procedure (which we previously sketched in [21]) that does not require the or operator yet is able to realize automatically whether multiple patterns are required and, in that case, to actually generate such patterns with an appropriate trade-off between specificity and generality.

A separate-and-conquer search consists of an iterative procedure in which, at each iteration, a GP search is performed and the snippets correctly extracted by the set of regular expressions generated so far are removed from the training set for the next iteration. This general scheme [91] is useful to cope with scenarios in which several problem sub-instances that are not explicitly delimited could be identified, such as in various forms of rule inference [13, 145, 163]. In detail, initially the target regular expression $\hat{r}$ is set to the empty string, then the following sequence of steps is repeated:

1. Perform a GP search on $\mathcal{T}$ and obtain r.
2. If $\text{Prec}(r, \mathcal{T}) = 1$, then assign $\hat{r} := \hat{r}|r$ (i.e., concatenate $\hat{r}$, the regular expression *or* operator $|$, and r), otherwise terminate.
3. For each $(s, X_s^d, X_s^u) \in \mathcal{T}$, assign $X_s^d := X_s^d \backslash [\mathcal{X}_s]_{\hat{r}}$;
4. If $\bigcup_{(s,X_s^d,X_s^u)\in\mathcal{T}} X_s^d$ is empty, then terminate.

In other words, at each iteration we require the currently generated regular expression r to have perfect precision (step 2): i.e., r must extract only snippets which are indeed to be extracted, but it might miss some other snippets. Since $\hat{r}$ is built up with the or operator, it extracts every snippet which is extracted by at least one of its components: it follows that $\hat{r}$ will have perfect precision and a recall greater than each of its components. The constraint on perfect precision of step 2 is indeed the reason for which we chose to favor the precision among individuals of the same Pareto frontier (see Section 2.4.1): the most prominent objective is exactly to maximize $\text{Prec}(p, \mathcal{T})$. Subsequent iterations will target the snippets still missed by $\hat{r}$ (step 3).

The GP search at step 1 is performed with $n_{\text{stop}} \ll n_{\text{gen}}$, so as to leave "difficult" examples for subsequent iterations of the separate-and-conquer procedure (by allowing an *early termination* of the search) and to avoid to over-focus on a training set when no significant improvements appear to be achievable.

2.4.2 High level organization of GP searches

Since a GP execution is a stochastic procedure, we follow a common approach in evolutionary algorithms which consists in executing multiple independent searches on the same training set and then selecting one of the solutions according to a predefined criterion.

In detail, we proceed as follows.

1. We partition the set E of examples available for learning in two sets E_t and E_v.
2. We build the training set $\mathcal{T}$ of the GP search based on E_t (see below) and keep E_v not available to the search.
3. We execute $2n_{job}$ independent GP searches, all with the same $\mathcal{T}$ but each with a different random seed. We call each such search a *job*. Execution of this step generates a pool of $2n_{\text{job}}$ solutions.
4. We compute the F-measure of each of the $2n_{\text{job}}$ solutions on the full set of learning examples $E = E_t \cup E_v$ and select the solution with best F-measure.

In other words, we use E_v as a *validation set* for assessing the generalization ability of a proposed solution on examples that were not available to the learning process —i.e., to prevent overfitting while promoting generalization.

The partitioning of E is made randomly so that the number of the snippets in E_t and E_v are roughly the same, i.e., $\sum_{(s,X_s)\in E_t} |X_s| \approx \sum_{(s,X_s)\in E_v} |X_s|$. The training set $\mathcal{T}$ for jobs is built simply: for each $(s, X_s) \in E_t$, a triplet $(s, X_s, \{s_0\} \ominus X_s)$ is inserted in $\mathcal{T}$ (i.e., $X_s^d := X_s$ and $X_s^u := \{s_0\} \ominus X_s$).

In order to broaden the spectrum of problem instances that can be addressed effectively, we do not execute all the $2n_{\text{job}}$ jobs in the same way. Instead, we execute n_{job} jobs according to separate-and-conquer, while each of the other n_{job} jobs consist of a *single* GP search where all the available generations (i.e., $n_{\text{stop}} = n_{\text{gen}}$) are devoted to learning a single pattern on the full training set $\mathcal{T}$.

2.5 Experimental evaluation

We carried out a thorough experimental evaluation for addressing the following questions: 1. How does our method perform on realistic problem instances, even w.r.t. manual authorship of regular expressions? 2. How do other relevant methods perform compared to ours? 3. What is the role of some of the key features of our proposal? We analyze each question in the following subsections.

We implemented the method here proposed as a Java application[4] in which jobs are executed in parallel. The implementation includes some significant optimizations aimed at speeding up executions: a full description can be found in section 2.6. We tuned the values for the parameters n_{job}, n_{pop}, n_{gen} and n_{stop} (the latter actually matters only in separate-and-conquer jobs) after exploratory experimentation and taking into account the abundant state of the art about GP. We set $n_{\text{job}} = 16$ (4 for the web version), $n_{\text{pop}} = 500$, $n_{\text{gen}} = 1000$ and $n_{\text{stop}} = 200$.

2.5.1 Extraction tasks and datasets

We considered 20 different extraction tasks defined by relevant combinations of 17 entity types to be extracted from 12 text corpora. We made available[5] part of the extraction tasks: we excluded those previously used in other works and those which cannot be included for privacy issues (e.g., those containing email addresses).

Table 2.2 (four leftmost columns) shows salient information about the 20 extraction tasks: number of examples $|E_0|$, their overall length (in thousands of characters) $\sum_{(s,X_s)\in E_0} \ell(s)$, and overall number of

[4] The source code is available on https://github.com/MaLeLabTs/RegexGenerator. A web-based version of the application is available on http://regex.inginf.units.it.

[5] http://machinelearning.inginf.units.it/data-and-tools/annotated-strings-for-learning-text-extractors

snippets $\sum_{(s,X_s)\in E_0} |X_s|$. The name of each extraction task is composed of the name of the corpus (see below) followed by the name of the entity type to be extracted. Entity names should be self-explanatory: Username corresponds to extracting only the username from Twitter citations (e.g., only MaleLabTs instead of @MaleLabTs); Email-ForTo corresponds to extracting email addresses appearing after the strings for: or to: (possibly capitalized). It seems fair to claim that these extraction tasks are quite challenging and representative of real world applications. Names ending with a $*$ suffix indicate extraction tasks with context (Section 2.3.2).

The text corpora are listed below. Some of them have been used in previous works about text extraction with the same (or similar) entity types to be extracted—all corpora but the last three ones have been used also in [17].

ReLIE-Web: portions of several web pages from the publicly available University of Michigan Web page collection. Used in [133].

ReLIE-Email: portions of the body of several emails from the publicly available Enron email collection. Used in [44, 133].

Cetinkaya-HTML: lines of the HTML source of 3 web pages. Used in [55].

Cetinkaya-Web: lines of plain text taken from 3 web pages after rendering. Used in [55].

Twitter: 50 000 Twitter messages collected using the Twitter Streaming API.

Log: 20 000 log entries collected from our lab gateway server running the vuurmuur firewall software.

Email-Headers: 101 headers obtained from several emails collected from personal mail boxes of our lab staff.

NoProfit-HTML: lines of the HTML source of the address book web page of a local nonprofit association.

Web-HTML: lines of the HTML source of several pages.

CongressBills: 600 US Congress bills from the THOMAS online database. In order to vary the format of the dates, we changed the format of the dates as to obtain 9 different formats—including 3 formats in which the month is expressed by name rather than by number.

BibTeX: 200 bibliographic references in the form of BibTeX elements obtained with Google Scholar.

Reference: 198 bibliographic references (the same of the BibTeX corpus with two removals) formatted according to the Springer LNCS format.

Table 2.1 shows the entity types along with the corresponding regular expressions. Entity names should be self-explanatory (left column): Username corresponds to extracting only the username from Twitter citations (e.g., only MaleLabTs instead of @MaleLabTs); Email-ForTo corresponds to extracting email addresses appearing after the strings for: or to: (possibly capitalized). It seems fair to claim that these extraction tasks are quite challenging and representative of real world applications.

2.5.2 Proposed method effectiveness

We evaluated our method as follows. For each extraction task we built several problem instances $(E, E^\star)$ differing in the overall number of snippets $\sum_{(s,X_s)\in E} |X_s|$ available for learning. In each problem instance we partitioned the set of examples E_0 in a learning set E and a testing set $E^\star = E_0 \backslash E$. We experimented with the values $24, 50, 100$ for the number of snippets in E. We applied our method 5 times for each of those values, by randomly varying the composition of E and hence $E^\star$, and averaged the obtained figures of precision and recall over the 5 repetitions. Hence, we analyzed 300 problem instances—5 repetitions for each of the 60 different combinations of extraction task and number of snippets for learning.

Table 2.1: Regular expressions used to extract the entities of interest. Long regular expressions are split in two lines of the same row.

Entity type	Regular expression used to build examples of E_0
All-URL	(?:^|\b|\W)((?:(https?|s?ftps?|smb|mailto)://)(?:[a-zA-Z0-9][^\s:]+@)?(?:(?:[a-zA-Z0-9\-\.:]+\.[a-zA-Z]{2,3}) |(?:[0-9]{1,3}(?:\.[0-9]{1,3}){3}))(?::\d{1,5})?(?:[A-Za-z0-9.\-$+!*'(),~_+%=&#/?@%;:]*)?(?<![.()=]+))
HTTP-URL	(?:^|\b|\W)((?:https?://)(?:[a-zA-Z0-9][^\s:]+@)?(?:(?:[a-zA-Z0-9\-\.:]+\.[a-zA-Z]{2,3})|(?:[0-9]{1,3}(?:\. [0-9]{1,3}){3}))(?::\d{1,5})?(?:[A-Za-z0-9.\-$+!*'(),~_+%=&#/?@%;:]*)?(?<![.()=]+))
Phone-Number	(?:\W|^)((?:[0-9](?:␣|-|\.))?(?:\(?[0-9]{3}\)?|[0-9]{3})(?:\\\|/|␣|-|\.)?(?:[0-9]{3}(?:␣|-|\.)[0-9]{4}))
href	(href=(?:["][^"]*["])|(?:['][^']*['))
href-Content	((?<=href=")(?:[^"]*)(?=")|(?<=href=')(?:[^']*)(?='))
Hashtag+Citation	(?:\W|^)((?:@[A-Za-z0-9_]+)|(?:#[\w]+(?<!#\d{0,120})\))
Username	(?:\W|^)@([A-Za-z0-9_]+)
IP	(?:\b|^)((?<!\.)(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]−[01]?[0-9][0-9]?) (?!\.))\b
MAC	((?:[0-9A-Fa-f]{2}[:-]){5}(?:[0-9A-Fa-f]{2})
Email-ForTo	(?<=(?:\s|-|^)(?:To|For|to|for):\s{0,200}(?:"?[^@"]{0,200}"?)?\s{0,200}[<"]?)[\w\.\-+]+@[\w\.\-]+ (?:=[>"])?
Email	(?:\b|^)((?:[a-zA-Z0-9_\-+][a-zA-Z0-9_\-\.+]*)@(?:(?:\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.)|(?:(?:[a-zA-Z 0-9\-]+\.)+))(?:[a-zA-Z]{2,4}|[0-9]{1,3}))(?:\b|$)
Heading	(<h\d.*?>.*?</h\d>)
Heading-Content	<h\d[^<>]*?>(.*?)</h\d>
Date	((?:[12]?\d{1,3}\s*[\-/]\s*[012]?\d\s*[\-/]\s*[12]?\d{1,3})|(?:\d{1,2},\s*[A-Z][a-z]+\s+[12]\d{3}) |(?:[A-Z][a-z]+\s+\d{1,2},\s+[12]\d{3}))
Title	(?:\b|^)title␣*=␣*\{\{?(.*)\}?\}
Author	(?<=author␣{0,3}=␣{0,3}\{[^\n]{0,300}?)([A-Zd.{\d\\'"()][A-Z␣.a-z{}\d\'"\-;()]*?)(?:(?:␣and␣)|(?:\}, \s*[a-z])|\n)
First-Author	(?<!:.{0,500})([\pL\w\-'"]*[\pL\w_\-'"␣]*,␣[A-Z.]+)

Table 2.2 summarizes our main results. The table has 60 rows, one for each combination of extraction task and number of snippets for learning. Sixth and seventh columns contain the number of snippets for learning and the *learning ratio (LR)* defined as the ratio between the number of snippets for learning and the number of snippets in the full extraction task E_0. The remaining columns on the left illustrate performance indexes of the learned regular expression $\hat{r}$: F-measure on the learning data E and, most importantly, precision, recall, and F-Measure on the testing data $E^\star$. The two last columns provide indexes for assessing the computational effort: EC is the overall number of characters which have been evaluated by candidate regular expressions—e.g., a population of 100 individuals applied to a set E including strings totaling 1000 characters for 100 generations corresponds to EC $= 10^7$. Figures in the table are expressed in multiples of 10^{10}. TtL is the time required for solving a problem instance: we used a machine powered with a 6 core Intel Xeon E5-2440 (2.40 GHz) equipped with 32 GB of RAM.

The key outcome of this experimental campaign is that F-measure on testing data is very high in nearly all scenarios analyzed. This result is particularly relevant in itself and becomes even more relevant in light of the very low LR values of our experimental setting, which indicate that our method is indeed able to find solutions that generalize effectively. It can also be seen that, in many extraction tasks, F-measure is very high also when the learning information includes only 24 snippets. This suggests that the proposed method can be very effective even with *few* examples.

The only extraction task in which F-measure is definitely unsatisfactory—in the range 35.8–48.3%—is ReLIE-Email/Phone-Number. This task was executed with LR in the range 0.5–1.9%. We executed this task again with LR $\approx 80\%$, a value much closer to the values usually used in machine learning literature and obtained a much higher F-measure on the testing data $E^\star$: $\approx 85\%$. We believe that this result demonstrates the quality of our approach even for this task. We carefully analyzed the results for ReLIE-Email/Phone-Number and we believe that this task is unlikely to be solved effectively with a very low LR. In particular, it can be seen from Table 2.2 that the generated regular expressions exhibit a rather high recall (92.6–98.3%) and a low precision (37.1–22.7%) on $E^\star$—i.e., they tend to extract all the relevant snippets but also unrelevant portions of the strings in $E^\star$. We manually inspected the learning data E and verified that they are not adequately representative of the data that are *not* to be extracted: they did not contain substrings which look like, but are not, phone numbers.

Concerning the impact of the number of snippets available for learning, results of Table 2.2 generally confirm that the more information available for learning, the better the obtained F-measure. There are indeed a few anomalies to this trend which, we believe, are due to the very low LR values and the highly challenging nature of the extraction tasks.

With respect to the *computational effort* (i.e., EC and TtL), our experimental evaluation shows that the time needed to learn a regular expression for a problem instance is often in the order of a few tens of minutes. We also found, as expected, that TtL depends approximately linearly from EC, which itself strongly depends on the aggregate "size" of the learning information in terms of characters, i.e., $\sum_{(s,X_s)\in E} \ell(s)$. While the absolute value of TtL would seem to discourage the on-the-fly usage of our method, our experience with its web-based implementation suggests that TtLs do not hamper the practicality of our tool. Moreover, we believe that TtL should be assessed from a *relative* point of view: a user highly skilled in regular expression writing probably would not even use our tool, while a user moderately skilled or unskilled at all may solve problems that would otherwise be unable to solve—see also Section 2.5.4 for a preliminary analysis and chapter 3 for an extensive study.

We found that tasks which may take advantage of modern regular expression constructs (lookarounds, possessive quantifiers) tend to require a longer execution time. We think this finding is motivated by the fact that our tool operates with a real-world regular expression engine (the one included in the Java platform): that engine cannot guarantee that the processing time of *every* regular expression grows linearly with the input string length, because the previously mentioned constructs cannot be implemented using automata; it follows that tasks in which the evolution tends to favor regular expressions with modern constructs, take much longer times to be solved. A list of the generated regular expressions is available in the following subsection.

Table 2.2: Results and salient information about the extraction tasks. The overall length $\sum_{(s,X_s)\in E_0} \ell(s)$ of examples is expressed in thousands of characters. EC is expressed in 10^{10} evaluated characters; TtL is expressed in minutes.

						On E		On $E^\star$			
Extraction task E_0	$\lvert E_0\rvert$	$\sum_{E_0} \ell(s)$	$\sum_{E_0} \lvert X_s\rvert$	$\sum_E \lvert X_s\rvert$	LR	Fm	Prec	Rec	Fm	EC	TtL
				24	5.0	99.2	90.0	91.9	90.9	2.6	15
ReLIE-Web/All-URL	3877	4240	502	50	10.0	99.2	92.1	95.0	93.5	6.4	35
				100	19.9	98.9	94.8	96.5	95.6	13.7	71
				24	5.0	99.2	86.3	89.0	87.6	2.5	11
ReLIE-Web/HTTP-URL	3877	4240	499	50	10.0	99.0	91.0	93.3	92.2	5.8	32
				100	20.0	98.8	92.9	96.8	94.8	13.1	66
				24	0.5	97.7	37.1	92.6	48.3	3.4	8
ReLIE-Email/Phone-Number	41 832	8805	5184	50	1.0	99.0	29.9	96.6	43.3	6.0	16
				100	1.9	98.9	22.7	98.3	35.8	14.4	39
				24	11.7	100.0	98.7	99.2	98.9	2.5	12
Cetinkaya-HTML/href	3425	154	214	50	23.4	100.0	98.1	98.7	98.4	4.9	26
				100	46.7	99.8	98.4	99.1	98.8	9.0	59
				24	11.7	98.4	74.9	98.7	80.6	2.4	16
Cetinkaya-HTML/href-Content*	3425	154	214	50	23.4	98.5	85.1	98.8	88.2	4.8	29
				100	46.7	98.5	83.2	96.8	86.2	10.5	67
				24	14.9	99.2	99.4	98.8	99.1	1.7	3
Cetinkaya-Web/All-URL	1234	39	168	50	29.8	100.0	95.5	98.6	96.9	3.2	8
				100	59.5	99.5	98.8	98.8	98.8	5.2	16
				24	0.1	100.0	98.8	100.0	99.4	1.2	3
Twitter/Hashtag+Citation	50 000	4344	56 994	50	0.1	99.6	99.2	100.0	99.6	2.2	4
				100	0.2	99.8	99.0	100.0	99.5	4.6	7
				24	0.2	100.0	94.7	98.5	96.6	1.8	3
Twitter/All-URL	50 000	4344	14 628	50	0.3	100.0	96.2	98.3	97.2	3.4	8
				100	0.7	99.4	96.1	98.0	97.0	7.7	16
				24	0.1	100.0	99.3	100.0	99.7	1.2	2
Twitter/Username*	50 000	4344	42 352	50	0.1	100.0	99.2	100.0	99.6	2.2	2
				100	0.2	99.9	99.3	100.0	99.7	4.6	2
				24	0.1	100.0	99.8	100.0	99.9	1.3	2
Log/IP	20 000	4126	75 958	50	0.1	100.0	99.7	100.0	99.8	2.3	2
				100	0.2	100.0	99.8	100.0	99.9	4.6	3
				24	0.1	100.0	100.0	100.0	100.0	2.0	2
Log/MAC	20 000	4126	38 812	50	0.1	100.0	100.0	99.4	99.7	4.3	3
				100	0.3	100.0	100.0	99.4	99.7	8.3	6
				24	2.9	97.5	86.7	87.9	86.9	5.9	18
Email-Headers/IP	101	261	848	50	5.9	92.7	90.9	82.2	86.0	14.0	56
				100	11.8	94.5	95.2	84.9	89.6	28.5	89
				24	7.6	78.5	70.7	52.5	59.3	17.9	131
Email-Headers/Email-ForTo*	101	261	331	50	15.1	71.5	76.4	52.8	62.0	33.7	398
				100	30.2	79.8	90.4	66.6	76.4	65.5	429
				24	2.3	100.0	83.2	100.0	85.5	0.9	2
NoProfit-HTML/Email	25 590	860	1094	50	4.6	100.0	100.0	100.0	100.0	1.9	3
				100	9.1	100.0	100.0	100.0	100.0	3.7	7
				24	2.3	99.2	93.1	89.4	91.2	7.6	30
Web-HTML/Heading	49 026	4541	1083	50	4.6	96.2	93.3	90.2	91.7	15.3	83
				100	9.2	99.2	98.2	96.2	97.2	29.7	256
				24	2.3	93.6	95.5	80.1	86.6	6.6	76
Web-HTML/Heading-Content*	49 026	4541	1083	50	4.6	95.9	99.1	85.8	91.8	13.6	168
				100	9.2	98.9	99.4	96.1	97.7	28.0	379
				24	0.8	64.5	57.1	52.3	50.0	2.1	30
CongressBill/Date	600	16 511	3085	50	1.6	72.1	55.4	81.3	64.1	6.9	584
				100	3.2	76.1	62.7	81.4	69.7	11.3	513
				24	12.5	89.6	79.1	65.1	70.7	5.1	43
BibTeX/Title	200	54	200	50	25.0	90.3	82.6	74.3	78.0	11.1	141
				100	50.0	82.0	84.8	63.4	72.1	21.5	218
				24	4.2	92.9	90.5	78.1	83.1	2.0	8
BibTeX/Author	200	54	589	50	8.5	93.9	89.9	86.1	87.7	4.1	20
				100	17.0	90.7	91.9	81.6	86.2	7.5	34
				24	12.6	99.0	99.7	96.0	97.8	2.8	12
References/First-Author*	198	30	198	50	25.3	96.3	99.6	93.6	96.5	5.4	26
				100	50.5	100.0	100.0	100.0	100.0	12.4	56

2.5.3 Learned regular expressions

Table 2.3 shows samples of the learned regular expressions (for configurations with $\sum_{(s,X_s)\in E} |X_s| = 100$). Since we executed 5 repetitions for each configuration, the table shows the expression $\hat{r}$ that was generated more times or, in case of tie, the one learned first. A domain-expert user would perhaps appreciate that, in several cases, $\hat{r}$ looks as if it had been generated by a human operator: e.g., for Twitter/Username*, BibTeX/Title*, or ReLIE-Web/HTTP-URL. This is an interesting result, because we did not inclue in our tool any mechanism aimed at favoring *readability*, other than simply penalizing long regular expressions (which was motivated by the need of avoiding bloating). It can also be seen that, for some extraction tasks, the generated regular expressions contain the or operator |: this occurred, in particular, in those problem instances in which the snippets to be extracted indeed exhibited different formats, e.g., CongressBill/date, ReLIE-Email/Phone-Number.

Table 2.3: Learned regular expressions with $\sum_E |X_s| = 100$. Long regular expressions are split in two or more lines of the same cell.

Extraction task	$\hat{r}$
ReLIE-Web/All-URL	http(?:\w*+\.*+[^\\>␣])*+\w++|http://([^␣]\w++/?+)++
ReLIE-Web/HTTP-URL	http([^␣][\w/]++)++
ReLIE-Email/Phone-Number	\(\w++\)␣\w++\-\w++|\w\w\w\-\w++\-\w++|\d++[^␣][^8]\d++[^␣]\d++
Cetinkaya-HTML/href	href="[^"]*+"|href[^>␣]++
Cetinkaya-HTML/href-Content*	(?<=href=")[^"]++|(?<=='\)[\w\./]++[<']++
Cetinkaya-Web/All-URL	ftp:[^␣]*+
Twitter/Hashtag+Citation	[@#]\w++
Twitter/All-URL	\w++://\w\.co/\w++
Twitter/Username*	(?<=@)\w++
Log/IP	(\w++\.)++\w++
Log/MAC	\w++:(\w\w:)++\w++
Email-Headers/IP	(?<!@)(?<!\.)(?<!\w)(?:2012)*+6?+\d++\.\w*+\.\d++\.\d++
Email-Headers/Email-ForTo*	(?<=To:␣)[\w\.\-]++@[\w\.]++|(?<=\-\w\w\w\w\w\-To:␣<) [\+\w\-]++@[^>]++
NoProfit-HTML/Email	[^@]++[^<]++
Web-HTML/Heading	<\w\d(?:[^;][^>][^\d]*+)*+\w>
Web-HTML/Heading-Content*	(?<=>)(<*+[^<]++(?=<))++(?=</\w\d)
CongressBill/Date	\w++␣\w{1,2}+,␣[^>]*+\d++|\w++/\w\w/\w++| \w++\.\w\w\.\w++|\w++\-\w\w\-\w++| \d++,␣\w\w\w\w\w\w++␣\w\d++
BibTeX/Title*	(?<=title=\{)[^\}]++
BibTeX/Author*	(?<=␣)\w++,␣[\^␣]++(?:␣\w++(?=␣and))?+(?=␣and)| \w++,␣\w\w++(?<![year])(?:␣\w++(?!(?!␣and)␣))*+(?!␣and␣\w++␣)
Reference/First-Author*	(?<=\.␣)[^,]*+,␣(?:\w\.)++

2.5.4 Comparison with human operators

In order to assess the ability of our method to compete with human operators, we executed an experiment using a web application which we crafted ad hoc.

The web app presented concise instructions about the experiment ("write a regular expression for extracting text portions which follow a pattern specified by examples") and then asked the user to indicate the level of familiarity with regular expressions—one among novice, intermediate, and experienced. The web app then proposed a sequence of extraction tasks: for each task the web app showed a text on which the snippets to be extracted were highlighted; the user could write and modify a regular expression in an input field at will; the web app immediately highlighted the snippets actually extracted by the current

Table 2.4: F-measure for $\sum_E |X_s| = 24$ obtained by human operators (novice (SN), intermediate (SI), and experienced (SE)) and our approach (O).

	F-measure on E				F-measure on $E^\star$				Time spent/TtL			
Extraction task	SN	SI	SE	O	SN	SI	SE	O	SN	SI	SE	O
ReLIE-Web/All-URL	83.6	96.2	87.2	100.0	74.7	90.2	80.6	95.5	3	3	2	15
ReLIE-Web/HTTP-URL	86.4	89.3	91.3	95.8	77.3	83.0	76.6	82.3	20	8	9	11
ReLIE-Email/Phone-Number	88.5	96.3	100.0	100.0	70.2	84.7	91.0	34.6	8	6	6	11
Cetinkaya-HTML/href	91.9	99.6	99.8	100.0	91.6	98.8	98.8	100.0	4	2	3	12
Cetinkaya-Web/All-URL	96.9	100.0	100.0	100.0	95.2	98.3	98.6	99.0	80	2	1	3
Log/IP	91.3	100.0	100.0	100.0	91.0	100.0	100.0	100.0	5	1	2	2
Log/MAC	87.6	91.7	100.0	100.0	87.6	91.7	100.0	100.0	6	6	4	2
Web-HTML/Heading	87.6	99.5	99.6	100.0	82.3	90.9	95.6	90.0	7	5	2	30
BibTeX/Author*	71.3	55.2	88.7	100.0	64.6	50.1	81.4	90.3	20	10	10	8

expression along with the corresponding extraction mistakes (if any). The web app also showed the F-measure and the user was informed that a value of 100% meant a perfect score on the task. The user was not required to obtain a perfect F-measure before going to the next task—i.e., he could give up on a task. In the limit, he could also not write any regular expression for a task (*unanswered task*). The web app recorded, for each task and for each user, the authored regular expression and the overall time spent.

We included in the web app 9 of the extraction tasks presented in the Section 2.5.1. For each task, we chose exactly the E set we used while experimenting with our method for repetition 1 and $\sum_E |X_s| = 24$. We spread a link to the web app among CS graduate and undergraduate students of our University. Each user interacted with the web app autonomously and in an unconstrained enviroment—in particular, users were allowed to (and not explicitly instructed not to) refer to any knowledge base concerning regular expressions.

We gathered results from 73 users—60% novice, 20% intermediate, and 20% experienced. Several tasks were left unanswered: 42% for novice, 40% for intermediate, and 12% for experienced. The average time for solving the answered tasks was 16.1 min, 4.8 min, and 4.7 min, respectively. As a comparison, our method on the very same data required TtL $= 10.4$ min on the average.

The key finding is in Table 2.4, which shows the F-measure on $E^\star$ for each task. It can be seen that the F-measure obtained by our method is almost always greater than or equal to the one obtained by human users (on the average). The only exceptions are: the ReLIE-Email/Phone-Number task (whose peculiarity has been analyzed in Section 2.5.2); the Web-HTML/Heading task, in which our method improves over novice users and is only sligthly worse than intermediate users. We believe this result is remarkable and highly encouraging. Indeed, we are not aware of any proposal for automatic generation of regular expressions in which human operators were used as a baseline.

2.5.5 Comparison with other methods

The previous section considered a baseline in terms of human operators. In this section we consider a baseline in terms of other approaches for learning text extractors from examples: *Smart State Labeling DFA Learning (SSL-DFA)* [140], *FlashExtract* [129], and *GP-Regex* [17]. These methods are representative of the state of the art for learning syntactical patterns (see also Section 2.2), but differ in the actual nature of the learned artifact: SSL-DFA produces Deterministic Finite Automata (DFA), FlashExtract produces extraction programs expressed in a specific language, and GP-Regex produces regular expressions.

SSL-DFA

We chose to consider SSL-DFA because the problem of DFA learning from examples is long established and several solutions have been proposed. In particular, SSL-DFA was developed a few years after a competition that was highly influential in the grammar learning community and outperformed (optimized versions of) the winners of the competition, on the same class of problems [60, 126] and even in the

presence of noisy data. Another reason for our choice is because, in our experience at conferences and received reviews, we were often told that the problem considered in this work is not interesting because DFA learning is a solved problem.

Three notable differences exist between the DFA learning scenario usually considered in literature and the text extraction scenario here considered. First, DFA learning methods are tailored to short strings defined over a binary alphabet, whereas in text extraction one needs to cope with longer strings defined over a much larger alphabet (in general, UTF-8). Second, DFA learning methods assume that the examples are drawn uniformly from the input space, whereas in text extraction this assumption is hard to be verified in practice. Third, learned DFA are intended as classifiers, i.e., given an input string, their outcome states if the whole string is accepted or rejected by the DFA: some adaptation is hence needed in order to use a DFA for extraction. DFA learning and text extraction are thus quite different problems. An approach designed for the former problem may or may not perform well for the latter. Thus, it would not be surprising if approaches explicitly designed for text extraction outperformed SSL-DFA—our experimental evaluation suggests that this is indeed the case.

The SSL-DFA method described in [140] takes as input a triplet (S_A, S_R, n), where S_A is a set of strings which should be accepted, S_R is a set of strings which should be rejected, and n is the number of states of the DFA to be learned; the output is a DFA with n states. Given a problem instance $(E, E^\star)$, we obtain (S_A, S_R, n) as follows. Initially, we set $S_A = S_R = \varnothing$; then, for each example (s, X_s) in E, we add to S_A the textual content of each snippet in X_s and we add to S_R the textual content of each snippet in $\mathcal{X}_s \ominus X_s$; finally, we set $n = \max_{s \in S_A} \ell(s)$. With respect to the experimental setting considered in [140], the learning information available to SSL-DFA in our experiments is not balanced (i.e., in general, $|S_R| \gg |S_A|$): we verified experimentally that attempting to balance it by sampling the strings in S_R did not lead to better performance.

Concerning the actual extraction, we define the set eds of the extractions obtained by applying a DFA d to a string s as the set of all the non-overlapping snippets of maximal length which are accepted by d. We implemented this method in C++ basing on the description in [140].

FlashExtract

FlashExtract is s a powerful and sophisticated framework for extracting multiple different fields automatically in semi-structured documents [129]. It consists of an inductive synthesis algorithm for synthesizing data extraction programs from few examples, in which programs are expressed in any underlying domain-specific language supporting a predefined algebra of few core operators. The cited work presents also a language designed to operate on text which perfectly fits the extraction problem considered in this chapter. The findings of [129] resulted in a tool included in the Windows Powershell as the ConvertFrom-String command: we used this tool to perform the experiments.

The current FlashExtract implementation does not allow reusing a program induced by a given set of examples. Thus, in our experimentation the two phases of learning and testing were not separated: we invoked the tool by specifying as input the examples in E and the strings in $E^\star$; we obtained as output a set of substrings extracted from $E^\star$ based on the description in E (which we had to recast in the syntax required by the tool). In many cases the tool crashed, thereby preventing the extraction to actually complete. We highlighted these cases in the results.

GP-Regex

GP-Regex is the method we proposed in [17] and the base for the research here presented. The numerous differences between our method and GP-Regex were listed in the introduction. We emphasize again that in GP-Regex each example consists of a string and at most one single snippet to be extracted from that string. In order to build learning examples suitable for GP-Regex, we considered for each (s, X_s), only the leftmost snippet in X_s, if any.

It may be useful to remark that in [17] an experimental comparison was made against the approaches of [44, 133] on two datasets previously used by the latter: the proposal in [17] exhibited better accuracy,

even when the amount of available learning examples was smaller by more than order of magnitude. Moreover, the authors of [44, 133] showed that their approaches exhibited performance similar to *Conditional Random Fields*.

Comparison results

We selected 7 extraction tasks including tasks with context and tasks in which snippets exhibit widely differing formats. We exercised all methods with the same experimental settings described in section 2.5.2, thereby obtaining 105 problem instances—5 repetitions for each of the 21 different combinations of extraction task and number of snippets for learning.

Table 2.5 shows the results in terms of F-measure and TtL. The foremost outcome of this comparison is that our method clearly outperforms all the other three methods (except for ReLIE-Email/Phone-Number, discussed below).

Table 2.5: Results. TtL is expressed in minutes. Missing values for FlashExtract, denoted with $-$, correspond to problem instances for which no repetition completed successfully (see text).

		SSL-DFA		FlashExtract		GP-Regex		Our proposal	
Extraction task	$\sum_E \lvert X_s \rvert$	Fm	TtL	Fm	TtL	Fm	TtL	Fm	TtL
ReLIE-Web/All-URL	24	13.8	1	15.2	1	78.3	5	90.9	15
ReLIE-Web/All-URL	50	18.2	1	25.2	3	88.0	10	93.5	35
ReLIE-Web/All-URL	100	18.1	2	21.5	3	93.0	16	95.6	71
ReLIE-Email/Phone-Number	24	27.7	1	69.5	1	84.0	5	48.3	8
ReLIE-Email/Phone-Number	50	11.3	2	–	–	91.7	13	43.3	16
ReLIE-Email/Phone-Number	100	15.9	2	–	–	90.2	18	35.8	39
Cetinkaya-HTML/href	24	21.2	1	23.5	1	46.9	13	98.9	12
Cetinkaya-HTML/href	50	12.0	1	28.7	1	81.6	26	98.4	26
Cetinkaya-HTML/href	100	9.2	2	32.3	3	89.6	44	98.8	59
Cetinkaya-Web/All-URL	24	18.5	1	33.9	71	83.4	8	99.1	3
Cetinkaya-Web/All-URL	50	14.7	1	52.2	52	92.7	18	96.9	8
Cetinkaya-Web/All-URL	100	17.6	1	61.8	25	94.9	31	98.8	16
Twitter/Hashtag+Citation	24	14.5	1	–	–	94.8	3	99.6	3
Twitter/Hashtag+Citation	50	21.8	1	–	–	97.3	5	99.5	4
Twitter/Hashtag+Citation	100	28.4	1	–	–	100.0	8	99.6	7
Web-HTML/Heading-Content*	24	11.9	1	10.6	4	4.4	34	86.6	76
Web-HTML/Heading-Content*	50	11.9	2	10.2	3	5.0	196	91.8	168
Web-HTML/Heading-Content*	100	18.2	3	–	–	10.2	672	97.7	379
CongressBill/Date	24	37.4	1	–	–	29.8	361	64.1	30
CongressBill/Date	50	25.2	3	–	–	27.8	432	69.7	584
CongressBill/Date	100	46.5	4	–	–	38.0	386	70.7	513

The performance gap with both SSL-DFA and FlashExtract is substantial—at the expense of a much longer TtL, though. Concerning SSL-DFA, we believe this finding is interesting because both SSL-DFA and our method are based, broadly speaking, on evolutionary computation and SSL-DFA is representative of the state-of-the-art in its field. On the other hand, based on the previous considerations about the significant differences between typical settings for DFA learning and text extraction, we do not find this result particularly surprising either. Indeed, as pointed out earlier by different authors, benchmark problems for DFA learning from examples are not inspired by any real world application [60] and the applicability of the corresponding learning algorithms to other application domains is still largely unexplored [40].

Concerning FlashExtract, the fact that we obtain much better accuracy in all settings (with the only exception of ReLIE-Email/Phone-Number, discussed below) is also very interesting. We are not able to provide any principled interpretation for this result. We may only speculate that our approach is perhaps

Table 2.6: F-measure for $\sum_E |X_s| = 100$ with our fitness (O) and with the F-measure based fitness (F).

Extraction task	O	F	ΔFm
ReLIE-Web/All-URL	95.6	11.7	83.9
ReLIE-Web/HTTP-URL	94.8	14.8	80.0
Cetinkaya-HTML/href	98.8	98.6	0.2
BibTeX/Title*	72.1	3.3	68.8
BibTeX/Author*	86.2	24.9	61.3
References/First-Author*	100.0	NaN	NaN

more suitable for coping with loosely structured or unstructured datasets than FlashExtract. We also noticed that, for many problem instances, the ConvertFrom-String tool crashed, thereby preventing the extraction to actually complete. For the extraction tasks for which at least one on 5 repetition completed without errors, Table 2.5 shows the performances (F-measure and TtL) averaged across the completed executions. In the other cases, we were not able to obtain any extraction program, neither splitting the testing set in small chunks: those cases are denoted with an en dash in the table.

Concerning GP-Regex, we should isolate two groups of extraction tasks: (i) those that requires either a context (Web-HTML/Heading-Content*) or the ability to learn widely differing patterns (CongressBill/Date), (ii) all the other tasks. The key observation is that our current proposal improves over GP-Regex in all cases (except for ReLIE-Email/Phone-Number), the improvement being substantial in case i. Indeed, our current proposal makes it possible to handle both Web-HTML/Heading-Content* and CongressBill/Date with good accuracy, while GP-Regex does not. It is also interesting to observe that in case ii GP-Regex provides much better accuracy than SSL-DFA and FlashExtract, while in case i GP-Regex is either comparable to those methods or worse.

Finally, concerning the ReLIE-Email/Phone-Number extraction task, we observe that this is the same task with a sort of anomalous behavior already discussed in the previous section. In particular, we remark that when executing our method on this task with a learning ratio[6] LR $\approx 80\%$ we obtained $\approx 85\%$ F-measure on the testing data. We could not execute FlashExtract in those conditions because it always crashed: the only result that we could obtain is in Table 2.5, where F-measure (with very few examples available for learning) is 69%. The reason why GP-Regex happens to deliver better accuracy on this task is because it tends to overfit the snippets to be extracted more than the method here presented. As discussed in the previous section, processing this task with a very small LR value incurs in a poor representativeness of the text that is not to be extracted; as it turns out, thus, the slightly overfitting behavior exhibited by GP-Regex in this case turns out to be a pro.

2.5.6 Assessment of specific contributions

In order to gain further insights into our proposal, we executed a further suite of experiments on a subset of the extraction tasks aimed at assessing the effect of: (i) choice of the fitness, (ii) initialization of the population from E, and (iii) separate-and-conquer jobs.

Fitness

We built a variant of our method in which the fitness tuple of a solution consists in the F-measure on the examples in the training set and the length of the corresponding regular expression: $f(r) := (\text{Fm}(r, \mathcal{T}), \ell(r))$. In other words, we replace snippet-level precision and character-level accuracy (see Section 2.4.1 for the exact definition) by snippet-level F-measure, i.e., by the main performance index desired by the solution.

Table 2.6 presents the results. The rightmost column shows the improvement ΔFm obtained by our proposal w.r.t. the method with the fitness modified as above. It can be seen that the modified method leads

[6] *Learning ratio (LR)* is defined as the ratio between the number of snippets for learning and the number of snippets in the full extraction task E_0.

Table 2.7: F-measure with $\sum_E |X_s| = 100$ with and without initialization.

Extraction task	w/	w/o	ΔFm
ReLIE-Web/All-URL	95.6	73.6	22.0
ReLIE-Web/HTTP-URL	94.8	82.6	12.2
Cetinkaya-HTML/href	98.8	48.8	50.0
BibTeX/Title*	72.1	65.1	7.0
BibTeX/Author*	86.2	67.2	19.0
References/First-Author*	100.0	78.7	21.3

to a much worse F-measure, despite F-measure being exactly the index optimized by that method: our proposal leads to an improvement, on the average, around $\Delta\text{Fm} \approx 60\%$ ($\sum_E |X_s| = 100 \in \{24, 50, 100\}$). In other words, driving the evolutionary search by the key index of interest is not the optimal fitness choice. This finding corroborates some arguments made in [17] and augments them with an experimental evaluation.

It is worth to note that for the References/First-Author* task, the modified method is simply unable to produce a solution which can correctly extract at least one snippet. Our explanation is that the solving regular expression for that task is rather complex, since it includes multiple lookaround operators: light modifications to a regular expression which includes operators of this kind may result in very different extraction behaviors. In such a case, a fitness based on full snippets rather than individual characters does not acknowledge for small improvements and is not hence able to drive the evolution—in other words, it imposes an excessive evolutionary pressure.

Initialization

We built a variant of our method in which the initial population is totally built at random, instead of being partially generated using the snippets of the training set.

Table 2.7 shows the comparison results, which clearly indicate that the unmodified version is much more effective ($\Delta\text{Fm} \approx 25\%$, on the average for $\sum_E |X_s| \in \{24, 50, 100\}$). The rationale of the population initialization from the examples was to start the evolutionary search from a "good" point in the solution space. For this reason we inserted in the initial population individuals which fitted the snippets to be extracted while at the same time generalizing beyond them, e.g., we insert the regular expression `\d++-\d++-\d++` from the snippet `07-02-2011` (see Section 2.4.1).

Separate-and-conquer

We built a variant of our method in which all the $2n_{\text{job}}$ jobs are executed without the separate-and-conquer strategy, i.e., all jobs consist of a single GP search for which $n_{\text{stop}} = n_{\text{gen}}$.

Table 2.8 shows the comparison results. For this comparison, we considered also an extraction task (CongressBill/Date) in which the snippets to be extracted exhibit widely differing formats. As expected, the unmodified method clearly outperforms the modified one on CongressBill/Date ($\Delta\text{Fm} \approx 30\%$ for $\sum_E |X_s| \in \{24, 50, 100\}$). On the other hand, it can be seen that some not negligible improvement can be obtained also for other tasks, namely BibTeX/Title* and BibTeX/Author*. We think that the motivation is in that those tasks are more difficult and hence the possibility, enabled by the separate-and-conquer strategy, to split a problem in smaller subproblems may allow the method to better cope with such difficulty.

Table 2.8: F-measure with $\sum_E |X_s| = 100$ with and without separate-and-conquer.

Extraction task	w/	w/o	ΔFm
ReLIE-Web/All-URL	95.6	89.5	6.1
ReLIE-Web/HTTP-URL	94.8	87.6	7.2
Cetinkaya-HTML/href	98.8	95.3	3.5
CongressBill/Date	69.7	32.7	37.0
BibTeX/Title*	72.1	62.0	10.1
BibTeX/Author*	86.2	71.5	14.7
References/First-Author*	100.0	97.3	2.7

2.6 Implementation

We implemented the method here proposed as a Java application[7] in which jobs are executed in parallel. We tuned the values for the parameters n_{job}, n_{pop}, n_{gen} and n_{stop} (the latter actually matters only in separate-and-conquer jobs) after exploratory experimentation and taking into account the abundant state of the art about GP. We set $n_{\text{job}} = 16$, $n_{\text{pop}} = 500$, $n_{\text{gen}} = 1000$ and $n_{\text{stop}} = 200$. For the web-based version, we set $n_{\text{job}} = 4$ to save computing resources and support an higher number of concurrent usages.

Our application includes two significant optimizations aimed at speeding up executions. We implemented a *caching* mechanism for reducing repeated evaluations of the same regular expression. This mechanism consists of a Java WeakHashMap[8], in which the key is the string transformation of an individual and the value is the set of extractions of that individual in the training data. When a given individual continues to exist across many generations, thus, the corresponding regular expression will be applied to all training data only once, rather than at each generation. Furthermore, this data structure is shared among all jobs in a search. It follows that the cached extractions may be exploited even for identical individuals that come into existence in different jobs We found experimentally that this caching mechanism allows to save roughly 50% of the computation time, on the average.

The second optimization comes into play in those problem instances in which the overall length of the examples is very large. This optimization consists in a procedure, that we call *shrinking*, aimed at reducing the overall length of the examples while not affecting the salient information available for learning the regular expression. The procedure transforms the set of examples E in another set of examples E_{shrunk} in such a way that long strings in E are transformed into much smaller strings in E_{shrunk} while preserving the content around each snippet (and dropping examples with $X_s = \varnothing$, i.e., without any snippet to be extracted). More in detail, for each example $(s, X_s) \in E$, a subset $E' \subset E_{\text{shrunk}}$ exists such that (i) each snippet in X_s occurs exactly once in E', (ii) each string s of an example in E' contains at most $10\ell(x_s)$ characters before and $10\ell(x_s)$ characters after each snippet x_s. Of course, the shrinking procedure leads to a loss of information in the training data. However, we found that this heuristic works well in practice, in particular, because large training data are usually highly unbalanced, with relatively few snippets to be extracted surrounded by very many characters that are not to be extracted. Indeed, the shrinking procedure makes it possible to handle datasets that could hardly be processed otherwise. We chose to trigger the procedure when $\sum_{(s,X_s)\in E} \ell(s) \geqslant 10^7$ characters, as training data of such size lead to a processing time that is too long to be practical.

2.7 Remarks

We have described a system for synthesizing a regular expression automatically, based solely on examples of the desired behavior. The regular expression is meant to be used for extraction problems of practical complexity, from text streams that are either loosely structured or fully unstructured. As such, our

[7] The source code is available on https://github.com/MaLeLabTs/RegexGenerator. A web-based version of the application is available on http://regex.inginf.units.it.

[8] https://docs.oracle.com/javase/7/docs/api/java/util/WeakHashMap.html

approach is able to handle potentially large alphabets effectively, thereby overcoming one of the principal limitations of much existing work in this area, and has been designed to address such practical needs as context-dependent extractions, widely different formats, and potentially large and unsegmented input streams.

We have analyzed our proposal experimentally in depth, by applying it on 20 challenging extraction tasks of realistic size and complexity, with a very small portion of the dataset available for learning. The results have been very good and compared very favorably with significant baseline methods. Most importantly, the results are highly competitive also with respect to a pool of more than 70 human operators.

We made publicly available the source code of our system (https://github.com/MaLeLabTs/RegexGenerator) and deployed an implementation as a web app (http://regex.inginf.units.it).

While our work may certainly be improved and enriched in several ways—faster learning, interactive learning procedures capable of starting with a very small number of snippets, even better accuracy, just to name a few—we do believe that our work may constitute a useful solution to a practically relevant and highly challenging problem.

Chapter 3

Can A Machine Replace Humans In Building Regular Expressions?

3.1 Overview

Regular expressions are routinely used in a variety of different application domains and are widely viewed as one of the fundamental tools that should be in a programmer's toolbox. Building a regular expression tailored to a specific problem is often difficult, tricky and time-consuming, though. In March 2016 web site Stack Overflow, the most popular Question & Answer programming forum, features more than $140,000$ questions on this topic with "regex" being the 25-th most popular question tag in a set including more than $44,000$ tags. Nearly all of the question tags which are more popular than "regex" refer to a specific programming language or library—"arrays" is the only general tag more popular than "regex", while "ajax" and "json" are only slightly more popular than "regex".

There is no doubt that writing a regular expression requires a considerable amount of skill, expertise and creativity by the programmer. In this chapter we investigate whether a machine may surrogate these qualities and construct automatically regular expressions for tasks of realistic complexity. We address this question based on a large scale experiment involving more than 1700 users on 10 challenging tasks. We asked users to construct a regular expression based on a few examples of the desired behavior and then compared their solutions to those obtained with an automatic tool that we recently developed and described in full detail in earlier works [46, 77]. Our tool is based on Genetic Programming. Both the users and the tool were given the very same information: examples of the desired behavior without any hint about the structure of the target expression. We compared the results along two axes: quality of the solution assessed on a hold-out testing set and the time required for constructing the solution.

The quality of automatically-constructed solutions was very similar to the quality of solutions constructed by (self-proclaimed) experienced users; and, the time that our tool took to construct a solution was similar to that of humans performing the same task. The machine was thus able to indeed surrogate expertise and creativity of programmers in a traditionally difficult synthesis activity (see also the sidebar).

3.2 Problem Statement

Regular expressions are often used for binary classifying strings, depending on whether a string matches or does not match the pattern encoded by the expression. We consider instead extraction problems in which it is also required to identify all the substrings matching the specified pattern. Extraction is more general than classification in the sense that a solution to the former is also a solution to the latter, while the opposite is not true—a string could include many instances of the specified pattern; the knowledge that at least one instance of the pattern occurs somewhere in the string may not help very much in actually locating all those instances.

To specify the problem we need a few definitions. A regular expression applied on a string s deterministically extracts zero or more substrings from s, that we call extractions. The problem input consists of a set of examples, where an example is a string s coupled with a (possibly empty) set X_s of non-overlapping substrings of s. Set X_s represents the desired extractions from s, i.e., all the substrings in X_s are to be extracted whereas any other substring of s is not to be extracted. We do not make any assumptions on either the length or the internal structure of string s, which may be a text line, or an email message, or a log file, and so on. In practice, substrings in X_s may be specified easily by annotating portions of s with a GUI (see next section).

The problem consists of learning a regular expression whose extraction behavior is consistent with the provided examples: for each example, should extract from each string s all and only the desired extractions X_s. Furthermore, should capture the pattern describing the extractions, thereby generalizing beyond the provided examples. In other words, the examples constitute an incomplete specification of the extraction behavior of an ideal and unknown regular expression . The learning algorithm should infer the extraction behavior of .

3.3 Out tool

Our tool is available as a live web app[1] and in source code on GitHub[2]. Internally it is based on Genetic Programming (GP) and and described in full detail in [46, 77]. Chapter 3 provides all the details about our tool, here we provide only a brief outline, for convenience of the reader.

Our tool works as follows, we evolve a population of 500 regular expressions, represented by abstract syntax trees, by applying classical genetic operators such as mutation and crossover for 1000 iterations. We generate the initial population partly at random and partly based on the desired extractions, i.e., for each desired extraction x we generate 4 different regular expressions with a deterministic heuristics ensuring that all these expressions extract x. We drive evolution by means of a multiobjective optimization algorithm based on the length of regular expressions (to be minimized) and their extraction performance computed on the learning data (to be maximized). We use a separate-and-conquer heuristics for discovering automatically whether the extraction task may be solved by a single regular expression or whether a set R of multiple regular expressions, to be eventually joined by an "or" operator, is required [77]. In particular, every 200 iterations we check whether the currently best regular expression ri exhibits perfect precision on a subset X of the desired extractions. In that case, we remove X from the set of desired extractions, we insert r_i in R and we let the search continue (R is initially empty). Finally, we join all the elements in R and the best regular expression upon the end of the search by an "or" operator.

A screenshot of the web app is given in Figure 3.1. The user may load examples as UTF-8 files and then annotate text in these files graphically to identify desired extractions. The number of examples is irrelevant; what matters is the number of desired extractions: 10–20 usually suffice to obtain good solutions. We used 24 in the experiment described below. Examples and the resulting expressions may be saved for later analysis and reuse. systems.

3.4 The Challenge Platform

For our experiment, we developed a challenge web app for assisting human operators in the task of developing a regular expression for text extraction based on examples of the desired behavior[3]. The challenge web app starts by presenting concise instructions ("write a regular expression for extracting text portions which follow a pattern specified by examples") and asks the user to indicate his/her perceived level of familiarity with regular expressions: novice, intermediate, or experienced. Then, the challenge web app proposes a sequence of extraction tasks. Each task is presented as a text area in which the substrings to be extracted are highlighted.

[1] http://regex.inginf.units.it/
[2] https://github.com/MaLeLabTs/RegexGenerator
[3] http://play.inginf.units.it/

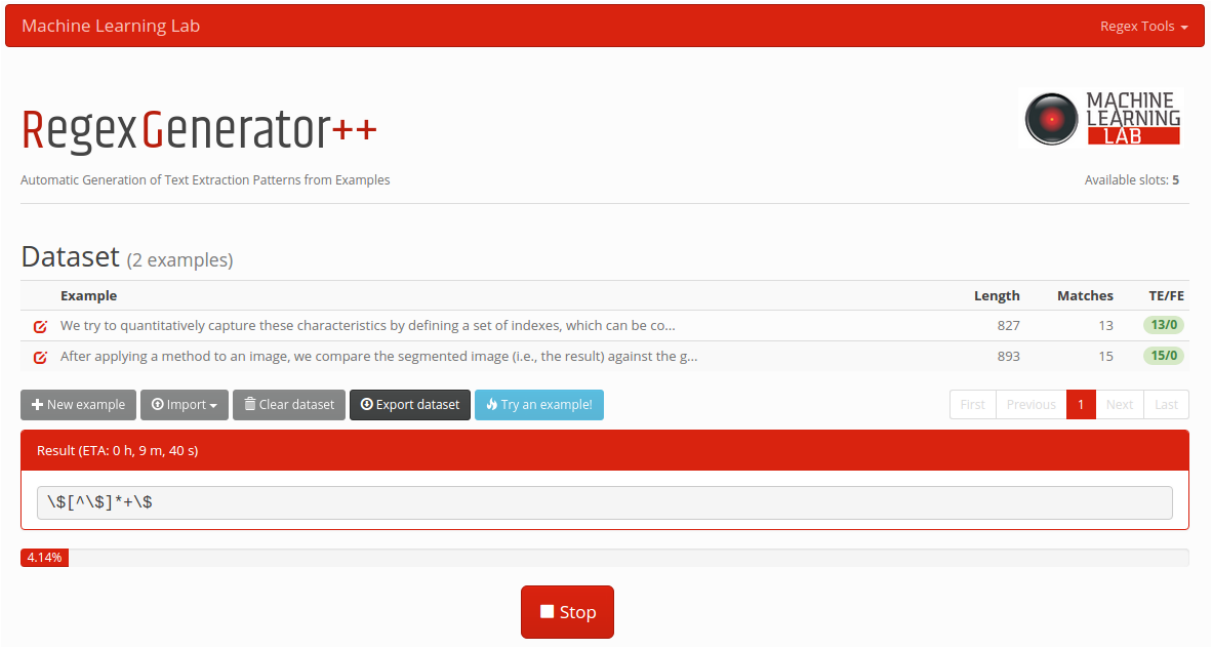


Figure 3.1: Snapshot of our tool taken during a search. The tool shows the best solution currently found.

The user writes a regular expression in a dedicated input field and the challenge web app highlights, with negligible latency, the substrings extracted by the expression along with the corresponding extraction mistakes. An example is in Figure3.2. The user may refine the regular expression interactively, that is, he may modify the expression at will and obtain an immediate feedback about the modified expression. We emphasize that the interactive nature of the challenge web app should make it easier for human operators to solve the proposed tasks, both in terms of quality of solutions and time required for their construction.

The challenge web app also shows the F-measure on the current task. To avoid the need of understanding what the F-measure actually represents, the user is informed that a value of 100% means a perfect score on the task. The user is not required to obtain a perfect F-measure before going to the next task and could even leave a task completely unanswered. Furthermore, the user need not execute all the tasks in a single session: when connecting, the challenge web app presents to the user the extraction task he was working on when disconnecting. The challenge web app records, for each task and for each user, the authored regular expression and the overall time spent on the task, excluding disconnection intervals.

In practice, users craft regular expressions in many ways. They may describe them using natural language, examples of matching strings, or with a combination of both. Users' descriptions may be underspecified, in the sense that they do not specify how every possible input sequence should be classified, and their descriptions can be refined during several iterations. The challenge web app specifies an extraction tasks solely by means of examples. This is necessarily an approximation of user behavior, but it nevertheless preserves the essence of the problem of constructing a regular expression, and it is simple for users to understand. The annotations can be done quickly, which is important because the challenge web app presented examples already annotated but a user willing to use our tool instead of crafting a regex would have to annotate examples. In the experiments described in the next section, we considered extraction tasks specified with 24 desired extractions; annotating the corresponding data took 1.5–2.5 minutes, depending on the task.

3.5 Procedure

We constructed 10 challenging extraction tasks, summarized in Table 3.1 and Table 3.2. Task names consist of the corpus name followed by the name of the entity type to be extracted: ReLIE-HTML: portions of a subset of the $50,000$ web pages obtained from the publicly available University of Michigan

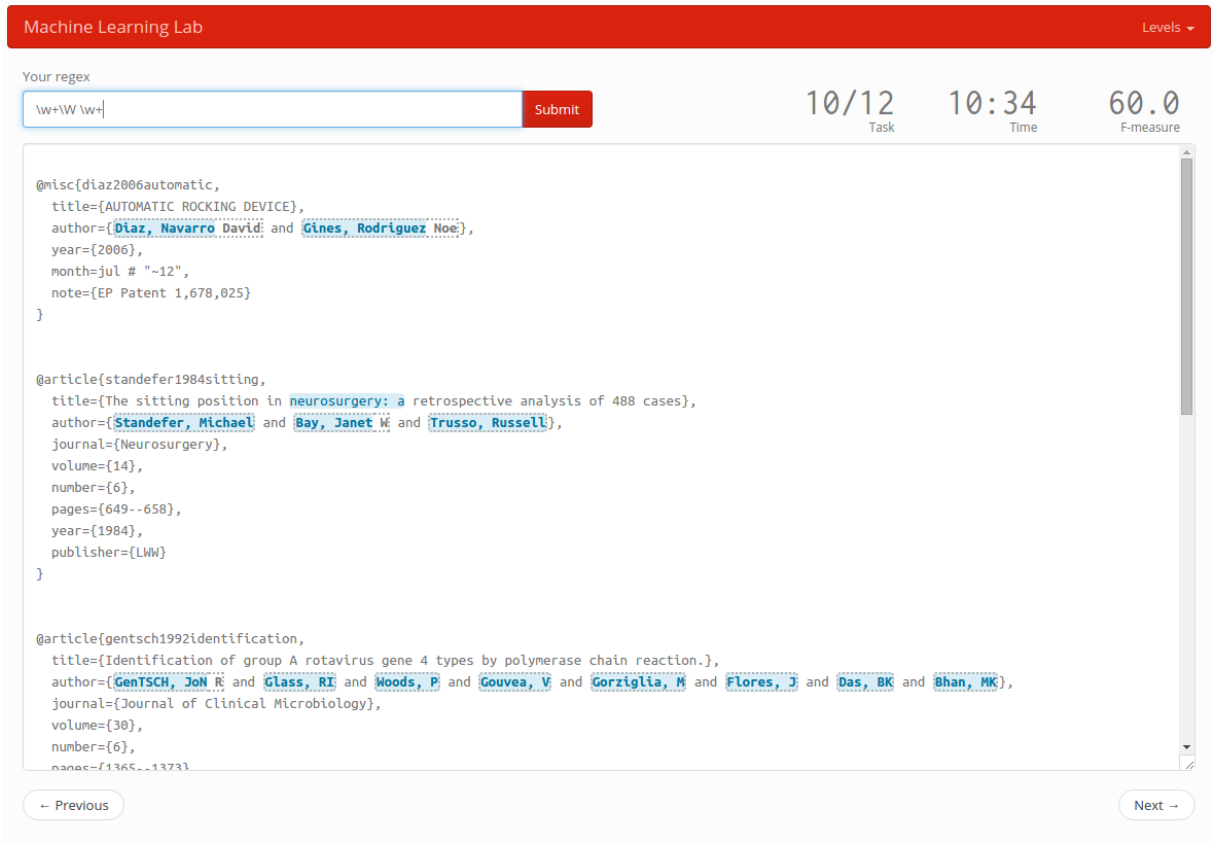


Figure 3.2: Snapshot of the challenge web app presented to users. The user has inserted the regex `\w+\W \w+` and the webapp highlights the extractions of this regex in blue: it can be seen that this regex results in undue extractions (i.e., highlighted text outside of the dashed boxes) and missed extractions (i.e., non highlighted text within the dashed boxes).

Web page collection (used also in [46, 85]). ReLIE-Email: portions of the $10,000$ emails obtained from the publicly available Enron email collection (used also in [85, 127]). Cetinkaya-HTML: full HTML source of 3 web pages (used also in [46, 60]). Cetin [46, 60]). Log: log entries collected from our lab firewall (used also in [46]). Web: full HTML source of a richer collection of web pages than Cetinkaya. BibTeX: BibTeX elements obtained by querying Google Scholar. References: references in the Springer LNCS format obtained from the BibteX corpus.

For each task, we randomly selected a set of examples containing 24 desired extractions (note that this corresponds, for each task, to a very small portion of the full corpus) and embedded the corresponding set in the web app. We published a post on Reddit encouraging users to challenge themselves [4]. Next, we executed our tool by using the very same set of examples as the learning set. We repeated each execution four times and averaged the performance indexes (see next section).

We chose not to distribute different sets of examples to different users because we did not expect to receive thousands of submissions and in a preliminary experiment we observed that many tasks were left unanswered. We thought that presenting different data to different users might have not allowed collecting a meaningful set of results. We have assessed the performance of our tool also with different learning sets, by executing a 5-fold procedure on each task. The resulting slight difference in the actual values of the indexes was negligible. We included two simple tasks at the beginning of the task sequence aimed solely at allowing users to practice and familiarize with the web app interface. We did not include these tasks in the analysis (their results are qualitatively similar to those of the other tasks, though).

[4] https://www.reddit.com/r/programming/comments/3eblji/how_good_are_you_in_writing_regex_challange/

3.6 Results

We gathered results from a large population: $1,764$ users participating from July 23-rd 2015 to September 20-th 2015. These users qualified themselves as follows: 44% novice, 38% intermediate, and 18% experienced. Users completed $10,439$ out of the $17,640$ tasks. Novice users completed 52% of the tasks, intermediate users 61%, and experienced users 71%.

We analyze results along two axes: quality of the solution assessed with F-measure and the time required for constructing the solution. We report average values for each category of users by taking into account only completed tasks with construction time between percentiles 1% and 99% (Figure3.3). Execution times for our tool have been obtained on a 6-core Intel Xeon 2.4 GHz with 32 GB RAM.

The key finding is that, on average, our tool delivered solutions with F-measure almost always greater than or equal to the one obtained by each category of human users, both on the learning data and on the testing data. Furthermore, on average, the time required by our tool was almost always smaller than the time required by human operators. We believe these results are remarkable and highly significant. Indeed, we are not aware of any similar tool exhibiting such human-competitive performance indexes.

By looking at the actual distributions of F-measure2, one may always find a significant fraction of humans which obtain better results than our tool. In other words, while our tool is not systematically better than humans, it does deliver F-measure that is comparable to humans and that, on average, is even better. Actual distributions of construction time indicate that our tool tends to be systematically faster than most humans on most tasks. This indication is also statistically significant.

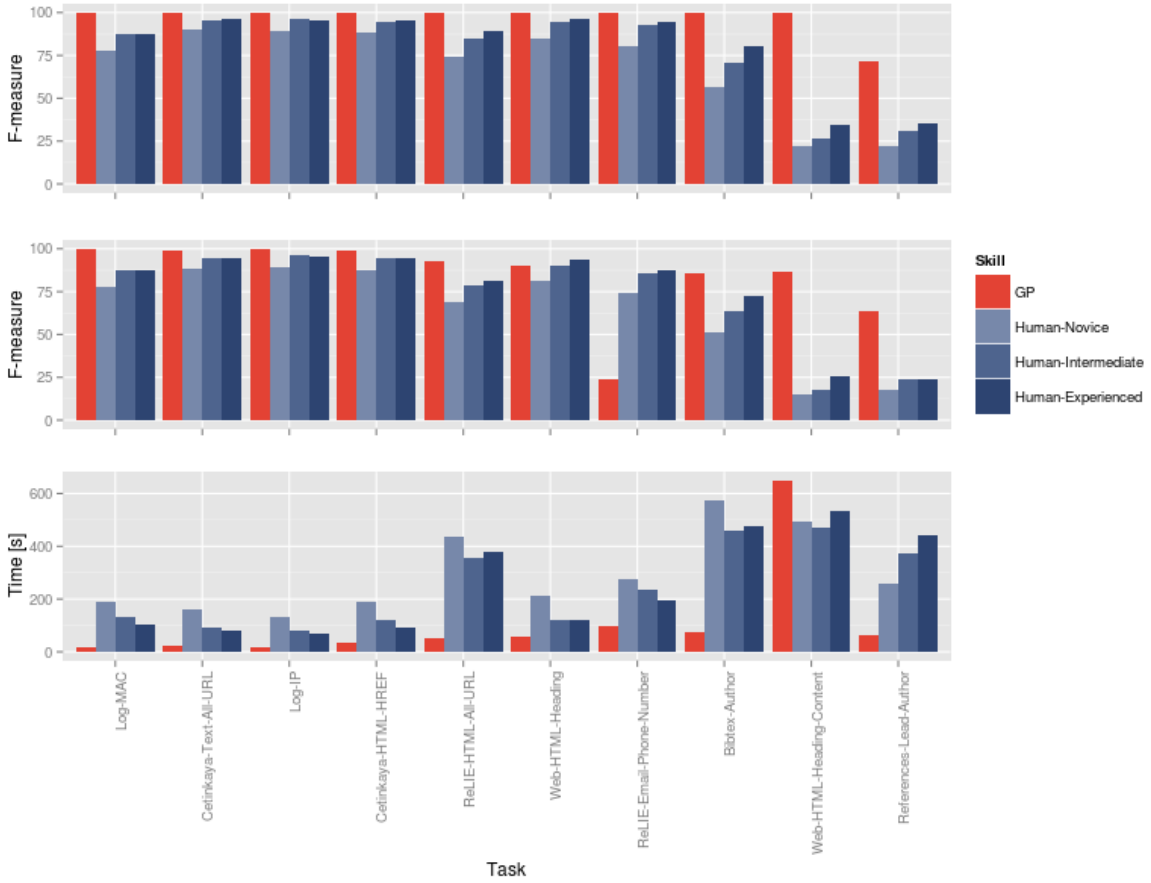


Figure 3.3: F-measure on the learning set (upper chart), F-measure on the remaining part of each dataset, i.e., on a hold-out testing set (middle), and the construction time (lower chart).

The only task in which our tool delivers unsatisfactory F-measure on the testing data, despite a very good value on training data, is ReLIE-Email/Phone-Number. A closer inspection of the dataset shows that,

for this task, the training data happens not to be adequately representative of the testing data, in particular, concerning substrings that look like phone numbers but are not. Executing our tool on a larger training set result in F-measure around 85%.

Table 3.1: Datasets Summary

Task name	Number of characters (x10^3)	Number of desired extractions
ReLIE-HTML/All-URL	4240	502
ReLIE-Email/Phone-Number	4240	499
Cetinkaya-HTML/HREF	154	214
Cetinkaya-Text/All-URL	39	168
Log/IP	4126	75958
Log/MAC	4126	38812
Web-HTML/Heading	4541	1083
Web-HTML/Heading-Content	4541	1083
Bibtex/Author	54	589
References/Lead-Author	30	198

Table 3.2: Datasets Snippets

Task name	example
ReLIE-HTML/All-URL	`Click here to access index history <http://www.intcx.com/SubscriberServlet/subscriberservlet.class?operation=powerIndexForm\&hub=All>. * volume represents sell-side only * Hub High Low Wtd Avg Index Change (\$) Vol (Mwh) Ciner`
ReLIE-Email/Phone-Number	`3784 SSWB  (734) 763-6276  <a href=\mailto:ddavies@umich.edu\>ddavies@umich.edu </a> </td> abs Client Services Center at:</FONT><TD align=middle><FONT face=Arial size=2> 734/936-2598 (Local), 800/862-7284 (Michigan Only) or 800/537-7284 (Outside Michigan) </FONT> </TD>`
Cetinkaya-HTML/HREF	`<a href="http://www.gutenberg.org/" target="\_parent">Project Gutenberg</a> <a href="http://www.scitation.org" target="\_-parent">Scitation</a>`
Cetinkaya-Text/All-URL	`Fedora Extras http ftp rsync ftp://ftp7.br.FreeBSD.org/pub/FreeBSD/ (ftp) ftp://ftp3.de.FreeBSD.org/pub/FreeBSD/ (ftp) ftp://ftp.is.FreeBSD.org/pub/FreeBSD/ (ftp / rsync)`
Log/IP	`Jan 13 05:49:47: ACCEPT service dns from 74.125.189.23 to firewall(pub-nic-dns), prefix: "none" (in: eth0 74.125.189.23(00:80:38:fa:8a:7e) :51027 -> 140.105.63.158(00:00:76:fe:75:e2):53 UDP len:80 ttl:49)`

Table 3.2: Datasets Snippets

Task name	example
Log/MAC	`Jan 13 17:44:52: DROP service 68->67(udp) from 172.45.240.237 to 217.70.177.60, prefix: "spoof iana-0/8" (in: eth0 216.34.90.16(00:21:91:fe:a2:6f):68 -> 69.43.85.253(00:07:e1:7c:53:db):67 UDP len:328 ttl:64)`
Web-HTML/Heading	`e se non fosse che 'n sul passo d'Arno [modifica] Infra strutture e trasporti Visite Libero.HF.adjust800 = function () \{`
Web-HTML/Heading-Content	`e se non fosse che 'n sul passo d'Arno [modifica] Infra strutture e trasporti Visite Libero.HF.adjust800 = function () \{`
Bibtex/Author	`@inproceedings\{arellano2004study, title=\{Study of the structure changes caused by earthquakes in Chile applying the lineament analysis to the Aster (Terra) satellite data.\}, author=\{Arellano-Baeza, A and Zverev, A and Malinnikov, V}, booktitle=\{35th COSPAR Scientific Assembly\},`
References/Lead-Author	`130. Andrews, D.G., Holton, J.R., Leovy, C.B.: Middle atmosphere dynamics. Number 40. Academic press (1987)`

3.7 Remarks

While we do not claim that a tool like ours may be effective in each and every possible application of regular expressions, we do believe to have provided strong indications that a machine may indeed constitute a practically viable tool for synthesizing regular expressions from scratch. In our challenging tasks, the machine has proven its ability to surrogate the expertise and skills required by human programmers. We believe that this result is relevant in itself and, more broadly, as a further demonstration of the practical capabilities of Genetic Programming techniques even on commodity hardware.

An issue that we have not yet addressed is readability of the solutions. While this property is orthogonal to F-measure, it may nevertheless be important in practice: users might not trust a result that

they do not fully understand or whose behavior in corner cases might be difficult to predict. As an aside, these remarks apply also to other popular machine learning paradigms, e.g., neural networks. Manual inspection of a few solutions suggest that human operators tend to construct shorter solutions, but we could not find any clear cut between the categories: even automatically-constructed solutions may be very compact and highly readable; and, there is ample variability between operators with task difficulty playing a key role.

We plan to assess readability of the solutions as part of a broader investigation on this important question: what are the key differences between solutions constructed by human programmers and automatically-constructed solutions? Is it possible to distill such differences—for example including readability—into a fitness definition capable of driving the evolutionary search toward regions of the solution space closer to those explored by human operators? We believe that ideas of this kind may provide an exciting line of research in evolutionary computing.

Chapter 4

Regex-based Entity Extraction with Active Learning

4.1 Overview

A large class of entity *extraction* tasks from *unstructured data* may be addressed by *regular expressions*, because in many practical cases the relevant entities follow an underlying syntactical pattern and this pattern may be described by a regular expression. A long-standing problem in this area consists in the *automatic* generation of a regular expression suitable for a specific task based solely on *examples* of the desired behavior.

A wealth of research efforts in this area considered *classification* problems either in *formal* languages [45, 77, 78, 84, 117, 199] or in the realm of *deterministic finite automata* (DFA) [40, 60, 126, 140]. Those results considered scenarios that do not fit practical text processing applications, which have to cope with much longer sequences of symbols drawn from a much larger alphabet. Text *extraction* problems of non trivial size and complexity were first considered in a procedure that automatically optimized an initial regular expression to be provided by the user based on examples of the desired functioning [133]. Later proposals still required an initial regular expression but were more robust toward initial expressions of modest accuracy and noisy datasets [10, 152]. The need of an initial solution was later removed in several proposal [12, 44, 55]. A more recent proposal based on Genetic Programming advanced significantly over earlier approaches and is capable of addressing text extraction tasks of practical complexity effectively, with a few tens of examples of the desired behavior [15, 17].

In this chapter, we investigate the feasibility of an *active learning* approach for relieving the user from the need of examining the full input text (i.e., the dataset) in search of all the desired extractions to be annotated for learning [6, 52, 139, 186, 194]. We develop and evaluate experimentally a framework in which the user initially marks *only one* snippet of the input text as desired extraction. A learner based on Genetic Programming then constructs a solution, digs into the (possibly very long) input text, selects the most appropriate snippet to be used for improving the current model and presents it to the user as an *extraction query*. The user merely answers the query by specifying whether the selected snippet has to be extracted or not extracted and the process continues iteratively, improving the solution at each query.

The resulting framework is highly attractive and may greatly broaden the potential scope of automatic regex generation from examples. On the other hand, actually implementing this framework is challenging because the scenario presents significant differences from successful applications of active learning.

Active learning approaches usually consider datasets where each item is an input instance and thus a candidate query. This property is shared also by approaches based on Genetic Programming [70, 113, 155]. Our case is different because the dataset is a single, possibly long, input text without any native segmentation in smaller units. Depending on the application, it may consist of one very long line or several lines with possibly variable length; furthermore, more than one desired extractions may occur within a single line (e.g., IP addresses in network logs) or a single desired extraction may span across

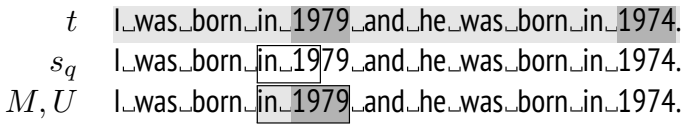


Figure 4.1: Oracle annotation example: desired (undesired) extractions are in dark (light) gray; the query is boxed.

several lines (e.g., HTML elements including their content). Assuming that the text is natively segmented in lines or in sentences (as in, e.g., [82]) would severely restrict the scope of possible applications of the system. Moreover, the size of the query to be presented to the user should be chosen carefully. Presenting a large snippet (e.g., one or more entire lines) to the user for annotation may nullify the objective of minimizing user annotation effort. On the other extreme, repeatedly asking the user to annotate very short snippets may not be effective.

In other words, not only we have the problem of *choosing* the next query among candidate queries, we also have the problem of *constructing* candidate queries out of the available input text. In this respect, it is useful to remark that the number of possible queries in (i.e., the number of snippets of) an input text grows quadratically with the text size and becomes huge very quickly—e.g., even if we assume that the learner cannot generate queries ex novo and can only query a snippet of the input text, if the latter size is just 10^5characters then there are $\approx 10^{10}$ candidate queries. Furthermore, active learning usually targets scenarios with hundreds of queries (e.g., [52, 139, 194]) whereas we must be able to improve over a random query chooser and provide solutions of good quality even with a few tens of examples, similarly to [113].

Our contribution consists in: (a) a model for the external oracle that may participate in the construction of queries, which improves the quality of annotation information while at the same time maintaining a behavior very intuitive to unskilled users; (b) a technique for constructing queries suitable to regex-based entity extraction from unstructured text, which does not assume any internal segmentation of input text; (c) an implementation of several active learning approaches taking into account the need of constructing candidate queries; (d) a novel variant for the learner in which the number of generations executed between consecutive queries may vary dynamically depending on the quality of the current solution; and, (e) an experimental analysis on a number of challenging datasets of several active learning approaches, which target different accuracy/annotation effort regions of the design space.

4.2 Our approach

The problem consists in generating a regular expression automatically based on examples of the desired extraction behavior on a text t. Such examples are *annotations*: snippets of t that are to be extracted (*matches*) or snippets of t that are not to be extracted (*unmatches*).

We propose an approach based on active learning, as follows. Initially an external *oracle*, i.e., the user, annotates an extremely small portion of t—we experimented with only one match. The learner consists of three components: the *solver*, which generates a regular expression suited to the matches and unmatches annotated by the oracle so far; the *query trigger*, which determines *when* a query has to be proposed to the oracle; and the *query builder*, which *constructs* candidate queries and determines *which query* should be proposed to the oracle.

Each query consists of a snippet of t, denoted s_q, to be annotated by the oracle. We propose the following behavior for the oracle: the oracle's answer is a pair M, U, where M is the (possibly empty) set of all matches which overlap s_q and U is the (possibly empty) set of maximal subsnippets of s_q which are unmatches—Figure 4.1 shows an example of annotation.

In other words, we propose an oracle that may modify the received query slightly and then answer the modified query. With most active learning approaches the user is required to provide the class of queried data and is not allowed to modify those data. The proposed behavior for the oracle is very practical and is easily implemented with a GUI, though. When the queried snippet consists exactly of a desired extraction or does not contain any desired extraction, one single click suffices to answer the query.

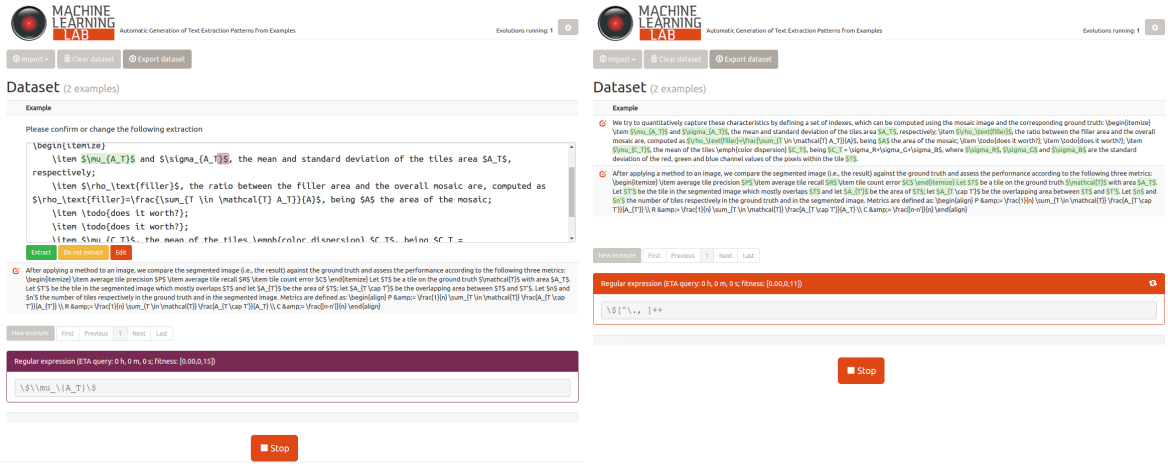


Figure 4.2: Screenshots of the web-based prototype developed for our framework: query submitted to the user (left), learning based on the currently available annotations (right).

Otherwise, when the query partly overlaps a match, the user is expected to expand the query on either or both sides—an action which is more intuitive to unskilled users, nevertheless results in answers which are more informative to the learner.

We developed a web-based prototype with a GUI that efficiently implements the proposed interaction model. Figure 4.2 shows how the user interface appears when a query is made (left) and while the learning algorithm is running (right). In the first case, a query is shown as a highlighted portion of the text (in purple) t and the user is presented with 3 buttons: 'Extract", "Do not extract" and "Edit". When the query corresponds exactly to a desired extraction or does not contain any desired extraction, then one single click suffices to answer the query (button "Extract" or "Do not extract", respectively). Otherwise, when the user has to describe a more complex answer, by clicking the "Edit" button the user may extend the selection boundaries of the query and delimit desired extractions precisely. The GUI also highlights (in green) the extractions of the current best solution, in order to help the user in understanding the behaviour of the current solution. The state of the current solution is reported also while the search is in progress, as illustrated in the left part of Figure 4.2. The aim of this design is to help the user in deciding when to stop the regex search—i.e., when the user is satisfied by the current solution.

The solver is based on the proposal in [17, 23], whose code is publicly available[1]. The proposal is based on Genetic Programming [120]: a population of regular expressions, represented by abstract syntax trees, is iteratively evolved by applying the genetic operators across many iterations (*generations*). A multiobjective optimization algorithm drives evolution of regular expressions according to their length (to be minimized) and their extraction performance computed on the matches and unmatches (to be maximized). We refer the reader to the cited paper for full details.

We considered two variants for the query trigger. The *Const* variant has been used in other active learning proposals for Genetic Programming [70, 113, 155] and generates a new query whenever a predefined number of generations of the solver has been executed. W experimented with 30 and with 200 generations. The *Solved* variant is an optimization that we explore in this work. This variant triggers the query builder when the best regular expression in the population, as assessed on the current set of matches and unmatches, has remained unchanged for a predefined number of generations of the solver—i.e., a new query is triggered when no further progress seems to be achievable with the available annotations. We experimented with 200 generations, i.e., one of the values selected for the *Const* variant, in order to assess the accuracy/speed trade-off of the two variants.

The query builder constructs candidate queries based on the notion of *disagreement*: given a set C of regular expressions (the *committee*), we define as disagreement of C on a character c of the input text t the quantity $d_C(c) = 1 - 2\text{abs}\left(\frac{1}{2} - \frac{|C_c|}{|C|}\right)$, where $C_c \subseteq C$ is the subset of regular expressions which

[1] https://github.com/MaLeLabTs/RegexGenerator

extract c—$d_C(c) = 1$ if half of the committee extracts c (maximum disagreement), $d_C(c) = 0$ if the all committee agrees on the processing of c (minimum disagreement). Note that we quantify disagreement based on the class chosen by each candidate solution in C (extracted vs. not extracted) [141] without any reference to forms of confidence value, margin or probability [131, 147]. As we pointed out already in the introduction, such notions are not made available by the solver that we have chosen to use.

The procedure for constructing candidate queries takes a set of regular expressions C as parameter and determines the character $c^* \in t$ with maximal disagreement $d_C(c^*)$ in the full input set t. Next, the procedure determines the set S of candidate queries as the set composed of all snippets of t which meet the following conditions: they (a) are extracted by at least a regular expression in C, (b) overlap c^*, and (c) do not overlap any available annotation.

We implemented two variants of a query builder. The *Query by committee (QbC)* variant works as follows: (a) construct the set S of candidate queries using the full population as committee C, (b) compute, for each snippet in S, the average disagreement among the characters of the snippet, and (c) choose the snippet with minimal average disagreement as query. The *Query by restricted committee (rQbC)* variant is similar to QbC except that the committee C contains only the best 25% of the current population (ranking being based on the current set of matches and unmatches).

QbC and rQbC are based on a principle widely used in active learning [160, 186], i.e., on the assumption that the query for which an ensemble of competing hypotheses exhibits maximal disagreement is the most informative for the learning task [187]. Such a principle has been used also in active learning for Genetic Programming [70, 113, 155]—in those scenarios there is the problem of choosing a candidate query but not the one of constructing queries, though. Indeed, the proposal in [113] augments this principle by also taking into account a measure of diversity between each candidate query and queries already answered. Our preliminary exploration of this additional principle, that we do not illustrate for space reasons, has not delivered satisfactory results. We believe the reason consists in the difficulty of finding a diversity measure for text snippets correlated with diversity between regular expressions—e.g., two text snippets could be very different while at the same time they could be captured by the same regular expression or by regular expressions that are very similar.

Concerning query builders we also observe that a wealth of active learning approaches choose queries based on *uncertainty* of the current solution, especially when the learner is not based on an ensemble of competing hypotheses [131, 160, 180, 194]. On the other hand, such approaches do not fit the state-of-the-art regex learner that we use in our system, because such a learner does not provide any confidence level about the handling of a given snippet (i.e., extracted vs. not extracted) by the current solution.

We also implemented a third query builder that randomly chooses an unannotated snippet. We place an upper bound to the maximum length of the query that may be generated: we set the actual bound value in our experimental evaluation to the maximum size of a desired extraction across all our datasets (few hundreds characters). The upper bound causes this query builder to filter out candidate queries which are too long, which hence advantages this builder w.r.t. one which selects a truly random snippet of t. For this reason, we call this builder *SmartRand*.

4.3 Experiments

We focused on the *extraction performance* of the regular expression generated for a given amount of *user annotation effort*. We quantify extraction performance with F-measure (Fm), which is the harmonic mean of precision (ratio between the number of correctly extracted snippets and the number of all the extracted snippets) and recall (ratio between the number of correctly extracted snippets and the number of all the snippets which should have been extracted). We chose to quantify user annotation effort by the number of annotated characters (AC).

We evaluated all the 9 combinations between the proposed design variants and we considered 11 challenging extraction tasks used in [17]. For each extraction task, we randomly selected a subset of the original corpus containing approximately 100 desired extractions. The name of each extraction task can be seen—along with the size of the input text expressed in number of characters—in Table 4.1: it is

Table 4.1: The F-measure obtained with each variant on each task. The average F-measure, CE and AC are also shown.

Task	Size	QbC Const30	QbC Const200	QbC Solved	SmartRand Const30	SmartRand Const200	SmartRand Solved	rQbC Const30	rQbC Const200	rQbC Solved
ReLIE-HTML/All-URL	16 655	0.61	0.82	0.76	0.75	0.86	0.82	0.69	0.84	0.85
ReLIE-Email/Phone-Num.	18 123	0.97	0.99	0.99	0.57	0.95	0.70	0.97	0.98	0.99
Cetinkaya-HTML/href	14 922	0.77	0.87	0.88	0.95	1.00	1.00	0.81	1.00	1.00
Cetinkaya-Text/All-URL	7573	0.98	0.99	0.99	0.99	0.99	0.99	0.98	0.99	0.99
Twitter/Hashtag+Citation	5308	0.91	0.93	0.98	0.98	0.99	0.99	0.92	0.99	0.99
Twitter/All-URL	9537	0.92	0.92	1.00	1.00	0.92	1.00	0.92	1.00	1.00
Log/IP	5766	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Log/MAC	10 387	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Email-Headers/IP	36 925	0.89	0.80	0.94	0.39	0.85	0.53	0.69	0.71	0.77
NoProfit-HTML/Email	4651	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Web-HTML/Heading	37 678	0.51	0.54	0.54	0.81	0.75	0.82	0.52	0.83	0.60
Average Fm		0.87	0.89	0.91	0.86	0.94	0.89	0.86	0.94	0.92
Average AC		3311	3202	2734	2997	2864	2646	3238	2506	2525
Average CE ($\times 10^9$)		6.5	45.4	44.2	4.3	33.6	27.3	7.2	40.0	27.2

composed of the name of the corpus followed by the name of the entity type to be extracted.

We assessed our system variants as follows. For each task and variant, we chose a random desired extraction as the only starting annotated snippet and executed the variant with a simulated oracle. We repeated the above procedure 15 times, with 5 different starting matches and 3 different random seeds. We terminated each execution upon the query for which either at least 25% of the available characters was annotated or the F-measure on the full input text (i.e., not only on the annotated portion) was 1. Although a real deployment cannot quantify F-measure on a yet unannotated input text, we chose to include the latter condition in the termination criterion in order to provide a fair assessment of variants which are able to generate perfect solutions before reaching the predefined annotation budget. We chose 25% of the available characters as annotation budget because we have found that, with these datasets, it corresponds to a few minutes of actual annotation.

Table 4.1 shows the main results (statistical significance is analyzed in more detail later). For each task, Fm is computed on the full input text and averaged across the 15 repetitions of each experiment. Values in the bottom rows of the table are averaged across all tasks. We define the *computational effort* (CE) as the number of characters analyzed for fitness evaluations across an execution. This quantity is a hardware-independent performance index. Execution times are in the order of minutes, similarly to [17], we do not list them in detail for space reasons: the time taken by the query trigger and the query builder is negligible w.r.t. the time taken by the solver.

It can be seen that for nearly all tasks several of our active learning variants are able to generate regular expressions of very good quality. This result is significant because it strongly suggests that active learning is indeed a viable framework for the task of automatic generation of regular expressions.

Another important outcome is that the rQbC query builder tends to deliver better F-measure than the SmartRand query builder while requiring less annotations—ΔFm between 0.05 and 0.1 on the average. In many applications of active learning, a random query chooser is often quite effective and often turns out to be a challenging baseline for more sophisticated query choice strategies [9, 86, 194]. Although we may observe this phenomenon also in our scenario (in which the random selection is enhanced by a lenght-based filtering, see Section 4.2), we also observe a clear superiority of approaches based on rQbC. The QbC query builder, on the other hand, is not effective as it tends to exhibit worse results from the three points of view summarized in the table: F-measure, annotation effort, computational effort.

We speculate that the superiority of rQbC over SmartRand may become even more apparent with datasets in which the density of desired extractions is smaller than ours—in our datasets, the likelihood of randomly choosing a snippet that partly overlaps a desired extraction is not very small. We need to investigate this conjecture further, however.

Concerning the behavior of query triggers with rQbC, it can be seen that each of the three options analyzed belongs to a different region of the design space. The Const30 query trigger is much faster (CE) at the expense of obtaining a relatively good but smaller F-measure, while at the same time requiring more

annotations (AC). Const200 and Solved represent more useful trade-offs because they deliver the best average F-measure: they require the same amount of annotations, trading a small difference in F-measure for a substantial difference in computational effort.

In order to illustrate the significance of these results further, we executed the state-of-the-art learner proposed in [17] on the same tasks. This learner requires a training set fully annotated before starting execution. For each task we randomly generated 5 training sets, each one with 25% of the available characters and with a random generation procedure carefully tailored so as to ensure that each training set contains approximately 25% of the desired extractions. It may be useful to emphasize that the size of the training set corresponds to the size of the training set of active learning upon the *last* query: in this case the training set is instead available to the solver for the *full* execution; furthermore, in active learning the user need not take any effort to dig out an adequate amount of desired extractions from the (potentially large) available data. We executed each task 5 times, each execution using one of the 5 different training sets. We obtained, on average, Fm $= 0.97$, CE $= 29.8 \times 10^9$ and AC $= 3748$, i.e., 49% more annotated characters than rQbC-Const200 and 48% more than rQbC-Solved.

We performed an analysis of the statistical significance of the results based on the Wilcoxon signed-rank test: we chose this test since it is non-parametric and does not require the population to be normally distributed. The results are in Table 4.2 (F-measure, above, and annotated characters, AC, below)—we omit results about CE for space reasons. In each table, cell (i, j) contains the difference in the average value of the corresponding performance index between variant in row i and variant in row j. Statistical significance of performance index comparison is indicated for varying p-values of the test and highlighted with asterisks.

These results confirm the analysis of Table 4.1, but they also indicate that the rQbC/Const200 and rQbC/Solved actually does not guarantee any statistically significant improvement in Fm over SmartRand/Const200. On the other hand, there is indeed some statistically significant evidence of an improvement in terms of smaller annotation effort—12.5% for rQbC/Const200 and 11.8% for rQbC/Solved. Concerning CE (not shown for space reasons), rQbC/Const200 requires 19% more character evaluations but this result is not statistically significant; rQbC/Solved instead requires 19% less character evaluations with the strongest statistical significance.

Figure 4.3 illustrates the trade-off AC vs. F-measure (left) and AC vs. CE (right). The figure contains one point for each task; the different query triggers are represented as points of different colors while the different query builders are represented with different shapes. For each point, F-measure, AC and CE are averaged among 15 experiment repetitions—5 folds and 3 different random seeds.

In the left figure it can be seen that points representing the Const30 query trigger—light grey points—tend to be distributed in the rightmost and lower part of the figure—i.e., this query trigger requires high AC but obtains low F-measure. Points representing the Const200 and Solved query trigger—dark gray and black points—tend instead to be distributed in the leftmost and higher part of the figure, i.e., for each AC value we may obtain high F-measure values. Concerning query builders, the graphical distribution of points does not provide any significant insights; in this respect, the other analyses discussed previously are more effective. In the right figure shows for each point the average CE vs the average AC for one task it can be seen that points representing the Const200 and Solved query triggers tend to be distributed in the highest part of the figure, as expected Const200 and Solved query triggers require CE values higher than the Const30 ones. We may note that the points representing the SmartRand query builder tend to occupy the highest part of the figure, in other words SmartRand query builders require CE values higher than the QbC and rQbC ones.

Finally, in Table 4.3 we report the detailed execution trace of two significant experiments based on the rQbC/Solved configuration: one for the Twitter/Hashtag+Citation task and another for the Email-Headers/IP task. The table contains one row for each query constructed by the system. Each row contains: the sequential number of the query; the number of annotated matches $|\bigcup M|$ and unmatches $|\bigcup U|$, available to the learning algorithm; the content of the query s_q; the response provided by the user in terms of desired matches M and desired unmatches U. Each row also contains the currently best solution, along with the F-measure associated with such solution and the total amount of AC.

Table 4.2: Average differences of Fm and AC of pairs of the proposed variants. For each pair, the statistical significance is shown: *: $p < 0.1$, **: $p < 0.05$, ***: $p < 0.01$ (the last condition corresponds to the strongest statistical significance; absence of any asterisk indicates that the comparison is not statistically significant, i.e., $p \geqslant 0.1$).

F-measure (Fm)

Variant	QbC Const30	QbC Const200	QbC Solved	SmartRand Const30	SmartRand Const200	SmartRand Solved	rQbC Const30	rQbC Const200	rQbC Solved
QbC/Const30		−0.03***	−0.05***	0.01	−0.07***	−0.03**	0.01*	−0.07***	−0.05***
QbC/Const200	0.03***		−0.02*	0.04**	−0.04**	0.00	0.03***	−0.04***	−0.03***
QbC/Solved	0.05***	0.02*		0.06***	−0.02	0.02	0.05***	−0.02**	−0.01***
SmartRand/Const30	−0.01	−0.04**	−0.06***		−0.08***	−0.04***	−0.01	−0.08***	−0.07***
SmartRand/Const200	0.07***	0.04**	0.02	0.08***		0.04***	0.07***	0.00	0.01
SmartRand/Solved	0.03**	0.00	−0.02	0.04***	−0.04***		0.03**	−0.04***	−0.03***
rQbC/Const30	−0.01*	−0.03***	−0.05***	0.01	−0.07***	−0.03**		−0.07***	−0.06***
rQbC/Const200	0.07***	0.04***	0.02**	0.08***	0.00	0.04***	0.07***		0.01
rQbC/Solved	0.05***	0.03***	0.01***	0.07***	−0.01	0.03***	0.06***	−0.01	

Annotated characters (AC)

Variant	QbC Const30	QbC Const200	QbC Solved	SmartRand Const30	SmartRand Const200	SmartRand Solved	rQbC Const30	rQbC Const200	rQbC Solved
QbC/Const30		109***	577***	314***	447***	665***	73**	805***	786***
QbC/Const200	−109***		468***	205**	338***	556***	−35	696***	677***
QbC/Solved	−577***	−468***		−263**	−129	88	−503***	228**	209**
SmartRand/Const30	−314***	−205**	263**		133***	351***	−240***	491***	472***
SmartRand/Const200	−447***	−338***	129	−133***		218***	−374***	358*	338*
SmartRand/Solved	−665***	−556***	−88	−351***	−218***		−592***	140	120
rQbC/Const30	−73**	35	503***	240***	374***	592***		732***	712***
rQbC/Const200	−805***	−696***	−228**	−491***	−358*	−140	−732***		−19***
rQbC/Solved	−786***	−677***	−209**	−472***	−338*	−120	−712***	19***	

Table 4.3: Sequences of queries generated for two different experiments. For each query are reported the total number of matches and unmatches annotated, the query s_q, the user answer in terms of M and U, the current best solution, the corresponding F-measure and the current AC.

#	$\lvert\bigcup M\rvert$	$\lvert\bigcup U\rvert$	s_q	M	U	Best regex	F-m	AC
1	1	0	#20topsongsever	#20topsongsever		`#\w++`	0.39	24
2	2	0	#hacking	#hacking		`#\w++`	0.39	32
3	3	0	#ti	#tips		`#\w++`	0.39	37
4	4	0	#O	#OpPiggyBank		`#\w++`	0.39	49
5	5	0	#plurfamily	#plurfamily		`#\w++`	0.39	60
6	6	1	#FF␣mee␣!!!"	#FF	␣mee␣!!!"	`#\w++`	0.39	72
7	7	2	#bast@Rd	#bast	@Rd	`#\w++`	0.39	80
8	8	2	@Callum	@Callum␣Rose		`[@#]\w++`	1.00	92
1	1	0	199.87	199.87.247.43		`199\.87\.247\.43`	0.08	26
2	2	0	209.85.216.170	209.85.216.170		`\w++\.\w++\.\w++\.\w++`	0.77	40
3	3	0	10.2	10.231.24.9		`\w++\.\w++\.\w++\.\w++`	0.77	51
4	4	2	:␣by␣10.231.102.195␣with␣SMTP	10.231.102.195	:␣by␣ ␣with␣SMTP	`(?:\d++\.)++\d++`	0.68	80
5	5	2	10.236.195.3	10.236.195.34		`(?:\w++\.)++\d++`	0.66	93
6	5	3	go2mr11586177wib.22		go2mr11586177wib.22	`\w++\.\w++\.\w++\.\w++`	0.77	112
7	5	4	etPan.528e775f.6ce90669.a		etPan.528e775f.6ce90669.a	`\d++\.\d++\.\d++\.\d++`	0.92	137
8	6	4	199.7.202.190	199.7.202.190		`\d++\.\d++\.\d++\.\d++`	0.92	150
9	7	4	199.7.202.190	199.7.202.190		`\d++\.\w++\.\w++\.\w++`	0.84	163
10	7	5	Exim␣4.80.1		Exim␣4.80.1	`\d++\.\w++\.\w++\.\w++`	0.84	174
11	7	6	h6mr48792qew.9		h6mr48792qew.9	`\w++\.\w++\.\w++\.\d++`	0.87	188
12	7	7	5.1gphBQfkbkwG8rjXKOhM		5.1gphBQfkbkwG8rjXKOhM	`\w++\.\w++\.\w++\.\d++`	0.87	210
13	7	8	x8mr5889809oek.49.1		x8mr5889809oek.49.1	`\w++\.\w++\.\w++\.\d++`	0.87	229
14	7	9	jx4mr3506406vec.35.1		jx4mr3506406vec.35.1	`\w++\.\w++\.\w++\.\d++`	0.87	249
15	7	10	6.0.3790.4		6.0.3790.4	`\w\w++\.\w++\.\d++\.\d++`	0.91	259
16	7	11	1.2013.11.11		1.2013.11.11	`\w\d++\.[^1]\w*+\.(?!3)\d++\.\d++`	0.80	271
18	8	13	217.12.10.166;	217.12.10.166	;	`\w\d++\.\w++\.\d*+\.\d++`	0.95	360

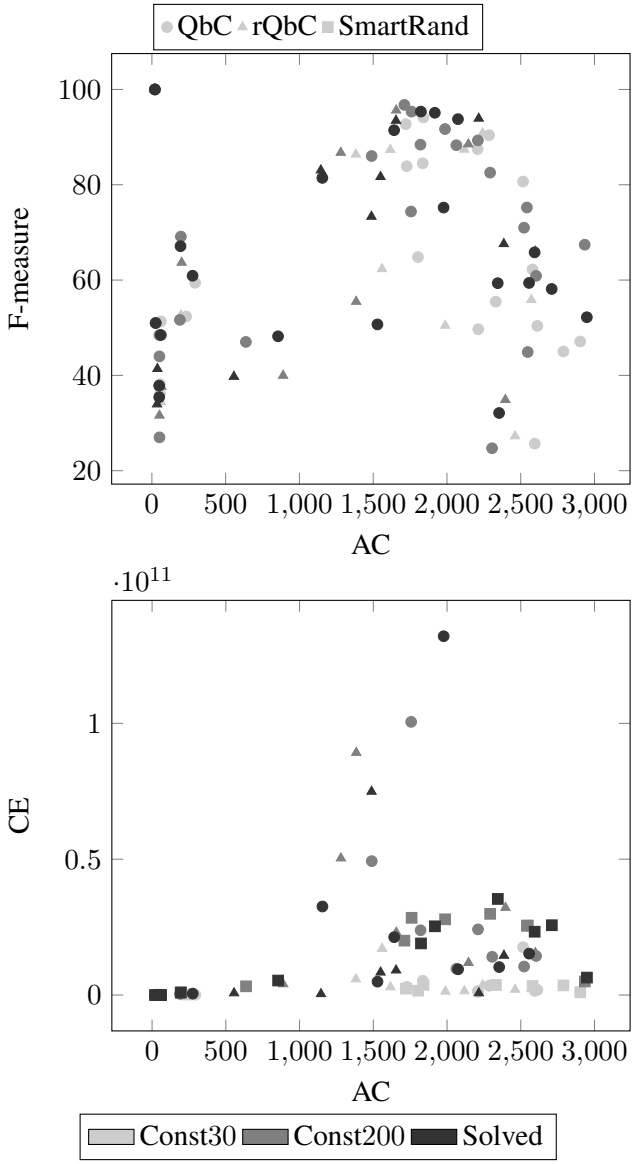


Figure 4.3: AC vs. F-measure (left) or vs. CE (right): one point for each task (corresponding to the average index across the repetitions).

4.4 Remarks

In this chapter we have proposed several active learning approaches tailored to the automatic generation of regular expressions for entity extraction from unstructured text. We have assessed these approaches experimentally on a number of challenging extraction tasks that have been previously used in the literature. The results indicate that active learning, starting with only one annotated match, is indeed a viable framework for this application domain and may thus significantly decrease the amount of costly user annotation effort. We have also identified design options and explored the design space, in terms of computational effort and annotation effort, while delivering very good F-measure. We believe that our results are significant and highly promising.

As future work we intend to broaden the experimental analysis by taking into account more facets of the user effort, including a measure of the user annotation time as a function of the number, length and complexity of individual queries. We also intend to devise a suitable metric for taking into account the user cost broadly involved in the elapsed time between consecutive queries. In the following chapters we may explore the feasibility of an online estimate of the difficulty of obtaining a suitable regular expression given the current set of annotations.

Chapter 5

Predicting the Effectiveness of Pattern-based Entity Extractor Inference

5.1 Overview

An essential component of any workflow leveraging digital data consists in the identification and extraction of relevant *patterns* from a data stream. This task occurs routinely in virtually every sector of business, government, science, technology, and so on. In this chapter we are concerned with extraction from an unstructured text stream of entities that adhere to a *syntactic* pattern. We consider a scenario in which an extractor is obtained by tailoring a generic tool to a specific problem instance. The extractor may consist, e.g., of a regular expression, or of an expression in a more general formalism [98], or of full programs suitable to be executed by NLP tools [119, 169]. The problem instance is characterized by a dataset from which a specified entity type is to be extracted, e.g., VAT numbers, IP addresses, or more complex entities.

The difficulty of generating an extractor is clearly dependent on the specific problem. However, we are not aware of any methodology for providing a practically useful answer to questions of this sort: generating an extractor for describing IP addresses is more or less difficult than generating one for extracting email addresses? Is it possible to generate an extractor for drug dosages in medical recipes, or for ingredients in cake recipes, with a specified accuracy level? Does the difficulty of generating an extractor for a specified entity type depend on the properties of the text that is *not* to be extracted? Not only answering such questions may provide crucial insights on extractor generation techniques, it may also be of practical interest to end users. For example, a prediction of low effectiveness could be exploited by providing more examples of the desired extraction behavior; the user might even decide to adopt a manual approach, perhaps in crowdsourcing, for problems that appear to be beyond the scope of the extractor generation technique being used.

In this work we propose an approach for addressing questions of this sort systematically. We consider on a scenario of increasing interest in which the problem instance is specified by *examples* of the desired behavior and the target extractor is generated based on those examples automatically [17, 21, 23, 31, 44, 62, 69, 129, 134]. We propose a methodology for *predicting* the accuracy of the extractor that may be inferred by a given extraction inference engine from the available examples. Our prediction methodology does not depend on the inference engine internals and can in principle be applied to any inference engine: indeed, we validate it on two different engines which infer different forms of extractors.

The basic idea is to use string similarity metrics to characterize the examples. In this respect, an "easy" problem instance is one in which (i) strings to be extracted are "similar" to each other, (ii) strings not to be extracted are "similar" to each other, and (iii) strings to be extracted are not "similar" to strings not to be extracted. Despite its apparent simplicity, implementing this idea is highly challenging for several

reasons.

To be practically useful, a prediction methodology shall satisfy these requirements: (a) the prediction must be reliable; (b) it must be computed without actually generating the extractor; (c) it must be computed very quickly w.r.t. the time taken for inferring the extractor. First and foremost, predicting the performance of a solution without actually generating the solution is clearly very difficult (see also the related work section).

Second, it is not clear to which degree a string similarity metric can capture the actual difficulty in inferring an extractor for a given problem instance. Consider, for instance, the Levenshtein distance (string edit distance) applied to a problem instance in which entities to be extracted are dates. Two dates (e.g., `2-3-1979` and `7-2-2011`, whose edit distance is 6) could be as distant as a date and a snippet not to be extracted (e.g, `2-3-1979` and `19.79$`, whose edit distance is 6 too); yet dates could be extracted by an extractor in the form of regular expression that is very compact, does not extract any of the other snippets and could be very easy to generate (`\d+-\d+-\d+`). However, many string similarity metrics exist and their effectiveness is tightly dependent on the specific application [58, 65]. Indeed, one of the contributions of our proposal is precisely to investigate which metric is the most suitable for assessing the difficulty of extractor inference.

Third, the number of snippets in an input text grows quadratically with the text size and becomes huge very quickly—e.g., a text composed of just 10^5 characters includes $\approx 10^{10}$ snippets. It follows that computing forms of similarity between all pairs of snippets may be feasible for snippets that are to be extracted but is not practically feasible for snippets that are *not* to be extracted.

We propose several prediction techniques and analyze experimentally our proposals in great depth, with reference to a number of different similarity metrics and of challenging problem instances. We validate our techniques with respect to a state-of-the-art extractor generator[1] approach that we have recently proposed [17, 21, 23]; we further validate our predictor on a worse-performing alternative extractor generator [140] which works internally in a different way. The results are highly encouraging suggesting that reliable predictions for tasks of practical complexity may indeed be obtained quickly.

5.2 Related work

Although we are not aware of any work specifically devoted to predicting the effectiveness of a pattern-based entity extractor inference method, there are several research fields that addressed similar issues. The underlying common motivation is twofold: inferring a solution to a given problem instance may be a lengthy procedure; and, the inference procedure is based on heuristics that cannot provide any optimality guarantees. Consequently, lightweight methods for estimating the quality of a solution before actually generating that solution are highly desirable.

In *combinatorial optimization* a wealth of research efforts have been devoted to the problem of estimating the difficulty of a given problem instance [192]. Such efforts may be broadly categorized in two classes: identifying features of a problem instance which may impact difficulty in terms of quality of a solution; and, identifying problem instance-independent features that may help in characterizing the difficulty of a task in general.

The work in [165] considers a specific class of combinatorial optimization tasks (TSP: travelling salesman problem) and follows a different line of research aimed at identifying features of a problem instance that may be helpful in choosing from a portfolio of algorithms the best one for that instance. The cited work actually considers only two such algorithms and assesses the ability of several classifiers, trained on a number of problem instances, to predict the relative performance of these two algorithms.

A recent proposal in this area followed a common approach consisting in the generation of a number of problem instances for a specific problem class (TSP) by means of a parametrized generation method [135]. A regressor for the solutions was then generated by using features of each problem instance that included

[1] A web based version is available on http://regex.inginf.units.it/; the source code is published on https://github.com/MaLeLabTs/RegexGenerator.

values for the generation parameter. Our approach is similar except that we address a radically different task, thereby calling for radically different features.

An indirect but strong indication that the problem that we are addressing is amenable only to heuristic solutions without any formal guarantee is provided in [106], which considers optimization problems and proves that predictive measures that run in polynomial time do not exist.

Problem difficulty prediction is a very important research topic in *evolutionary computation*: an excellent survey can be found in [96]. The cited work presents a general method for estimating performance of evolutionary program induction algorithms with an experimental evaluation on two important classes of tasks, i.e., symbolic regression and Boolean function induction. The method is based on regressors trained on features extracted from problem instances. Features are defined over forms of distances computed over input-output pairs of the problem instance. We are not aware of any instantiation of this method for application domains involving string similarity computations, where there are many metrics that can be used and their effectiveness is tightly dependent on the specific task (e.g., [130, 205]).

A systematic analysis of a number of measures aimed at characterizing the difficulty of a *classification* problem is presented in [110]. In principle, this analysis could be applied also to text extraction problems, because such problems require classifying each individual character in a stream depending on whether the character is to be extracted. On the other hand, the cited work focuses on the geometrical properties of classification, considering measures that may highlight the separation between classes in the measurement space. Text extraction problems are generally not suitable to interpretations of this kind.

Performance prediction is an important research topic in *information retrieval*, aimed at assessing effectiveness of a query before or during early stages of retrieval [94, 137, 172, 176]. Methods in this area generally require an indexing phase of the document corpus and then emit a prediction for a query based on a quick comparison between query terms and various indexed structures [104] (a corpus-independent approach is proposed in [116]).

As mentioned above, the effectiveness of a given string similarity metrics is usually highly dependent on the specific class of task. For this reason, we apply our proposal on a number of different metrics following an approach taken in other application domains. Several preprocessing strategies in combination with a variety of similarity metrics were assessed with reference to *ontology alignment* task [58]. The focus was finding the combination which exhibits best performance on a wide selection of problem instances representative of the ontology alignment task. A number of similarity metrics proposed by various research communities were applied to the task of matching entity names to database records [65]. The focus was finding the metric most suitable to the specific task. The key difference from our approach is that we investigate different string metrics as a tool for predicting the quality of a solution. The solution itself, i.e., the extractor tailored to a specific task instance, is built with a method which does not use string metrics in any way.

The availability of an estimate of costly data elaborations may be desirable also when dealing with data quality. For instance, the authors of [108] propose a method for estimating the number of duplicates in a dataset, before actually applying more complex specific duplicate detection algorithms. As in our case, a key requirement for the practicality of their proposal is that the estimation procedure has to run much faster than the actual algorithm.

5.3 Problem statement and motivations

5.3.1 Pattern-based entity extraction

The application problem consists in extracting entities that follow a syntactic pattern from a potentially large text. Extraction is performed by means of an *extractor* tailored to the specific pattern of interest. We consider a scenario in which the extractor is generated automatically by an *extraction inference engine*, based on examples of the desired behavior in the form of snippets to be extracted (i.e., the entities) and of snippets not to be extracted. Such examples usually consist of user-provided annotations on the text to be processed by the extractor.

A *snippet* X of a string s is a substring of s, identified by the starting and ending indexes in s. We denote by $\mathcal{X}$ the set of all snippets of string s. An *example* (s, X) is a string s associated with a (possibly empty) set of non-overlapping snippets $X \subset \mathcal{X}$. We do not make any assumption on the length or internal structure of string s, which may be, e.g., a text line, or an email message, or a collection of email messages, or a log file and so on. Set X represents all and only the *desired extractions* from s, i.e., snippets in $\mathcal{X} \backslash X$ are *not* to be extracted.

The extractor inference engine takes an example (s, X) as input and outputs an extractor e whose extraction behavior is consistent with the provided example—e should extract from s only the desired extractions X. Furthermore, e should capture the *pattern* describing the extractions, thereby *generalizing* beyond the actual examples. In other words, (s, X) constitutes an incomplete specification of the extraction behavior of an ideal and unknown extractor $e^\star$ and the extractor inference engine should aim at inferring an extractor with the same extraction behavior of $e^\star$.

To quantify the quality of a solution e, another example (s', X') is used such that both (s, X) and (s', X') represent the extraction behavior of $e^\star$. The extraction behavior of e is compared against that of $e^\star$ in terms of its F-measure (harmonic mean of precision and recall) on (s', X'): F-measure is 1 if and only if e extracts all and only the snippets X' from s'. We emphasize that (s, X) is the input of the extraction inference engine, that is, (s', X') is *not* required for actually generating e. We use (s', X') only for assessing the quality of a generated extractor.

Let $(s, X), (s', X')$ be a pair representing the extraction behavior of the target unknown extractor $e^\star$. We define the tuple (s, X, s') to be a *problem instance*. Let f' be the F-measure on (s', X') of the extractor e generated from (s, X) by the extraction inference engine. We define the tuple (s, X, s', X', f') to be a *solved problem instance*.

5.3.2 Effectiveness prediction

With reference to a pair $(s, X), (s', X')$ representing the extraction behavior of the target unknown extractor $e^\star$, let e denote the extractor generated by the extractor inference engine. The prediction problem consists in predicting F-measure f' of e on (s', X') based solely on (s, X, s')—that is, neither e nor X' are available for constructing the prediction. Prediction reliability may be measured in terms of the prediction error on a set of solved problem instances: we present the specific indexes that we used to this purpose in Section 5.5.2. In general, we are interested in minimizing the prediction error, i.e., the difference between the predicted value $\hat{f}'$ and the actual value f', which, clearly, is not available while computing the prediction.

This problem statement models the practical scenario in which the user has annotated some text for generating the extractor e and is interested in assessing the quality that will be obtained by applying e on a given unannotated text, *before* actually generating e.

In order to train the predictor, we assume that a set of solved problem instances are available. Note that knowledge of e for solved problem instances is not required.

Our overall framework do not make any assumptions on the implementation of the extractor inference engine or on the nature of the text extractor itself. However, our interest in this problem as well as the detailed experimental assessment in this work are based on evolutionary generation of extractors consisting of *regular expressions*. In particular, we will consider solved problem instances obtained from the extraction inference engine in [17]. However, we further validate our framework by applying it also on a different extraction engine which generates extractors in the form of Deterministic Finite Automata (DFA) [140].

5.4 Our prediction method

Our prediction method consists of three steps. First, we transform the input (s, X, s') in an intermediate representation which is suitable to be processed using string similarities. Second, we extract a set of numerical features consisting in several statistics of similarities among strings of the intermediate

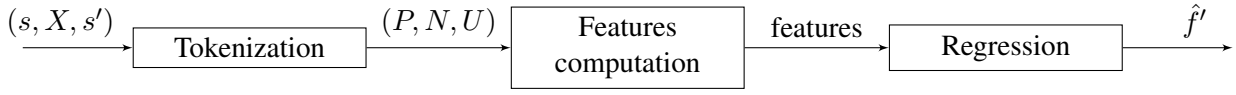


Figure 5.1: An overview of our prediction method. The input is a problem instance and the output is an estimate $\hat{f}'$ of the F-measure on (s', X').

$$
\begin{aligned}
s &= \text{The␣file␣has␣been␣sent␣on␣11.10.2013␣and␣has␣been␣received} \\
&\quad \text{␣on␣15-10-2013;␣the␣content␣has␣been␣written␣on␣7/2/2011.} \\
X &= \{\text{11.10.2013}, \text{15-10-2013}, \text{7-2-2011}\} \\
s' &= \text{Today␣is␣18-12-2015;␣nice!}
\end{aligned}
$$

(a) Problem instance (s, X, s').

$$
\begin{aligned}
\mathcal{S}_0 &= \{\text{␣}, ;, .\} & & \\
\mathcal{S}_1 &= \{\text{␣}\} & \Rightarrow |T_1 \cap X| &= 1 \\
\mathcal{S} = \mathcal{S}_2 &= \{\text{␣}, ;\} & \Rightarrow |T_2 \cap X| &= 2 \\
\mathcal{S}_3 &= \{\text{␣}, ;, .\} & \Rightarrow |T_3 \cap X| &= 2
\end{aligned}
$$

(b) Choice of the separators set $\mathcal{S}$.

$$
\begin{aligned}
P &= \{\text{11.10.2013}, \text{15-10-2013}, \text{7-2-2011}\} \\
N &= \{\text{The}, \text{file}, \text{has}, \ldots, \text{on}\} \\
U &= \{\text{Today}, \text{is}, \text{18-12-2015}, \text{nice!}\}
\end{aligned}
$$

(c) Tokenization outcome (P, N, U).

Figure 5.2: Tokenization example.

representation. Finally, we apply a regressor to the vector of features and obtain an estimate $\hat{f}'$ of the F-measure f' which an extractor would have on X'.

In the following sections, we describe each step in detail. Figure 5.1 shows an overview of the prediction method.

5.4.1 Tokenization

We transform the input (s, X, s') into a triplet (P, N, U) whose elements are multisets of strings among which similarities can be computed. Multiset P (*positives*) includes all the snippets in X (i.e., all the desired extractions). Multisets N (*negatives*) and U (*unlabeled*) are obtained after splitting s and s' in *tokens*, according to the tokenization procedure described below. In particular, N includes all the tokens of s which do not overlaps snippets in X while U includes all the tokens of s'.

The aim of tokenization is to split strings in tokens whose length and content is "appropriate" with respect to the specific problem instance. To this end, we construct a set $\mathcal{S}$ of characters acting as token separators as follows. First, we construct the set of characters $\mathcal{S}_0$ including each character immediately preceding or immediately following each snippet in X. Second, we sort $\mathcal{S}_0$ in descending order according to the number of occurrences of each character. Third, we iterate the following steps starting from $i = 1$: (i) we construct the set $\mathcal{S}_i$ of the first i characters of $\mathcal{S}_0$, and (ii) we build the set $T_i \subset \mathcal{X}$ of tokens obtained by splitting s with the separators in $\mathcal{S}_i$. Finally, we assign $\mathcal{S}$ to the set $\mathcal{S}_i$ for which the number of tokens which are snippets to be extracted is maximal, i.e., $\mathcal{S} := \mathcal{S}_{i*}$, with $i^* = \arg\max_i |T_i \cap X|$—in case of tie, we choose the set $\mathcal{S}_i$ with smallest size.

Having determined the set of characters $\mathcal{S}$ acting as token separators, we split s and s' in tokens accordingly. Figure 5.2 shows an example of the tokenization procedure applied to a problem instance concerning the extraction of dates. In particular, Figure 5.2b shows the iterative procedure used to build the set of separators $\mathcal{S}$: in this case, $\mathcal{S}$ is assigned to $\mathcal{S}_2 = \{\text{␣}, ;\}$; the dot character is not considered as as a separator because in that case the snippet 11.10.2013 would be split.

5.4.2 Features computation

Given a triplet (P, N, U) and a string similarity metric m (see next section), we want to obtain a set of numerical features which are relevant to characterize the problem instance difficulty, and hence affect the

effectiveness of the extractor inference on that instance. As outlined in the introduction, the basic idea consists in computing some statistics from similarity measurements able to capture aggregate differences between strings to be extracted and strings not to be extracted. In particular, we should compute the similarity among the strings in P, N, and U; next across strings in P and N as well in P and U; finally, we could compute some statistics among all these computations. However, the size of the involved multisets (N and U in particular) would make an approach of this sort not feasible. Hence, we propose two different methods to drastically reduce the amount of similarity computations.

In the first method, which we call *Sample*, we proceed as follows. We construct a subset $N' \subset N$ by randomly sampling $|P|$ elements from N and a subset $U' \subset U$ by randomly sampling $|P|$ elements from U. Then, we compute the similarity values for all pairs of strings in $P \times P$, $P \times N'$, $P \times U'$, $N' \times N'$, and $U' \times U'$ and compute 6 statistics for each of the 5 sets of measurements: min, max, mean, median, 25th-percentile, and 75th-percentile—max is not taken into account for pairs of strings of the same set ($P \times P$, $N' \times N'$ and $U' \times U'$). We predict f' by using as features $|P|$, $|N|$, $|U|$ and all the previously computed figures—$3 + 6 \cdot 5 - 3 = 30$ features.

In the second method, which we call *Rep*, we build a set P' containing 3 representatives of the positives in P, as follows. We compute, for each $p \in P$, its average similarity to all the other positives (i.e., $\bar{m}(p) := \frac{1}{|P|} \sum_{p' \in P} m(p, p')$); then, we insert in P': (1) the positive with the lowest $\bar{m}$; (2) the positive with the greatest $\bar{m}$; and, (3) the positive with the $\bar{m}$ closest to $\frac{1}{|P|} \sum_{p \in P} \bar{m}(p)$ (i.e., the average value for the average similarity). Finally, we compute the similarity values for all pairs of strings in $P' \times P'$, $P' \times N$, $P' \times U$ and the same 6 statistics as above—max is not taken into account for $P' \times P'$. We predict f' by using as features $|P|$, $|N|$, $|U|$ and all the previously computed figures—$3 + 6 \cdot 3 - 1 = 20$ features.

String similarity metrics

Several different string similarity metrics exist. In order to limit the number of possible prediction method variants by considering only the more promising metrics, we referred to previous studies which compared string metrics [39, 58, 63] and we chose the most promising ones: Jaccard, Jaro, JaroWinkler, Levenshtein, NeedlemanWunsch, SmithWaterman, SoftTFIDF, UnsmoothedJS. We used the SecondString Java library [64] to actually implement the metric computation. We provide below a high-level overview of each metric and refer the reader to [65] and to the documentation of the SecondString library for full details.

The Jaccard similarity index between two strings a and b is computed by considering a string as a set of characters or bigrams (as in our case) and is the ratio between the intersection and the union of the two sets a, b. The Jaro similarity index takes into account *matching* characters, i.e., characters appearing in both a and b at an offset smaller than a certain quantity, and *transpositions*, i.e., number of matching characters appearing in a different order in the two strings. The JaroWinkler index is a modified Jaro index in which similarity grows for strings that share a common prefix. Levenshtein takes into account the minimum number of single-character edits required to make a and b identical, i.e., it is a form of edit distance. NeedlemanWunsch is a form of edit distance which assigns different costs to edit operations (we used the standard cost configuration of the SecondString library). SmithWaterman is a variant of NeedlemanWunsch assigning lower costs to sequences of insertions or deletions. SoftTFIDF is a form of *cosine similarity* weighing substrings appearing in both a and b and substrings of either string for which a substring of the other exists that is sufficiently similar according to the JaroWinkler metric. Finally, UnsmoothedJS (Jensen-Shannon) is a similarity index taking into account the loss of information when representing either string with the other, where the loss of information is quantified by the Kullback-Leibler divergence.

5.4.3 Regression

We explored three different options: a linear model (LM), random forests (RF) regression, and support vector machines (SVM) regression. In particular, we used the gaussian kernel and $C = 1$ for SVM [56], and we used the algorithm proposed in [47] with $n_{\text{tree}} = 500$ for RF. In all cases we set the actual predicted

value to 1 if the model output is larger than 1 and to 0 if it is lower than 0—f' values are intrinsically defined in $[0, 1]$.

5.5 Experimental evaluation

We constructed and assessed experimentally all the 48 prediction model variants resulting from the combination of: 2 feature set construction methods (Sample and Rep, Section 10.3.1); 8 string similarity metrics (Section 5.4.2); 3 regressors (LM, RF, and SVM, Section 5.4.3). We trained each model variant with a set of solved problem instances $\mathcal{E}_{\text{train}}$ and assessed the resulting predictor on a set of solved problem instances disjoint from $\mathcal{E}_{\text{train}}$, as detailed in the next sections.

5.5.1 Data

We consider 19 challenging *extraction tasks* built over a text corpus fully annotated with the entities to be extracted. We use a selection of the tasks used in [17], summarized in Table 5.1. The name of each extraction task is composed of the name of the corpus followed by the name of the entity type to be extracted. Entity names should be self-explanatory: Username corresponds to extracting only the username from Twitter citations (e.g., only MaleLabTs instead of @MaleLabTs); Email-ForTo corresponds to extracting email addresses appearing after the strings for: or to: (possibly capitalized). Names ending with a $*$ suffix indicate extraction tasks with *context*. These are extraction tasks in which a text snippet must or must not be extracted depending on the text surrounding the snippet—e.g., an email address might have to be extracted only when following a Reply-To: header name.

Each extraction task consists of a string s_0 annotated with all the desired extractions. Table 5.1 shows, for each extraction task, the length $\ell(s_0)$ of the string s_0 (in thousands of characters) and the number $|X_0|$ of snippets corresponding to entities to be extracted. The construction of problem instances from extraction tasks is described in the next section. The table shows also the average F-measure obtained by the approach of [17] on those tasks see next section for more details) and the average time taken to learn the extractor.

5.5.2 Experimental procedure

We aimed at investigating the prediction effectiveness at varying difficulty of extraction, in particular concerning: (a) the amount of data available for learning; (b) the complexity of the pattern of the involved entity; and (c) the degree of representativeness of the learning data w.r.t. all the other data. To this end, we built a number of different solved problem instances (s, X, s', X', f') from our 19 extraction tasks, as follows.

Concerning the amount of data available for learning, we considered three values for the number n_X of snippets to be extracted, $\{25, 50, 100\}$. Then, for each extraction task and each n_X value, we built 5 solved problem instances (s, X, s', X', f') (*repetitions*): (i) we randomly chose a substring s of s_0 containing n_X snippets to be extracted; (ii) we generated an extractor from (s, X) with the method in [17]; (iii) we randomly chose a substring s' of s_0 non-overlapping s and such that $|X'| = 500$; (iv) we assessed the f-measure f' of the generated extractor on (s', X'). Concerning step ii, we used the tool described in chapter 2, in [17, 23], and available at https://github.com/MaLeLabTs/RegexGenerator: the tool generates a regular expression which aims at extracting from s all and only the snippets in X while trying to generalize the learning data. The regular expression is generated by means of an evolutionary procedure which searches the space of the regular expressions driven by a multiobjective optimization strategy—we refer the reader to chapter 2 and the cited paper for full details. Table 5.1 shows the average value of f'—i.e., the F-measure on (s', X') obtained by the regular expression generated by the cited tool from the examples in (s, X)—over the 5 repetitions for each task and each value of n_X.

Next, we used the 285 solved problem instances for assessing our 48 model variants. In particular, we executed a 5-fold cross validation of the behavior of each model variant for each pair extraction task and n_X value. That is, we trained each model variant on 4 of the 5 repetitions and assessed the prediction

Table 5.1: Salient information about the 19 extraction tasks. The length $\ell(s_0)$ of the string s_0 is expressed in thousands of characters. The four rightmost columns show average values for f' for different values of the number of snippets to be extracted n_X (columns 4–6) and the average time t_l (in min) taken by the inference engine to learn an extractor for $n_X = 50$.

			f'			
Extraction task	$\ell(s_0)$	$\lvert X_0 \rvert$	25	50	100	t_l
BibTeX/Author*	54	589	0.80	0.85	0.85	20
BibTeX/Title*	54	200	0.69	0.70	0.73	141
Cetinkaya-HTML/href	154	214	0.98	0.98	0.99	26
Cetinkaya-HTML/href-Content*	154	214	0.74	0.73	0.80	29
Cetinkaya-Text/All-URL	39	168	0.99	0.99	0.99	8
CongressBills/Date	16 511	3085	0.42	0.63	0.64	584
Email-Headers/Email	261	1244	0.64	0.64	0.73	224
Email-Headers/Email-To-For*	261	331	0.61	0.56	0.78	398
Email-Headers/IP	261	848	0.86	0.86	0.91	89
Log/IP	4126	75 958	1.00	1.00	1.00	2
Log/MAC	4126	38 812	1.00	1.00	1.00	3
NoProfit-HTML/Email	860	1094	0.86	1.00	1.00	3
Reference/First-Author*	30	198	0.97	0.97	1.00	26
ReLIE-Email/Phone-Number	8805	5184	0.79	0.80	0.78	16
ReLIE-HTML/All-URL	4240	502	0.88	0.92	0.95	35
ReLIE-HTML/HTTP-URL	4240	499	0.88	0.91	0.92	32
Twitter/All-URL	4344	14 628	0.98	0.98	0.98	8
Twitter/Hashtag+Citation	4344	56 994	1.00	1.00	1.00	4
Twitter/Username*	4344	42 352	1.00	1.00	1.00	2

on the remaining one. The rationale for partitioning the dataset of solved problem instances based on repetitions is the need of ensuring the presence in the training data of at least one problem instance for each extraction task. Such a partitioning corresponds to the scenario in which the data available for calibrating the prediction method are representative of a wide range of different extraction tasks. That scenario could be implemented in a real setting easily, as it would suffice to re-train the predictor after each new run of the engine. We remark, however, that *none* of the problem instances used for assessing the prediction method was available in the training phase. Later, in Section 5.5.3, we analyze the much more challenging scenario in which a novel extraction task arises.

For assessing the predictor, we computed the following indexes:

- Mean absolute error (MAE), which measures the average value of the absolute difference between the predicted value $\hat{f}'$ and the actual value f':

$$\text{MAE} = \frac{1}{|\mathcal{E}|} \sum_{\mathcal{E}} \left| \hat{f}' - f' \right|$$

  where $\mathcal{E}$ is the set of solved problem instances on which the predictor is assessed.

- Relative mean absolute error (RMAE), which is the average value of the ratio between the absolute error and the actual value:

$$\text{RMAE} = \frac{1}{|\mathcal{E}|} \sum_{\mathcal{E}} \left| \frac{\hat{f}' - f'}{f'} \right|$$

- Performance-class accuracy (CA), which measures accuracy on a classification task in which problem instances are partitioned in 3 difficulty classes, easy, medium, and hard, corresponding

to values of f' in the intervals $]0.95, 1]$, $]0.8, 0.95]$, and $[0, 0.8]$, respectively. In our setting, this partitioning corresponds to roughly 32%, 29%, and 39% problem instances in the respective performance classes.

$$\text{CA} = \frac{\left|\left\{C(f') = C(\hat{f}')\right\}\right|}{|\mathcal{E}|}$$

where $C : [0, 1] \to \{]0.95, 1],]0.8, 0.95], [0, 0.8]\}$ is the function which assigns the performance-class to a given value of F-measure. In other words, CA is the ratio between the number of instances for which the predicted value $\hat{f}'$ and the actual value f' belong to the same performance-class and the number of all instances.

MAE and RMAE are indexes commonly used for assessing predictors of continuous values; we defined the latter index (CA) in order to capture the ability of the proposed predictor in providing a coarser indication of the difficulty of a problem instance.

5.5.3 Results and discussion

Table 5.2 shows the values of MAE, RMAE, and CA for all the 48 model variants, averaged over all prediction problems. The table provides the values of the three indexes computed on the problem instances which has not been used for training the model (denoted by $\mathcal{E}_{\text{validate}}$); for completeness of analysis, the table also shows indexes values on the solved problem instances used for training the model (denoted by $\mathcal{E}_{\text{train}}$).

Since we could not identify any baseline method from the literature, we chose to use the following prediction as baseline method. Concerning f', the predicted value is the average value across all the solved problem instances in $\mathcal{E}_{\text{train}}$; concerning the performance-class, the predicted value is the most occurring class in $\mathcal{E}_{\text{train}}$.

The table shows also the average time t_t for training the predictor on $\mathcal{E}_{\text{train}}$ and the average time t_p for computing features for a single problem instance, both expressed in ms. All the experiments have been carried out on a workstation equipped with $8\,$GB and a Intel Core2 Quad CPU $2.5\,$GHz. It can be seen that a prediction can be obtained, in many cases, in less than a second: for reference, the inference of a regular expression by means of the method of chapter 2, as seen in [17], for the same tasks requires much longer times—ranging from $\approx 2\,$min to $\approx 500\,$min (see rightmost column of Table 5.1).

By analyzing the experimental results of Table 5.2, several considerations can be made.

It can be seen that the best prediction is quite good: RMAE $= 9.1\%$ for the RF-Sample-NeedlemanWunsch combination. This value corresponds to an average absolute error MAE $= 0.056$. Moreover, RMAE $\leqslant 10\%$ for 6 combinations, all based on RF and evenly distributed between Sample and Rep. To place this figure in perspective, we observe that the RMAE for the baseline predictor obtained 19.8%—that is, our prediction methodology halves the error w.r.t. the baseline. In order to investigate if the effectiveness is limited to the specific inference engine here considered, we also applied it to a different engine. The detailed results are presented at the end of this section: in brief, they confirm that that our methodology outperforms the baseline.

Quality of the prediction is good also when assessed in terms of accuracy of the performance-class prediction (CA): 80.9% for the RF-Sample-NeedlemanWunsch combination. Moreover, CA $\geqslant 75\%$ for 12 combinations, mostly based on RF-Sample. To place this figure in perspective, we observe that the CA for baseline class predictor is 33.8%.

Prediction based on the Sample method for feature construction tends to be more effective than prediction based on the Rep method. We speculate that choosing a few representative positives may succeed to capture diversity between positives, but fails at capturing the overall variability of the text which has to be processed, which is what actually matter in making a problem more or less hard to solve. It can also be observed that computing features takes in general longer for Rep.

With respect to the prediction model, RF appears to be the best performing choice and obtains in general better figures for all the indexes. Moreover, it appears to be more robust to the other factors involved—metric and feature computation method.

Table 5.2: Results with the 5-fold cross validation on repetitions.

		f' on $\mathcal{E}_{\text{train}}$			f' on $\mathcal{E}_{\text{validate}}$			t_t	t_p
	Metric m	MAE	RMAE	CA	MAE	RMAE	CA	[ms]	[ms]
RF-Sample	Jaccard	0.026	4.2	89.5	0.062	10.0	78.0	1053	585
	Jaro	0.026	4.3	89.0	0.065	10.5	74.9	856	49
	JaroWinkler	0.027	4.4	87.8	0.066	10.5	74.2	854	49
	Levenstein	0.024	4.0	90.1	0.059	9.6	79.8	999	492
	NeedlemanWunsch	0.023	3.8	89.8	0.056	9.1	80.9	985	487
	SmithWaterman	0.025	4.0	87.8	0.058	9.4	78.8	838	559
	SoftTFIDF	0.029	4.6	88.6	0.066	10.6	76.0	724	227
	UnsmoothedJS	0.029	4.7	87.3	0.065	10.5	72.8	729	88
RF-Rep	Jaccard	0.027	4.4	89.1	0.064	10.3	77.4	742	6072
	Jaro	0.027	4.4	87.4	0.064	10.3	73.9	650	324
	JaroWinkler	0.028	4.5	86.1	0.064	10.2	76.0	643	320
	Levenstein	0.026	4.2	88.6	0.061	9.8	74.6	699	2245
	NeedlemanWunsch	0.026	4.2	89.0	0.062	9.9	75.6	711	2247
	SmithWaterman	0.026	4.3	87.8	0.058	9.5	78.8	636	2329
	SoftTFIDF	0.032	5.1	87.5	0.070	11.1	69.3	570	1425
	UnsmoothedJS	0.034	5.4	85.6	0.073	11.4	66.4	561	862
SVM-Sample	Jaccard	0.050	8.5	85.2	0.064	10.6	75.9	59	585
	Jaro	0.089	11.0	9.2	0.144	20.0	21.6	46	49
	JaroWinkler	0.089	11.0	9.2	0.144	20.0	21.6	46	49
	Levenstein	0.053	8.9	79.1	0.071	11.6	72.8	59	492
	NeedlemanWunsch	0.050	8.4	83.0	0.066	10.7	78.1	56	487
	SmithWaterman	0.089	10.8	8.5	0.144	20.1	21.2	51	559
	SoftTFIDF	0.089	11.0	9.2	0.144	20.0	21.6	43	227
	UnsmoothedJS	0.089	11.0	9.2	0.144	20.0	21.6	45	88
SVM-Rep	Jaccard	0.052	8.9	80.7	0.065	10.7	73.9	49	6072
	Jaro	0.089	11.0	8.8	0.144	20.0	21.6	37	324
	JaroWinkler	0.089	11.0	8.8	0.144	20.0	21.6	40	320
	Levenstein	0.058	9.9	77.2	0.066	11.1	71.4	46	2245
	NeedlemanWunsch	0.058	9.9	77.2	0.066	11.1	71.4	49	2247
	SmithWaterman	0.089	10.8	9.1	0.143	20.0	21.2	44	2329
	SoftTFIDF	0.089	11.0	8.8	0.144	20.0	21.6	36	1425
	UnsmoothedJS	0.089	11.0	8.9	0.144	20.0	21.6	36	862
LM-Sample	Jaccard	0.075	11.1	66.0	0.085	12.7	62.1	17	585
	Jaro	0.081	11.9	53.0	0.089	13.1	50.2	15	49
	JaroWinkler	0.087	12.7	50.2	0.094	13.7	44.5	15	49
	Levenstein	0.082	12.0	57.3	0.096	14.2	54.4	16	492
	NeedlemanWunsch	0.073	10.5	58.0	0.091	12.9	52.7	17	487
	SmithWaterman	0.082	11.8	51.4	0.091	13.3	53.8	15	559
	SoftTFIDF	0.082	11.8	52.7	0.088	12.9	48.8	15	227
	UnsmoothedJS	0.084	12.2	51.9	0.089	13.2	49.1	14	88
LM-Rep	Jaccard	0.079	11.8	59.7	0.083	12.5	57.2	11	6072
	Jaro	0.087	12.7	51.0	0.091	13.3	50.5	12	324
	JaroWinkler	0.084	12.4	55.5	0.091	13.3	51.6	11	320
	Levenstein	0.083	12.0	56.4	0.096	13.7	53.4	10	2245
	NeedlemanWunsch	0.083	12.0	56.4	0.096	13.7	53.4	10	2247
	SmithWaterman	0.087	12.7	53.6	0.094	13.8	52.3	11	2329
	SoftTFIDF	0.093	13.6	47.5	0.096	14.2	45.9	11	1425
	UnsmoothedJS	0.090	13.2	49.6	0.093	13.7	47.0	11	862
Baseline		0.135	19.8	53.0	0.135	19.8	53.0		

Concerning the string similarity metrics our experiments suggest that NeedlemanWunsch is the best one. Good results can be obtained with Levenshtein and SmithWaterman metrics as well, however: this finding is not surprising because these 3 metrics are closely related (see Section 5.4.2).

Depending on the specific scenario in which a prediction should be provided, efficiency could play a role almost as important as effectiveness: a faster and slightly less accurate prediction may be more useful than a longer and more accurate one. In this sense, we observe that RF-Sample-Jaro allows computing features in just a tenth of the time required by RF-Sample-NeedlemanWunsch, with a moderate decrease in CA: 74.9%, roughly -6% less than the best combination. In fact, the time for obtaining a prediction mostly consists in the time taken to compute the features, since the regressor application time is negligible—t_t is the time taken to *train* rather than to apply the regressor.

Table 5.3 shows the results of the best performing method (i.e., RF-Sample-NeedlemanWunsch) for each considered extraction task: besides MAE, RMAE, and CA, the table also shows the mean and standard deviation of the actual (f') and predicted ($\hat{f}'$) values of the F-measure obtained for each task. It can be seen that for the majority of the tasks, the prediction is indeed accurate, being RMAE $\leqslant 5\%$. On the other hand, there are 3 on 19 tasks for which RMAE is greater than 20%: two of them, including the one for which the predictor is less effective, are tasks with context (see Section 5.5.1). Indeed, our prediction methodology bases only on the similarities between snippets to be extracted and snippets not to be extracted—the context around snippets is not taken into account. In this respect, we manually inspected several snippets in the tasks Email-Headers/Email-To-For* and Cetinkaya-HTML/href-Content* and found that (i) snippets are in general longer than other tasks and (ii) some snippets to be extracted are similar to some snippets not to be extracted (e.g., the same email address can be present in both categories in Email-Headers/Email-To-For*): these factors can indeed make the string similarity-based prediction harder.

Concerning the performance-class accuracy CA, it can be observed that, in addition to the two tasks mentioned above, the figure is unsatisfactory also for ReLIE-Email/Phone-Number (46.7%), for which, instead, RMAE is lower than the average. We believe that this result is an artifact deriving from the choice of the performance-class intervals and the specific f' values observed for that task—they lie right around 0.8, which is the bound between two performance classes (see Table 5.1).

In order to investigate on which features are most relevant for the prediction, we performed a feature ablation procedure. We focused on the RF-Sample-NeedlemanWunsch variant (i.e., the best one) and repeated the experiment several times by removing one by one each feature. Table 5.4 shows the results (MAE, RMAE, and CA) for each removed feature: the rows are sorted according to decreasing RMAE—i.e., the feature whose removal causes the greatest drop in RMAE comes first. Despite the figures do not allow to draw sharp conclusions, it can be seen that $|U|$ and $|P|$ appear to play an important role in the prediction, followed by the median and mean of similarities among negatives and among positives, respectively. From another point of view, all the statistics about similarities appear to capture the overall difficulty of the extraction task. Moreover, we interpret the importance of $|U|$ and $|P|$ as a measure of how "big" is the unknown data w.r.t. the data which was available for learning—a respect which impacts the *representativeness* of learning data for the chosen task, which is a crucial factor in all machine learning applications.

In order to gain further insights on our proposal, we performed two additional experimental campaigns aimed at (i) assessing the method in the challenging scenario where a novel extraction task arises; and, (ii) validating the method when applied to a different extractor inference technique.

Concerning the former aim, we considered the best variant (RF-Sample-NeedlemanWunsch) and modified the experimental procedure (Section 5.5.2) as follows: instead of executing a 5-fold cross validation on the repetition number, we executed a 19-fold cross validation on the extraction task. That is, for each extraction task, we trained our prediction model on the 270 solved problem instances of the other tasks and assessed the resulting predictor on the 15 instances of that task. Table 5.5 presents the results with the same structure of Table 5.3. It can be seen that, for the majority of the tasks, the RMAE still remains lower than 10%; moreover, for 6 tasks, the performance-class accuracy is larger than 66%. On average, as expected, the prediction is less reliable than in the scenario where the predictor is trained on

Table 5.3: Results of RF-Sample-NeedlemanWunsch for f' on $\mathcal{E}_{\text{validate}}$ for different tasks.

				f'		$\hat{f}'$	
Extraction task	MAE	RMAE	CA	avg	sd	avg	sd
BibTeX/Author*	0.060	7.4	73.3	0.837	0.026	0.829	0.021
BibTeX/Title*	0.079	12.2	86.7	0.705	0.095	0.716	0.014
Cetinkaya-HTML/href	0.022	2.3	86.7	0.984	0.007	0.964	0.010
Cetinkaya-HTML/href-Cont.*	0.230	37.9	40.0	0.757	0.189	0.759	0.035
Cetinkaya-Text/All-URL	0.012	1.2	100.0	0.991	0.006	0.981	0.003
CongressBills/Date	0.082	18.6	100.0	0.563	0.053	0.594	0.014
Email-Headers/Email	0.091	17.6	93.3	0.671	0.077	0.738	0.019
Email-Headers/Email-To-For*	0.113	23.9	86.7	0.649	0.068	0.677	0.041
Email-Headers/IP	0.066	8.1	86.7	0.878	0.050	0.880	0.016
Log/IP	0.014	1.4	100.0	1.000	0.000	0.987	0.005
Log/MAC	0.005	0.5	100.0	0.999	0.001	0.996	0.002
NoProfit-HTML/Email	0.104	20.9	33.3	0.952	0.107	0.924	0.038
References/First-Author*	0.016	1.6	100.0	0.978	0.009	0.979	0.003
ReLIE-Email/Phone-Number	0.055	6.9	46.7	0.788	0.051	0.786	0.005
ReLIE-HTML/All-URL	0.029	3.1	60.0	0.917	0.031	0.898	0.027
ReLIE-HTML/HTTP-URL	0.043	4.9	80.0	0.901	0.042	0.903	0.020
Twitter/All-URL	0.007	0.7	93.3	0.981	0.003	0.975	0.004
Twitter/Hashtag+Citation	0.014	1.4	93.3	0.999	0.001	0.984	0.018
Twitter/Username*	0.013	1.3	100.0	1.000	0.000	0.985	0.017
Average	0.056	9.1	80.9	0.871	0.043	0.871	0.016

problem instances from all the extraction tasks: MAE and RMAE roughly double, whereas CA roughly halves.

Finally, we applied our proposal (in the RF-Sample-NeedlemanWunsch variant) also to another inference engine which generates extractors in the form of Deterministic Finite Automata (DFA) [140]. We built the engine by implementing the method described in the cited paper and applied it to a selection of 6 extraction tasks: Cetinkaya-HTML/href, Cetinkaya-Text/All-URL, CongressBills/Date, ReLIE-Email/Phone-Number, ReLIE-HTML/All-URL, and Twitter/Hashtag+Citation—we hence obtained 90 solved problem instances. In this case the inference engine produces DFAs which tend to be much less effective than the extractors considered in the previous experiments, i.e., those generated by the engines in [17] and in chapter 2. We thus redefined the performance classes as $]0.25, 1]$, $]0.15, 0.25]$, and $[0, 0.15]$, which roughly correspond to 20%, 45%, and 35% problem instances, respectively. We executed the same 5-fold cross validation described in Section 5.5.2—i.e., the predictor is trained on 4 repetitions and assessed on the remaining one—and obtained 0.065, 57.2, and 55.9 for MAE, RMAE, and CA, respectively. We verified that our predictor scored better than the baseline ($+10\%$ for CA). On the other hand, the figures are in general worse than those obtained for the inference engine of [17]. We speculate that this difference is mostly motivated by the fact that the effectiveness of [140] is poor in general and appears to be only slightly affected by the specific problem instance.

5.6 Remarks

In this chapter we have considered a scenario in which an extraction inference engine generates an extractor automatically from user-provided examples of the entities to be extracted from a dataset. We have addressed the problem of predicting the accuracy of the extractor that may be inferred from the available examples, by requiring that the prediction be obtained very quickly w.r.t. the time required for actually inferring the extractor. This problem is highly challenging and we are not aware of any earlier

Table 5.4: Results of RF-Sample-NeedlemanWunsch on $\mathcal{E}_{\text{validate}}$ according to the feature ablation procedure.

Removed feature	MAE	RMAE	CA
$\lvert U\rvert$	0.058	9.3	81.6
$\lvert P\rvert$	0.057	9.3	80.9
median$_{N' \times N'}$	0.057	9.2	79.8
mean$_{P \times P}$	0.057	9.2	81.3
min$_{P \times P}$	0.057	9.2	82.0
max$_{P \times N'}$	0.057	9.2	80.9
max$_{P \times U'}$	0.057	9.2	80.2
75th-percentile$_{U' \times U'}$	0.057	9.2	81.2
median$_{P \times U'}$	0.056	9.2	80.2
75th-percentile$_{P \times N'}$	0.057	9.2	80.9
median$_{U' \times U'}$	0.056	9.2	80.9
75th-percentile$_{P \times P}$	0.056	9.2	81.2
$\lvert N\rvert$	0.057	9.2	80.2
25th-percentile$_{P \times U'}$	0.056	9.2	80.9
25th-percentile$_{P \times P}$	0.056	9.2	80.2
25th-percentile$_{N' \times N'}$	0.056	9.2	81.2
25th-percentile$_{P \times N'}$	0.056	9.1	81.3
75th-percentile$_{P \times U'}$	0.056	9.1	80.5
75th-percentile$_{N' \times N'}$	0.056	9.1	80.2
min$_{P \times U'}$	0.056	9.1	80.9
median$_{P \times N'}$	0.056	9.1	80.9
min$_{U' \times U'}$	0.056	9.1	81.2
mean$_{N' \times N'}$	0.056	9.1	80.2
min$_{N' \times N'}$	0.056	9.1	81.3
min$_{P \times N'}$	0.056	9.1	79.8
mean$_{U' \times U'}$	0.056	9.1	81.6
mean$_{P \times N'}$	0.056	9.1	81.2
mean$_{P \times U'}$	0.056	9.0	81.2
25th-percentile$_{U' \times U'}$	0.056	9.0	82.0
median$_{P \times P}$	0.055	9.0	80.9
No removals	0.056	9.1	80.9

proposal in this respect. With reference to extractors consisting of regular expressions, we have proposed several techniques and analyzed them experimentally in depth. The results suggest that reliable predictions for tasks of practical complexity may indeed be obtained quickly and without actually generating the extractor.

Table 5.5: Results, in the novel extraction task scenario, of RF-Sample-NeedlemanWunsch for f' on $\mathcal{E}_{\text{validate}}$ for different tasks.

				f'		$\hat{f}'$	
Extraction task	MAE	RMAE	CA	avg	sd	avg	sd
BibTex-Author*	0.066	8.0	53.3	0.837	0.067	0.811	0.043
BibTex-Title*	0.189	29.8	6.7	0.705	0.115	0.871	0.012
Cetinkaya-HTML/href	0.134	13.6	6.7	0.984	0.020	0.861	0.021
Cetinkaya-HTML/href-Cont.*	0.218	36.6	53.3	0.757	0.253	0.760	0.014
Cetinkaya-Text/All-URL	0.133	13.4	0.0	0.991	0.006	0.866	0.039
CongressBills-Date	0.239	51.1	40.0	0.563	0.122	0.804	0.013
Email-Headers/Email	0.134	25.6	69.2	0.672	0.136	0.795	0.061
Email-Headers/Email-To-For*	0.136	28.7	80.0	0.649	0.151	0.746	0.037
Email-Headers/IP	0.076	9.3	66.7	0.878	0.075	0.902	0.054
Log/IP	0.067	6.7	20.0	1.000	0.000	0.926	0.013
Log/MAC	0.214	21.4	0.0	0.999	0.003	0.800	0.010
NoProfit-HTML/Email	0.365	41.9	6.7	0.952	0.186	0.629	0.034
References/First-Author*	0.040	4.1	60.0	0.978	0.021	0.963	0.033
ReLIE-Email/Phone-Number	0.063	8.4	60.0	0.788	0.063	0.827	0.069
ReLIE-HTML/All-URL	0.039	4.2	60.0	0.917	0.050	0.886	0.032
ReLIE-HTML/HTTP-URL	0.044	5.0	73.3	0.901	0.051	0.909	0.033
Twitter/All-URL	0.093	9.5	0.0	0.981	0.004	0.889	0.006
Twitter/Hashtag+Citation	0.025	2.5	73.3	0.999	0.003	0.972	0.027
Twitter/Username*	0.025	2.5	100.0	1.000	0.000	0.976	0.012
Average	0.121	17.0	43.6				

Chapter 6

Regex golf

6.1 Overview

Regex golf has recently emerged as a specific kind of code golf, i.e., unstructured and informal programming competitions aimed at writing the shortest code solving a particular problem. A problem in regex golf usually consists in writing the shortest regular expression which matches all the strings in a given list and does not match any of the strings in another given list. Examples of such lists could be the names of all winners of an US presidential election and of the names of all losers (the specific constraints on the contents of these lists will be clarified later, e.g., their intersection must be empty). A trivial way for generating systematically a regular expression with these requirements consists in building a disjunction of all the desired matches—i.e., all the matches glued together by the | character which, in common regex syntax, means "or". To reward non trivial solutions, the *score* assigned to a given solution is higher for more compact expressions.

There has recently been a growing interest toward regex golf in the programmers' communities, motivated more by the challenge itself than by the actual utility of any given problem. Such interest has been fueled further by a blog post of a famous researcher—Peter Norvig—in which a simple yet powerful algorithm for solving regex golf problems systematically is proposed [157]. Norvig points out that problems of this sort are related to set cover problems, which are known to be NP-hard, and describes a greedy algorithm which is very efficient and works well in a number of cases, while at the same time identifying the fundamental trade-offs made in his proposal.

In this thesis work, we propose a methodology based on Genetic Programming (GP) for generating solutions to regex golf problems—a *regex golf player*. We generate a population of candidate regular expressions represented as trees and constructed with carefully selected regular expression operators and constants. We evolve such population based on a multi-objective fitness which maximizes the correct handling of the provided matches and unmatches while minimizing the length of the regular expression represented by the individual.

We implemented our proposal and assessed its performance on a recently proposed suite of 16 regex golf problems which is very popular. We used as baseline the algorithm proposed by Norvig—the only one we are aware of—and the GP-based system for generating regular expressions for text extraction tasks by examples [16]—the only GP-based system that was available when we perfomed the experimental capaign. Our proposal compares very favorably to the baseline and obtains the highest score on the full suite. We also attempted the construct a baseline based on scores obtained by human players, which is difficult because no structured collections of human players results are available: however, we collected several results by crawling the web and found that our proposal is ranked in the top positions.

A prototype of our regex golf player is available at http://regex.inginf.units.it/golf.

6.2 Related Work

The only algorithm explicitly designed for solving regex golf-related problems which we are aware of is the one by Peter Norvig mentioned in the introduction. We used this algorithm as baseline for our proposal.

Several proposals for learning regular expressions from examples exist for *text extraction* problems [11, 15, 16, 44, 118, 133, 209]. Text extraction from examples is radically different from regex golf in several crucial points. First, regex golf assumes an input stream segmented so that the input strings listed in the problem specification are processed by the solution one segment at a time. Text extraction requires instead the ability to identify and extract specific portions from a longer stream. In other words, regex golf consists in binary classifying input strings whereas text extraction requires the identification in the input string of the boundaries of the substring(s) to extract, if any. Second, a regex golf problem places no requirements on how strings not listed in the problem specification will be classified. Text extraction requires instead a form of generalization, i.e., the ability of inducing a general pattern from the provided examples. Third, text extraction requires the ability to identify a context for the desired extraction, that is, a given sequence of characters may or may not constitute a match depending on its surroundings. A requirement of this form is not meaningful in regex golf.

For example, a regex golf problem requiring the match of all winners of US presidential elections and no loser may be solved with a disjunction of `ls` and several short regexes [157]. Such a regular expression is not useful for the text extraction problem, because applying it to a superstring of a winner would provide no information about the substring which actually identifies the winner. Furthermore, any string containing the substring `ls` will thus be matched by the regex. On the other hand, a regex generated for text extraction might be applied to regex golf but it would be largely suboptimal: the solution generation process must induce a general pattern and there is clearly no syntactical pattern capable of predicting the names of future US presidents. In other words, learning approaches tailored to text extraction purposefully attempt to prevent any overfitting of the examples which is instead a necessity in regex golf.

Our proposal builds on the text extraction method in [16], which we modified and specialized by taking the specific requirements of regex golf into account. We included the cited method in the baseline because, although it was designed for a different problem, it is available as a webapp[1] and its inclusion demonstrates that solving regex golf effectively calls for a specialized solution.

Another proposal for learning regular expressions from examples is [55], but this work considers a problem whose requirements are a mix of regex golf and text extraction. On the one hand, the problem consists in merely classifying input strings without the need of identifying the boundaries of the matching substrings. On the other hand, the problem assumes input streams not necessarily segmented in advance at the granularity of the desired matches and unmatches. Moreover, and most importantly, the cited work aims at inferring a general pattern capable of solving the desired task beyond the provided examples.

Since a regular expression may be obtained from a deterministic finite automata (DFA), approaches for learning a DFA from labelled examples and counterexamples could be used (e.g., [40, 140]; see [60] for a survey). On the other hand, such proposals assume the number of states of the target DFA is known and, most importantly, they are not concerned with minimizing the length of the regular expression corresponding to the generated DFA. While approaches of this form may deserve further investigation, they do not appear to match the specific requirements of regex golf. Similar remarks may be applied also to proposals for induction of non-deterministic finite automata (NFA) from labelled examples [92, 207].

Finally, regex golf might be seen as a problem in the broader category of *programming by examples* (PBE), where a program in a given programming language is to be synthesized based on a set of input-output pairs. Notable results in this area have been obtained recently for problems of string manipulation [99, 148] and some of the corresponding algorithms have been included in the latest release of Microsoft Excel (Flash-Fill functionality). While such approaches are able to deal with *context-free grammars* and are thus potentially able to solve classification problems of the form encountered in regex golf, they use an underlying language which is much richer than regular expressions and thus may not

[1] http://regex.inginf.units.it

generate solutions useful for regex golf.

6.3 The Problem

While the term "regex golf" may indicate any challenge requiring the generation of a regular expression, its usual meaning is the one described in the introduction and formalized as follows.

We consider strings constructed over a large alphabet $\alpha =$ UTF-8. Strings may potentially include arbitrary characters in the alphabet, including spaces, newline and so on. A problem instance is defined by $\mathcal{I} = (M, U)$, where M and U are sets of strings whose intersection is empty.

The problem consists in generating a regular expression which:

1. matches all strings in M;
2. does not match any string in U; and,
3. is shorter than the regular expression constructed as a disjunction[2] of all strings in M.

Note that, for a given problem instance, it might not be known whether a regular expression satisfying the above requirements actually exists. Furthermore, given a solution r' satisfying the three requirements, it might not be known whether there exists a shorter solution r'' satisfying requirements 1 and 2.

Solutions may satisfy requirements 1 and 2 in part. That is, a solution might fail to match one or more strings in M and/or match one or more strings in U. Solutions are thus given a *score* quantifying their behavior in terms of the desired matches and unmatches, as well as their compactness.

We use the score definition in http://regex.alf.nu, from which we have also collected the suite of problem instances for our experimental evaluation. The definition is as follows. Let r be a candidate solution, let n_M and n_U denote the number of elements in M and U, respectively, which are matched by r. The score of r on instance $\mathcal{I} = (M, U)$ is:

$$w_{\mathcal{I}}(n_M - n_U) - \text{length}(r)$$

where $w_{\mathcal{I}}$ is a statically defined value which is meant to weigh the "difficulty" of the problem instance $\mathcal{I}$. Note that the numerical value of the score, as well as the range of possible values, is problem instance-dependent and that a solution may obtain a negative score.

6.4 Our Approach

The system requires a description of the problem instance $\mathcal{I} = (M, U)$ and generates a Javascript-compatible regular expression. A prototype is available at http://regex.inginf.units.it/golf.

Our proposal builds on the text extraction method in [16], which we modified and specialized by taking the specific requirements of regex golf into account. We will summarize the differences at the end of this section.

Every individual of the GP search process is a tree τ, where labels of leaf nodes are taken from a specified *terminal set* and labels of internal nodes from a specified *function set* as follows.

The function set consists of the following regular expressions operators (the central dot $\cdot$ represents a placeholder for a child node): *possessive quantifiers* ($\cdot$`*+`, $\cdot$`++`, $\cdot$`?+` and $\cdot${$\cdot$`,`$\cdot$}`+`), *group* `(`$\cdot$`)`, *character class* `[`$\cdot$`]` and *negated character class* `[^`$\cdot$`]`, *concatenator* $\cdot\cdot$—a binary node which concatenates its children—and *disjunction* $\cdot$`|`$\cdot$. We did not include greedy or lazy quantifiers [89] because, as indicated in [16], these operators lead to execution times which are not practically acceptable.

The terminal set consists of a set of terminals which do not depend on the problem instance $\mathcal{I}$ and other terminals which depend on $\mathcal{I}$. Instance independent terminals are: the alphabetical *ranges* `a-z` and

[2] More precisely, the disjunction of all strings in M, where each string is prefixed by the "start of string" anchor `^` and postfixed by the "end of string" anchor `$`.

Table 6.1: Salient information for the 16 problems.

	Problem name	$\|M\|$	$\|U\|$	$w_{\mathcal{I}}$	Ideal score
1	Plain strings	21	21	10	210
2	Anchors	21	21	10	210
3	Ranges	21	21	10	210
4	Backrefs	21	21	10	210
5	Abba	21	22	10	210
6	A man, a plan	19	21	10	190
7	Prime	20	20	15	300
8	Four	21	21	10	210
9	Order	21	21	10	210
10	Triples	21	21	30	630
11	Glob	21	21	20	420
12	Balance	32	32	10	320
13	Powers	11	11	10	110
14	Long count	1	20	270	270
15	Long count v2	1	21	270	270
16	Alphabetical	17	17	20	340
	Total				4320

`A-Z`, the start of string *anchor* `^` and the end of string anchor `$`, and the *wildcard character* `.`. Instance dependent terminals are: all characters appearing in M, *partial ranges* appearing in M, and *n-grams*.

Partial ranges are obtained as follows. We (i) build the sequence C of all characters appearing in M (without repetitions), sorted according to natural order; (ii) for each maximal subsequence of C which includes *all* characters between subsequence head c_h and tail c_t, build a partial range c_h-c_t. For example, if $M = \{\texttt{bar}, \texttt{den}, \texttt{foo}, \texttt{can}\}$, then the partial ranges are `a-e` and `n-o`.

n-grams are obtained as follows. We (i) build the set N of all n-grams occurring in M and U strings, with $2 \leqslant n \leqslant 4$; (ii) give a score to each n-gram as follows: $+1$ for each string in M which contains the n-gram and -1 for each string U which contains the n-gram; (iii) sort N according to descending score and (iv) select the smallest subset N' of all top-scoring n-grams such that the sum of their scores is at least $|M|$ and each individual score is positive. For example, if $M = \{\texttt{can}, \texttt{banana}, \texttt{and}, \texttt{ball}\}$ and $U = \{\texttt{indy}, \texttt{call}, \texttt{name}, \texttt{man}\}$, then the n-grams are `an` and `ba`, as they are the two top-scoring n-grams and the sum of their scores is $2 + 2$.

A tree τ is transformed into a string r_τ which represents a regular expression by means of a depth-first post order visit—Figure 6.1 shows an example of a tree and the corresponding regular expression (in the caption). In our implementation, each regular expression is evaluated by the Java regular expression engine, which works with possessive quantifiers. However, the regex golf competition being considered accepts only Javascript-compatible regular expressions and Javascript regular expression engine does not work with possessive quantifiers. Hence, we further transform r_τ into a Javascript-compatible regular expression by means of a mechanical transformation [89].

The initial population is generated as follows. Let $n_P = 500$ be the size of the population to be generated. For each string s in M, we generate an individual corresponding to s, built using only the concatenator node and single characters of s as terminals. We generate the remaining $n_P - |M|$ individuals randomly, with the ramped half-and-half method and depth of 1–5 levels.

We drive the evolutionary search based on two fitness indexes associated with each individual. Let r be an individual and let n_M and n_U be the number of elements in M and U, respectively, which are matched by r. The two fitness indexes are: $n_M - n_U$, which has to be maximized (the upper bound being $|M|$), and the length of r (in the Javascript-compatible version), which has to be minimized. We use the *Non-Dominated Sorting Genetic Algorithm II* (NSGA-II) [75] to rank individuals according to their fitness values.

Table 6.2: Best human score and solutions for the 16 problems.

	Problem name	Best human score	Best human solution
1	Plain strings	207	`foo`
2	Anchors	208	`k$`
3	Ranges	202	`^[a-f]*$`
4	Backrefs	201	`(...).*\1`
5	Abba	193	`^(?!.*(.)(.)\2\1)`
6	A man, a plan	177	`^(.)[^p].*$`
7	Prime	286	`^(?!(..+)\1$`
8	Four	199	`(.)(.\1){3}`
9	Order	199	`^.5[^e]?$`
10	Triples	596	`00($|3|6|9|12|15)|4.2|.1.+4|55|.17`
11	Glob	397	`ai|c$|^p|[bcnrw][bnopr]`
12	Balance	289	`^(<(<(<(<<?>?>|.9)>)>)>)$`
13	Powers	93	`^(?!(.(..)+)\1*$)`
14	Long count	254	`((.+)0 \2?1 ){7}`
15	Long count v2	254	`((.+)0 \2?1 ){7}`
16	Alphabetical	317	`.r.{32}r|a.{10}te|n.n..`
	Total	4072	

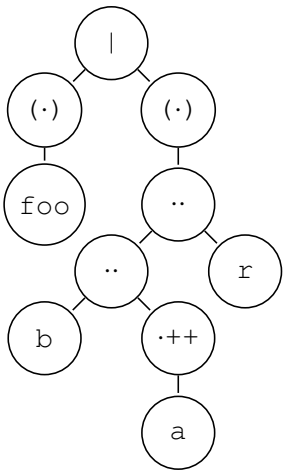


Figure 6.1: Tree representation of the regular expression `(foo)|(ba++r)`.

We evolve the population for a number of generations $n_g = 1000$, according to the following iterative procedure. Let P be the current population. We generate an evolved population P' as follows: 10% of the individuals are generated at random, 10% of the individuals are generated by applying the genetic operator "mutation" to individuals of P, and 80% of the individuals are generated by applying the genetic operator "crossover" to a pair of individuals of P. We select individuals for mutation and crossover with a *tournament* of size 7, i.e., we pick 7 individuals at random and then select the best individual in this set, according to NSGA-II. Finally, we generate the next population by choosing the individuals with highest fitness among those in P and P'. The size of the population is kept constant during the evolution. Upon generation of a new individual, we check the syntactic correctness of the corresponding expression: if the check fails, we discard the individual and generate a new one.

In order to generate a solution for a problem instance $\mathcal{I}$, we evolve $n_e = 32$ independent populations, with different random seeds, obtaining 32 candidate regular expressions. Finally, we choose the regular expression with the highest score.

We remark the key features of our proposal (w.r.t. [16]):

1. a method for constructing the terminal set based on the problem instance $\mathcal{I}$, rather than being defined once and for all;

2. a method for initializing the population based on the problem instance $\mathcal{I}$, rather than being completely random;

3. a different functions set which includes, in particular, a disjunction operator—which is difficult to use in text extraction because it tends to promote overfitting;

4. fitness definitions based on the number of examples handled correctly—definitions proven to be inadequate for text extraction [16];

5. usage of all learning information for synthesizing candidate solutions, that is, without reserving any partition as validation set for assessing the generalization capabilities of those solutions.

6.5 Experimental Evaluation

We considered the 16 problem instances along with the accompanying scores proposed in http://regex.alf.nu. Salient properties of these instances are summarized in Table 6.1 and 6.2. The Table 6.1 shows, for each problem instance, the ideal score—i.e., the score equal to $w_{\mathcal{I}}|M|$ which could be obtained with a zero-length regular expression matching all strings in M and no strings in U. The Table 6.2 shows, for each problem instance, the highest score obtained by (different) human players[3] and the corresponding regular expression.

6.5.1 Baseline

We used as baseline the algorithm by Peter Norvig, which we call Norvig-RegexGolf, and the system for generating regular expressions for text extraction presented in [16], which we call GP-RegexExtract.

We provide a brief outline of Norvig-RegexGolf below. Full details, including the (partially for fun) motivations and design trade-offs can be found in [157]. The solution for a given problem instance $\mathcal{I} = (M, U)$ is obtained as a disjunction of a set of *components*, a component being a short regular expression which matches at least one string in M and does not match any string in U. Initially, a pool of components is built with several heuristics, including the generation of a component for each n-gram of each string in M (up to n=4) and, for each such component, the generation of a component for every possible substitution of a single character with the dot character (meaning "match any" in regular expression syntax). Next a set of components from that pool is built, such that each string in M is matched

[3]The information is obtained from https://gist.github.com/jonathanmorley/8058871.

Table 6.3: Results of the Norvig-RegexGolf as score, score %, score % w.r.t. to best human score and competitive ratio (C.R., see text). For each problem, the score of the best algorithm is shown in bold.

	Norvig-RegexGolf			
Problem	Score	Score %	Hum. %	C.R.
1	**207**	98.6	100.0	66.3
2	**208**	99.1	100.0	105.5
3	191	91.0	94.6	8.5
4	**175**	83.3	87.0	8.0
5	**186**	88.6	96.4	11.5
6	**157**	82.6	88.7	5.1
7	-398	< 0	< 0	1.0
8	**192**	91.4	96.5	17.5
9	**190**	90.5	95.5	8.7
10	**589**	93.5	98.8	6.1
11	**392**	93.3	98.7	25.2
12	-1457	< 0	< 0	1.0
13	-1969	< 0	< 0	1.0
14	189	70.0	74.4	1.0
15	189	70.0	74.4	1.0
16	**294**	86.5	92.7	18.8
Total	-665	-	-	-

by at least one component in the set. Components in the resulting set are then glued together by the or regular expression operator `|`.

Concerning GP-RegexExtract, we reimplemented the algorithm according to the details presented in [16] (see also Section 6.2). We set those parameters which determine the computational weight of GP to the same values for GP-RegexExtract and GP-RegexGolf, in order to allow a fair comparison of the results w.r.t. computational weight: $n_e = 32$, $n_g = 500$ and $n_P = 500$. Note that GP-RegexExtract requires that examples are partitioned in a training set and a validation set: while using it as a regex golf player, we chose to use half of M and half of U strings as training set, and the remaining string as validation set.

6.5.2 Results

We executed each of the algorithms on each problem instance and computed the score of the corresponding solution. Tables 6.3, 6.4 and 6.5 summarize the resulting scores, which are presented as absolute value and as the percentage of the ideal and the best human score associated with each problem instance. Table 6.6 shows the regular expressions generated by GP-RegexGolf for each problem.

It can be seen that GP-RegexGolf outperforms both Norvig-RegexGolf and GP-RegexExtract: 3090 vs. -665 and 249, respectively. In particular, considering individual problem instances, GP-RegexGolf performs better than Norvig-RegexGolf in 6 problems, worse in 8 problems and obtains the same score in 2 problems. Despite obtaining a better score in 8 problems, Norvig-RegexGolf obtains a negative score on the full suite because on three problems (7, 12 and 13) it is not able to generate a non trivial solution: in these problems, the regular expression generated by Norvig-RegexGolf is the disjunction of all the M strings. Our algorithm, on the contrary, generates non trivial solutions for these problems. Concerning GP-RegexExtract, both its score on the full suite and its score on individual instances make it clear that this approach does not the requirements of regex golf.

Tables 6.3, 6.4 and 6.5 list also, for each algorithm, the *competitive ratio* of the solutions [157], defined as the ratio between the length of a trivial solution (disjoining all the strings in M) and the length of the corresponding solution. Note that this index does not take matches or unmatches into account. It can be seen that both Norvig-RegexGolf and GP-RegexGolf generate solutions which are much shorter than the trivial solution for several problems: regular expressions generated by GP-RegexGolf are shorter than

Table 6.4: Results of the GP-RegexGolf algorithm as score, score %, score % w.r.t. to best human score and competitive ratio (C.R., see text). For each problem, the score of the best algorithm is shown in bold.

	GP-RegexGolf			
Problem	Score	Score %	Hum. %	C.R.
1	**207**	98.6	100.0	66.3
2	**208**	99.1	100.0	105.5
3	**195**	92.9	96.5	10.7
4	138	65.7	68.7	6.7
5	184	87.6	95.3	17.2
6	136	71.6	76.8	7.0
7	**188**	35.3	37.0	24.1
8	183	87.1	92.0	11.7
9	186	88.6	93.5	7.3
10	430	68.3	72.2	12.6
11	340	81.0	85.6	17.7
12	**130**	40.6	45.0	11.1
13	**51**	46.4	54.8	109.4
14	**191**	70.7	75.2	1.0
15	**191**	70.7	75.2	1.0
16	132	38.8	41.6	8.0
Total	**3090**	-	-	-

Table 6.5: Results of the Gp-RegexExtract algorithm as score, score %, score % w.r.t. to best human score and competitive ratio (C.R., see text). For each problem, the score of the best algorithm is shown in bold.

	GP-RegexExtract			
Problem	Score	Score %	Hum. %	C.R.
1	170	81.0	82.1	5.0
2	185	88.1	88.9	8.4
3	107	51.0	53.0	7.0
4	-70	< 0	< 0	4.0
5	77	36.7	39.9	4.4
6	-246	< 0	< 0	0.7
7	-52	< 0	< 0	13.4
8	-45	< 0	< 0	7.0
9	-39	< 0	< 0	4.5
10	-106	< 0	< 0	2.4
11	-163	< 0	< 0	4.3
12	-85	< 0	< 0	20.9
13	-47	< 0	< 0	44.2
14	**191**	70.7	75.2	1.0
15	**191**	70.7	75.2	1.0
16	181	53.2	57.1	11.0
Total	249	-	-	-

Table 6.6: Regular expressions generated by GP-RegexGolf.

Problem	Regular expression
1	`foo`
2	`k$`
3	`(^..[a-f][a-f])`
4	`v|[^b][^o][^p]t|ngo|lo|[n]o|rp|rb|ro|ro|rf`
5	`z|.u|nv|st|ca|it`
6	`oo|x|^k|ed|^m|ah|^r|v|^t`
7	`^(?=((?:x[A-Zx])+))\1x`
8	`ell|j|W|ele|o.o|Ma|si|de|do`
9	`ch|[l-p]o|ad|fi|ac|ty|os`
10	`24|55|02|54|00|95|17`
11	`lo|ro|^p|(?=((c)+))\1r|en|^w|y.|le|^p|rr`
12	`((?=((?:<<\>\>\>)*))\2(?=((?:<<<(?=(<*))\4\>\><<<<)*))\3 (?=((?:<<<<<\>\>\>(?=(<*))\6\>\>\>)*))\5(?=((?:<<<<<<)*)) \7^(?=((?:<<\><<)*))\8(?=((?:<<<\>\>\>)*))\9<<)`
13	`^(?=(((x|^)x)+))\1$`
14	`0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111`
15	`0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111`
16	`tena|[^et][^etren](?=((?:(?:ren|eren.(?=((?:(?:ren|[^ren]))+))\2|eren.(?=((?:(?:ren|[^ren]))+))\3))+))\1|eas`

those of Norvig-RegexGolf in 9 problems, longer in 5 problems and with the same length in 2 problems.

Table 6.7 shows the time required by GP-RegexGolf and GP-RegexExtract for generating a solution. It is similar for all the problems (around 50 min) with the exception of 13, 14 and 15. For the latter problems, in which M is composed by very long strings, GP-RegexExtract attempts to generate a regular expression which extracts (rather than just matching) the each M string entirely: this leads to a population composed by very long regular expressions which require long times to be evaluated. The time required by Norvig-RegexGolf is practically negligible (less than a second per problem). All the experiments have been performed on a quad core Intel Xeon E5-2440 (2.40GHz) with 4 GB of RAM.

We wanted to investigate whether our approach can achieve better scores at the expense of increased computational weight. To this end, we repeated the experiments by setting $n_P = 1000$ and $n_P = 1500$, i.e., with an enlarged population: Table 6.8 shows the results in terms of score and competitive ratio. It can be seen that the full score does improve for larger values of n_P. Moreover, with $n_P = 1500$, the number of problems for which GP-RegexGolf score is not worse than Norvig-RegexGolf score is 11 vs. 8 with $n_P = 500$. The computation time for the full suite goes from 820 min for $n_P = 500$ to 1551 min and 2611 min for $n_p = 1000$ and $n_p = 1500$, respectively. As expected, with higher values for n_P GP-RegexGolf takes longer to generate a solution: yet, it is fair to claim that even such longer times may be acceptable for playing to a game of this kind.

Finally, we attempted to assess the performance of our proposal with respect to scores of highly skilled human players. We remark that there are several caveat concerning the assessment of the results obtained by human players. The web site hosting the challenge does not, at the time of this writing, provide a score ranking computed on the full suite of problems—on the other hand, there exists a collection of "Best possible answers collected so far for regex golf" (see Table 6.1) which shows, for each problem, the best solution. Results by human players are advertised on web forums by players themselves, often without providing any actual evidence of their results. On the other hand, there are players which do make some very good solutions publicly available, thereby simplifying the job of other players, which may either attempt to improve those solutions further or may use them for the corresponding problem instance while

Table 6.7: Execution times of GP-RegexGolf and GP-RegexExtract algorithms in minutes.

Problem	GP-RegexGolf	GP-RegexExtract
1	53	51
2	52	52
3	53	65
4	38	25
5	34	18
6	20	23
7	33	43
8	19	27
9	46	21
10	45	25
11	44	47
12	56	71
13	71	269
14	94	289
15	95	173
16	66	64
Total	820	1262

Table 6.8: Scores and competitive ratio (C.R.) of GP-RegexGolf with different values for n_P.

	$n_P = 1000$		$n_P = 1500$	
Problem	Score	C.R.	Score	C.R.
1	207	66.3	207	66.3
2	208	105.5	208	105.5
3	196	11.5	197	12.4
4	146	5.2	147	6.5
5	188	12.5	186	11.5
6	142	6.0	151	4.3
7	188	24.1	188	24.1
8	183	11.7	180	10.5
9	190	8.7	190	8.7
10	456	10.5	354	27.2
11	355	28.2	522	3.2
12	36	7.3	223	26.5
13	65	46.2	40	29.7
14	191	1.0	191	1.0
15	191	1.0	191	1.0
16	259	21.1	237	10.4
Total	3201	-	3412	-

Table 6.9: Total scores obtained by the 10 best humans and by the three algorithms.

	Player	Score
	Total ideal score	4320
	Best human score	4072
1	geniusleonid	4006
2	k_hanazuki	3785
3	bisqwit	3753
4	AlanDeSmet	3736
5	adamhiker	3693
	GP-RegexGolf ($n_P = 1500$)	3412
	GP-RegexGolf ($n_P = 1000$)	3201
6	adamschwartz	3181
7	flyingmeteor	3171
	GP-RegexGolf ($n_P = 500$)	3090
8	jpsim	3060
9	ItsIllak	2939
10	bg666	2683
	GP-RegexExtract	249
	Norvig-RegexGolf	-665

focusing their efforts on the remaining instances. In other words, the score obtained by a given player may actually result from efforts by multiple players. Finally, human players generally do not care to indicate the time they spent for generating a solution.

We collected several human players scores on the full suite from different web locations (including Reddit, Hacker News, Github) which we obtained by querying Google and Twitter with the search string "regex golf". Table 6.9 shows the 10 best scores we found, along with the total ideal score (i.e., the sum of ideal scores on the 16 problems), the best human score (i.e., the sum of the highest human player scores on the 16 problems) and the score of the three considered algorithms.

It can be seen that GP-RegexGolf would rank from 6th to 8th among human players (with $n_P = 1500$ and $n_P = 500$, respectively), whereas the scores of the other two algorithms are significantly lower than those of human players. In other words, leaving aside any caveat about how we gathered human scores, GP-RegexGolf is in the top ten of worldwide regex golf players.

6.6 Remarks

We have proposed and assessed experimentally an approach based on Genetic Programming for playing regex golf automatically, i.e., for generating automatically solutions to challenges which have recently become popular in the programmers' communities. The challenges consist in writing the shortest regular expression that matches all strings in a given list and does not match any string in another given list.

Our approach collects a score that is highly competitive against human players and improve significantly over a challenging baseline including a recently proposed algorithm tailored to this specific problem class and the proposal for automatic generation of regular expressions tailored to text extraction tasks presented in [16]. The time for generating a solution is in the order of tens of minutes and a prototype is available at http://regex.inginf.units.it/golf.

We think that our work shows how a GP-based approach running on modern IT machinery may deliver results, at least for this task, which are practically useful and can compete with humans.

Chapter 7

Evolutionary Learning of Syntax Patterns for Genic Interaction Extraction

7.1 Overview

A huge amount of knowledge is expressed in natural language text and the recent years have seen an explosion of interest in automated techniques capable of extracting such knowledge effectively. Biomedical information extraction, in particular, is a vast and rapidly growing field facing many important challenges [49,101,122,166,204,214,217]. A problem that has captured much attention is the construction of a systematic and structured description of the observed interactions between entities of biomedical interest reported in the scientific literature [90, 102, 132].

In this chapter, we investigate the feasibility of using *evolutionary computing* for attacking a challenging problem in this area: how to identify automatically, in scientific papers, sentences that contain interactions between genes and proteins, based on a dictionary of genes, proteins, interactors, and a small set of example sentences [49, 166, 217]. The problem is challenging because the mere occurrence of dictionary words in a sentence does not imply that the sentence is to be extracted (see Table 7.1). Evolutionary computing techniques have been applied to several problems in Natural Language Processing (NLP), including text summarization [136], sentence alignment for statistical machine translation [175], part-of-speech tagging [7], grammar induction from an annotated corpus [74, 185], word sense disambiguation [76]. A very good and broad review of evolutionary approaches to NLP is [8]. The *binary classification of sentences* that we consider here, though, has received very little attention so far and we are aware of only one evolutionary proposal in this area [42]. The cited work advocates the use of a probabilistic model called the hypernetwork classifier meant to capture the correlation between sets of words and the class in which a sentence containing those words belongs to. Actual values for the correlation between triplet of words and the output class in a learning corpus are computed by means of a stochastic procedure. The experimental evaluation considered $1\,200\,000$ such triplets and executed some preprocessing operations on the text that were not fully described, including deletion of so-called redundant components and insignificant words, as well as conversion of string values into numerical values using a customized dictionary.

In this chapter we propose a different evolutionary technique based on *Genetic Programming* (GP). We learn a model of the *syntax patterns* that occur in the sentences of each class and classify sentences accordingly. Such patterns are expressed in terms of *regular expressions* over standard *part-of-speech* annotations. We solve the difficult problem of actually learning those regular expressions despite the *large alphabet size* involved, by leveraging on recent results in this area [18]. Working at the abstraction level of part-of-speech annotations has important practical advantages: one may build upon the existing

state-of-the-art in NLP annotators, as well as switch to an annotator for a different language or incorporate any advances in annotation technology, without any changes in the framework.

We strove to design our framework without incorporating heuristic rules or findings from previous solutions to this problem. For example, part-of-speech patterns of interest are usually very short and include 2–4 words before and after entities of interest (i.e., genes or proteins) [101, 102]. Furthermore, one could refine a tentative pattern by carefully assigning different costs to each component of the pattern depending on its type—40 coefficients are proposed in [102] for weighing the contribution of 12 groups of part-of-speech tags depending on whether they are aligned correctly with a sample and, if not, depending on the nature of mismatch. We chose to not rely on any problem-specific heuristics because we were more interested in assessing the potential of evolutionary computation in this area than in squeezing accuracy figures. The opposite choice would have made it difficult to isolate the merit of evolutionary learning of syntax patterns from problem-specific heuristics—which of course deserve further investigation.

We remark that our evolutionary learning of syntax patterns is not aimed at grammar induction, a problem that has received much attention in the literature, both for natural text and for formal languages [8, 72]. Grammar induction aims at inferring a set of rules which describe the input data at differing granularity levels and, in the case of natural language, have a hierarchical structure [8, 79]. Usually, although not necessarily, the quality of a solution is assessed from a gold truth of rules [8]. Here we aim instead at *partitioning* text at the granularity level of full sentences, rules need not have any specific structure, no gold truth of rules exists and only the desired partitioning is specified. Of course, techniques for grammar induction could be useful in our scenario as well but it must be emphasized that grammar induction and binary classification of sentences are different problems.

A wealth of literature exist for building classifiers based on Genetic Programming [81], but natural text is outside of the scope of most proposals. Furthermore, classification of natural text is usually done at the level of full documents and such a granularity is excessively coarse in our context [109].

We assess our approach on a realistic dataset and compare the resulting accuracy to significant baseline methods. The results indicate that our GP-based proposal indeed learns syntax patterns from examples effectively, even in NLP scenarios of practical interest.

7.2 The Problem

We aim at generating a classifier $\mathcal{C}$ that takes a natural language sentence s in input and performs a binary classification based on whether s contains a genic-protein interaction.

We generate the classifier based on two *learning sets* S_L^+, S_L^-: sentences in S_L^+ contain a genic-protein interaction whereas sentences in S_L^- do not. Two dictionaries are also available of genes/proteins and interactors, as usually done in in applications of this sort [90, 101, 102].

In order to better appreciate nature and difficulty of the problem, it is useful to emphasize that sets S_L^+, S_L^-, are *not* labelled automatically based on some predefined pattern that is kept hidden to the classifier generation process. Instead, the labelling is done by domain experts which read and analyze the meaning of every single sentence as a whole. Domain experts do not choose the class for a sentence using a predefined pattern—a pattern suitable for classifying sentences as desired may not even exist.

7.3 Our Approach

We aim at inferring a classifier capable of detecting *syntax patterns* of the sentences to be extracted, beyond the mere co-occurrence of relevant words. The classifier operates on sequences of Unicode characters which we call *ϕ-strings*. In a nutshell, we transform each sentence to a sequence of *Part-of-Speech (POS)* annotations, we group annotations of genes/proteins and interactors, and map each annotation to an arbitrarily chosen Unicode character. Full details of this procedure are provided in Section 7.3.1, an example is given in Figure 7.1.

The target classifier $\mathcal{C}$ consists of a set of *regular expressions* $\{r_1^\star, r_2^\star, \dots\}$. The output of $\mathcal{C}$ for an input ϕ-string x will be *positive* (i.e., $\mathcal{C}$ deems that the sentence corresponding to x contains a genic-protein

Sentences w/ genic-protein interaction
In this mutant, *expression* of the **spoIIG** gene, whose transcription depends on both **sigma(A)** and the phosphorylated **Spo0A** protein, **Spo0A-P**, a major transcription factor during early stages of sporulation, was greatly reduced at 43 degrees C.
These results suggest that **YfhP** may act as a negative *regulator* for the transcription of **yfhQ**, **yfhR**, **sspE** and **yfhP**.
These results demonstrate that **sigmaK**-dependent transcription of **gerE** initiates a negative feedback loop in which GerE acts as a *repressor* to limit production of **sigmaK**.
In this study, we used footprinting and gel mobility retardation assays to reveal that bacterially sinthetized **Zta** fusion proteins *bound* directly to six **TGTGCAA**-like motifs within DSL.
Sentences w/o genic-protein interaction
From day 10, a significant increase in platelet count was observed in eight of the ten patients treated with heparin ($p < 0.05$), with return to the initial value after heparin cessation in six of the responders.
Two phosphopeptides, identified as **RS-[32P]SGASGLLTSEHHSR** and **S-[32P]SGASGLLTSEHHSR**, were obtained after stoichiometric *phosphorylation* and trypsinization of the peptide.
Levels of **TSG-14** protein (also termed **PTX-3**) become elevated in the serum of mice and humans after injection with bacterial lipopolysaccharide, but in contrast to conventional acute phase proteins, the bulk of **TSG-14** *synthesis* in the intact organism occurs outside the liver.
No mutations were found in follicular adenomas.

Table 7.1: Eight sentences of the corpus used in our experimentation, 4 which include (left) and 4 which do not include (right) genic-protein interactions. For the sake or comprehension, we highlighted in bold those words belonging to the D_{genes} dictionary and in italic those belonging to the $D_{\text{interactors}}$ dictionary (see Section 7.3.1).

interaction) if at least one $r_i^\star$ is matched by x or by a non-empty substring of x.

We attack the complex problem of actually generating $r_i^\star$ by means of a Genetic Programming (GP) procedure inspired by recent proposals for learning regular expressions from examples [15, 18]. The cited proposals are designed for solving text extraction problems at the level of short text snippets. We had to modify and tailor these proposals for a classification problem in a different domain—properties of text snippets and of POS sequences seem to be quite different. In particular, there are two significant differences between the two scenarios. First, regular expressions include constructs which enable generalization and compactness when applied on actual text but that are not very meaningful on sequences of POS annotations, i.e., character classes `\w` and `\d`. Second, the cited works assume that the examples consist of exactly the text snippets to be extracted, whereas in our context the examples consist of full sentences: the information to be captured by the target regular expression is much more noisy and diluted.

The number of regular expressions in the set $\mathcal{C}$ is not determined in advance and is instead discovered automatically. Specifically, we generate regular expressions one at a time by means of a *separate-and-conquer* procedure. Initially, we generate the first regular expression by using all the available data. Then, once a regular expression is found that provides adequate performance on a *subset* of the examples, we restart the evolutionary search from the scratch by using only the remaining examples that are not yet solved adequately. This procedure is also inspired by a recent proposal designed for extraction of short text snippets [21]: differently from the cited paper, here we focus on classification instead of extraction and allow the generation of regular expressions that do not exhibit perfect precision.

7.3.1 Sentence representation

Let D_{genes} and $D_{\text{interactors}}$ be the statically available dictionaries of words representing genes and interactors, respectively: Table 7.2 shows portions of the two dictionaries which we actually used in our experimentation. We transform each sentence s into a ϕ-string x, as follows. For ease of presentation, in the following we will always use the term "gene" to mean either a gene or a protein.

1. We split s into a sequence $\{t_1, \ldots, t_n\}$ of tokens according to the Penn-Treebank procedure[1].

2. We execute a *Part-of-Speech* (POS) annotator over $\{t_1, \ldots, t_n\}$ and obtain a sequence $\{a_1, \ldots, a_n\}$ of annotations, with $a_i \in A$. The set A of possible annotations depends on the specific POS annotator being used; we assume that three disjoint subsets $A_{\text{verb}}, A_{\text{adj}}, A_{\text{noun}}$ of A exist which correspond to verbs, adjectives, and nouns, respectively—A may include other elements not contained in $A_{\text{verb}} \cup A_{\text{adj}} \cup A_{\text{noun}}$. In our experiments we used the annotator developed by the Stanford Natural Language Processing Group[2]: Table 7.3 shows a partial list of the elements of the set A corresponding to this POS annotator.

3. We modify the sequence $\{a_1, \ldots, a_n\}$ of annotations, as follows. We say that $t \in^* D$ if t is equal to or starts with an element of D—e.g., if $D = \{\texttt{sigmaB}, \texttt{katX}\}$, then $\texttt{sigmaB-dependent} \in^* D$ and $\texttt{katX} \in^* D$. For each i,

   - if $t_i \in^* D_{\text{genes}}$, we set $a_i := \texttt{GENEPTN}$;
   - if $t_i \in D_{\text{interactors}}$ and $a_i \in A_{\text{verb}}$, we set $a_i := \texttt{IVERB}$;
   - if $t_i \in D_{\text{interactors}}$ and $a_i \in A_{\text{adj}}$, we set $a_i := \texttt{IADJ}$;
   - Finally, if $t_i \in D_{\text{interactors}}$ and $a_i \in A_{\text{noun}}$, we set $a_i := \texttt{INOUN}$.

   We denote with $A' = A \cup \{\texttt{GENEPTN}, \texttt{IVERB}, \texttt{IADJ}, \texttt{INOUN}\}$ the set of possible annotations after this step.

[1] http://www.cis.upenn.edu/~treebank
[2] http://nlp.stanford.edu/software/tagger.shtml

D_{genes}	$D_{\text{interactors}}$
amyE	abrogation
bmrUR	activation
ComK	destabilization
DksA	expression
Esig29	repression
GerE	affect
katX	bind
KinC	destabilize
sigmaH	exhibit
SpoIIAB	regulate
others	*others*

Table 7.2: Portions of the two dictionaries of genes/proteins and interactors used in our experimentation.

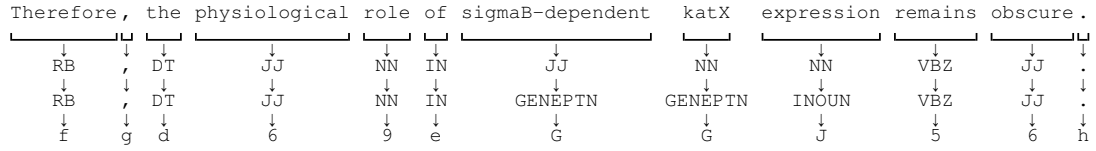


Figure 7.1: Example of sentence representation. The original textual sentence s (top) is transformed into a ϕ-string x (bottom) through two cascaded transformations: one producing a sequence of POS annotations and another where certain annotations are modified using the dictionaries (Section 7.3.1).

4. Let U be the set of characters including digits, lowercase letters and uppercase letters and let $\phi : A' \to U$ be an injective function which maps annotations to characters (see rightmost column of Table 7.3). We obtain the ϕ-string x from $\{a_1, \ldots, a_n\}$ by concatenating the characters resulting from the application of ϕ to each element of the sequence of annotations, i.e., we set $x = \phi(a_1) \ldots \phi(a_n)$.

Figure 7.1 shows the intermediate and final outcomes of the procedure here described when applied to an example sentence.

7.3.2 Regular expression generation

We here describe our procedure based on GP for obtaining a regular expression $r^\star$ from two *training sets* X^+, X^- of ϕ-strings. The aim of this procedure is to generate a regular expression $r^\star$ such that: (i) for each ϕ-string x in X^+, x, or a non-empty substring of x, match $r^\star$; and (ii) for each ϕ-string x in X^-, x and all the substrings of x do not match $r^\star$. The relation between the training sets and the learning sets available for synthesizing the classifier (Section 7.2) will be clarified later.

Solution representation and fitness definition

We represent a candidate solution, i.e., an *individual*, with a tree. Each tree represents a regular expression, as follows. The set of terminal nodes is composed of the wildcard character `.` and of each character in range(ϕ), i.e., each character in U which corresponds to an annotation in A'. The set of function nodes is composed of: the concatenator $\bullet\bullet$; the character class `[`$\bullet$`]` and negated character class `[^`$\bullet$`]`; the possessive quantifiers $\bullet\star+$, $\bullet{+}+$, $\bullet?+$ and $\bullet\{\bullet,\bullet\}+$; and the non-capturing group `(?:`$\bullet$`)`. A tree represents a regular expression by means of a depth-first visit in which each $\square$ symbol in a non-terminal node is replaced by the representation the corresponding child node.

The *fitness* of an individual r quantifies the behavior of the individual over the training sets. We define the fitness as a tuple $f(r) := (\text{FPR}(r, X^-) + \text{FNR}(r, X^+), \text{FPR}(r, X^-), \ell(r))$, where $\ell(r)$ is the length of the regular expression represented by r and $\text{FPR}(r, X^-)$ and $\text{FNR}(r, X^+)$ are the False Positive

a	Meaning	Subset	$\phi(a)$
VB	verb, base form	A_{verb}	0
VBD	verb, past tense		1
VBG	verb, pres. part. or gerund		2
VBN	verb, past participle		3
VBP	verb, pres. tense, ¬3rd p. s.		4
VBZ	verb, pres. tense, 3rd p. s.		5
JJ	adjective or numeral, ordinal	A_{adj}	6
JJR	adjective, comparative		7
JJS	adjective, superlative		8
NN	noun, common, sing. or mass	A_{noun}	9
NNP	noun, proper, singular		a
NNPS	noun, proper, plural		b
NNS	noun, common, plural		c
CC	conjunction, coordinating		d
CD	numeral, cardinal		e
DT	determiner		f
	other annotations		
GENEPTN	gene or protein	$A'\backslash A$	G
IVERB	verb interactor		R
IADJ	adjective interactor		P
INOUN	noun interactor		J

Table 7.3: A partial list of the elements of A': for space constraints, only a subset of $A\backslash(A_{\text{verb}} \cup A_{\text{adj}} \cup A_{\text{noun}})$ is shown.

Rate and False Negative Rate, respectively, of r on the training sets. In more detail, FPR(r, X^-) is the percentage of ϕ-strings $x \in X^-$ for which x, or a non-empty substring of x, match r; FPR(r, X^+) is the percentage of ϕ-strings $x \in X^+$ for which x and all the substrings of x do not match r. For all the elements of the fitness tuple, the lower, the better. The first objective (i.e., FPR(r, X^-) + FNR(r, X^+)) is a proxy for the accuracy of classification of r on the examples in X^-, X^+. We chose not to use accuracy in order to accommodate possibly unbalanced learning sets (our experimental evaluation used balanced sets, though).

We *rank* individuals based on their fitness tuples according to Pareto-dominance and lexicographic order, as follows. First, individuals are sorted by their Pareto frontier: an individual belongs to the i-th frontier if and only if it is Pareto-dominated only by individuals, if any, belonging to j-th frontier, with $j < i$——an individual Pareto-dominates another individual if it is better on at least one fitness element and not worse on the other two elements. Second, a total ordering is established among individuals in the same Pareto frontier: individuals with lower FPR + FNR come first; in case of equal FPR + FNR individuals with lower FPR come first; in case of equal FPR + FNR and equal FPR individuals with lower ℓ come first.

As pointed out in the introduction, we purposefully avoided to include any problem-specific knowledge in the fitness definition—e.g., part-of-speech tags associated with either genes or interactors do not have any special status. We use a multiobjective approach aimed at maximizing accuracy while promoting compact solutions for preventing bloat [71]. We add a third objective beyond accuracy and length because, as detailed in Section 7.3.3, the classifier does not consist of a single regular expression and is instead composed of a set of regular expressions generated from progressively smaller training sets. A form of evolutionary pressure on the FPR of each member of the set turns out to be beneficial to the aggregate FPR of the full set.

Search procedure

We execute the GP search with a population of n_{pop} individuals. The initial population is composed of a portion of randomly generated individuals and a portion of individuals designed to match at least one ϕ-string in X^+. In detail, we build 3 individuals generated from each $x \in X^+$:

1. an individual r representing a regular expression which is equal to the ϕ-string x;
2. an individual r' obtained by replacing in r each leaf node not included in $\{\phi(\texttt{GENEPTN}), \phi(\texttt{IVERB}), \phi(\texttt{IADJ}), \phi(\texttt{INOUN})\}$ with a subtree corresponding to the regular expression $\texttt{[\^{}}\phi(\texttt{GENEPTN})\phi(\texttt{IVERB})\phi(\texttt{IADJ})\phi(\texttt{INOU}$ i.e., with the character class which excludes characters corresponding to genes and interactions;
3. an individual r'' obtained by replacing in r' consecutive repetitions of the character class which excludes genes and interactions with $\texttt{[\^{}}\phi(\texttt{GENEPTN})\phi(\texttt{IVERB})\phi(\texttt{IADJ}), \phi(\texttt{INOUN})\texttt{]++}$, i.e., with the subtree corresponding to one or more repetitions of the character class.

If the number of individuals generated by this procedure is greater than n_{pop}, then we remove exceeding individuals chosen at random (this event does not occur in our experimental setting); otherwise, we generate missing individuals at random with a ramped half-and-half method.

We evolve the initial population by means of the following procedure. At each iteration (or *generation*), we generate n_{pop} new individuals: 80% of them by crossover between individuals in the current population; 10% of them by mutation of individuals in the current population; 10% of them at random with a ramped half-and-half method. We build the new current population by retaining only the n_{pop} individuals with best fitness from the resulting $2n_{\text{pop}}$ individuals (current population and new generated individuals). We select individuals for either crossover or mutation by means of a tournament selection of size 7. Whenever we generate an individual which represents a not valid regular expression, we discard that individual and generate a new one.

We enforce *genotypic diversity* among candidate solutions, i.e., whenever an individual r_1 is generated which represents the same regular expression represented by another individual r_2 in the current population, then r_1 is discarded and another individual is generated—a similar mechanism is used in [128] for preventing the creation of duplicated solutions. We chose to include this simple mechanism in the search—not present in [18]—because our earlier experiments clearly demonstrated its effectiveness in our scenario, the quality of generated solutions being substantially improved without any significant increase in processing time. Such an improvement is perhaps not surprising, because we do not start the search with a fully random population and because a multiobjective GP search may greatly benefit from an explicit mechanism for promoting forms of diversity among candidate solutions, either at the genotypic level or in the objective space or in their behavior [50, 71]. However, we remark that the mechanism that we have chosen is extremely easy to implement and, in particular, does not involve the problem of choosing how to *quantify* the amount of diversity between individuals. Furthermore, it does not involve the need of defining diversity as a *further objective* to be taken into account during the search, which might be difficult to achieve in our scenario since the resulting fitness would be composed of four indexes and thus more radical changes to the evolutionary strategy could be required [112]. A detailed analysis of the effectiveness of this mechanism, as well as of other possible diversity enforcement criteria (e.g., [50, 71]), is beyond the scope of this work, though.

The search terminates when one of the following occurs: (i) a predefined number of n_{gen} iterations has been executed; or, (ii) the fitness tuple of the best individual has remained unchanged for n_{stop} consecutive iterations. The regular expression represented by the best individual of the final population is the outcome of the search procedure.

7.3.3 Classifier generation

We apply the procedure described in Section 7.3.1 and transform the *learning* sets S_L^+, S_L^- to sets of ϕ-strings X_L^+, X_L^-, respectively. Then, we randomly sample X_L^+, X_L^- to build the *training* sets X^+, X^-, respectively, ensuring that $\frac{|X^+|}{|X^-|} = \frac{|X_L^+|}{|X_L^-|}$.

We start from an initially empty set of regular expressions ($\mathcal{C} = \varnothing$) and repeat the following iterative procedure (τ_{FPR} is a predefined threshold): 1) execute a search on X^+, X^- and obtain $r^\star$; 2) if either $\text{FPR}(r^\star, X^-) \leqslant \tau_{\text{FPR}}$ or the search terminated after executing n_{gen} generations, then assign $\mathcal{C} := \mathcal{C} \cup \{r^\star\}$, otherwise terminate; 3) remove from X^+ the ϕ-strings x for which x, or a substring of x, match $r^\star$; 4) if X^+ is empty, then terminate, otherwise go to step 1. The outcome of this procedure is a classifier $\mathcal{C}$.

We perform n_{job} independent executions of the procedure, all starting with the same training sets X^+, X^- but with different random seeds. Thus, we obtain n_{job} (possibly) different classifiers and choose the one with lowest error rate on the learning sets X_L^+, X_L^-. In other words, we validate the n_{job} classifiers on data which was not available during the training in order to prevent overfitting the data.

In our experimentation, we set $\tau_{\text{FPR}} = 0.3$, $n_{\text{gen}} = 1000$, $n_{\text{stop}} = 200$, $n_{\text{pop}} = 1000$, and $n_{\text{job}} = 8$; we chose these values after preliminary experimentation and basing (in particular for n_{gen}, n_{pop}, and n_{job} parameters) on the abundant literature about GP.

7.4 Experimental Evaluation

7.4.1 Datasets and baselines

We used a corpus of 456 sentences built by joining two corpora, both derived from genic-protein interactions extraction challenges of biomedical interest[3]. Each sentence was labelled by a domain expert. Table 7.1 shows some samples. It can be seen that the mere presence of words in the dictionaries D_{genes} and $D_{\text{interactors}}$ does not suffice to qualify a sentence as containing a genic-protein interaction.

In order to assess our results, we implemented several alternative classification techniques: a state-of-the-art evolutionary approach for inferring patterns from examples, two approaches that embed a large amount of problem-specific knowledge and two established methods for text classification.

DFA-based pattern evolutionary inference

We implemented a classifier based on the *Smart State Labelling Evolutionary Algorithm (SSLEA)* proposed in [140]. The cited work is a state-of-the-art algorithm for learning *deterministic finite automata (DFA)* from examples of the desired classification behavior. SSLEA was developed a few years after a competition that was highly influential in the grammar learning community and outperformed (optimized versions of) the winners of the competition, on the same class of problems [60, 126] and even in the presence of noisy data.

SSLEA represents a candidate solution (i.e., a DFA) by a pair composed of an output vector of size n and a transition matrix of size $n \times |\alpha|$, where n is the number of states in the target DFA and $|\alpha|$ is the number of symbols in the input alphabet: the former has one element for each DFA state and each element contains the label (accept or reject) for the corresponding state; the latter contains, for each state and transition, the corresponding destination state index. SSLEA implements a form of hill-climbing in which the fitness of a solution is the rate of examples classified correctly. The search terminates when either a DFA with perfect fitness is found or a predefined number n_{it} of iterations have been executed. We refer the reader to the cited work for full details.

We implemented SSLEA and applied it to ϕ-strings. After some exploratory experimentation, we found that it delivers best results with $\alpha = \text{range}(\phi)$, $n = 7$ and $n_{\text{it}} = 5000$ and we used these values in our assessment. In particular, we verified experimentally that increasing the number n_{it} of available iterations, even by a large amount, did not lead to better accuracy on the testing data.

Problem-specific baselines

We implemented two classifiers that embed a substantial amount of problem-specific knowledge.

[3] http://genome.jouy.inra.fr/texte/LLLchallenge/#task1 and https://www2.informatik.hu-berlin.de/~hakenber/corpora/#bc

The classifier that we call *Annotations-Co-occurrence* classifies a sentence s positively if and only if s contains at least two genes/proteins and at least one interactor—i.e., it is tightly tailored to this specific problem.

The classifier that we call *Annotations-LLL05-patterns* is built on results from [102], which proposes a method for identifying syntax alignment-patterns that describe interactions between genes and proteins in scientific text. An alignment-pattern is described in terms of sequences of salient POS annotations which must occur in a sentence, but that does not specify any constraint on type and quantity of further annotations that might occur between those salient annotations. The method exploits a fair amount of domain-specific knowledge for tuning tens of coefficients used for weighting alignment-pattern errors related with specific POS annotations, as well as with genes and interactors. The cited work provides a list of the 10 most frequent alignment-patterns learned from the *whole* corpus used in that paper, which is much larger than ours (≈ 1000 sentences). We built a classifier which classifies a sentence s positively if and only if s matches at least one of the 10 alignment-patterns in the aforementioned list. Our corpus is composed, for $\approx 90\%$ of the sentences, of a strict subset of that corpus.

Established methods for text classification

Finally, we implemented two well-established schemes for text classification [184], one based on Naive Bayes and the other based on Support Vector Machines (SVM).

We pre-process each sentence s as follows: (i) we replace each occurrence of a string contained in $D_{\text{interactors}}$ with `_interactor`; (ii) we replace each occurrence of a string contained in D_{genes} with `_geneptn`; (iii) we convert s to lowercase; (iv) we replace each non alphabetic (i.e., `[^a-zA-Z_]`) character with a space; (v) we perform stemming to each word (except of `_geneptn` and `_interactor`). We execute steps i and ii in order to exploit the knowledge embedded in the dictionaries, which would otherwise not available to the classifier.

In order to build the classifier:

1. We build a sorted set W of words composed of all words occurring at least once in the learning sets S_L^+, S_L^-.
2. We transform each sentence s into a vector $\mathbf{f}'$ in which the ith element corresponds to the number of occurrences in s of the word $w_i \in W$.
3. We execute a *feature selection* procedure (which is detailed below) for identifying the k elements of $\mathbf{f}'$ which best discriminate between sentences in S_L^+ or S_L^-. Let $\mathbf{f}$ be the vector obtained by keeping only those k elements from $\mathbf{f}'$—i.e., $\mathbf{f}$ contains only the occurrence counts of the words selected in the feature selection procedure.
4. Finally, we train a binary classifier using the vectors $\mathbf{f}$ corresponding to the sentences in the learning sets S_L^+, S_L^-.

Having built the classifier, to classify a previously unseen sentence s we: (i) pre-process s as described above; (ii) obtain the corresponding feature vector $\mathbf{f}$; and, finally, (iii) input $\mathbf{f}$ to the trained classifier.

The feature selection procedure is based on the vectors $\mathbf{f}'$. This procedure takes two numerical parameters k, k', with $k' \gg k$, and works in two steps, as follows. In the first step we compute, for each ith feature the relative difference δ_i between its mean value across sentences of the two sets:

$$\delta_i = \frac{\left| \frac{1}{|S_L^+|} \sum_{S_L^+} f_i' - \frac{1}{|S_L^-|} \sum_{S_L^-} f_i' \right|}{\max_{S_L^+ \cup S_L^-} f_i'}$$

We then select the k' features with the largest δ_i—among those for which $\max_{S_L^+ \cup S_L^-} f_i' > 0$. In the second step we compute, for each ith feature among the k' selected at the previous step, the mutual information I_i with the label—the label being a binary value which is positive for elements in S_L^+ and negative for elements in S_L^-. We then select the k features with the greatest I_i.

Classifier	Accuracy avg	Accuracy sd	FPR avg	FNR avg
Ann.-Co-occurrence	77.8		40.0	4.5
Ann.-LLL05-patterns	82.3		25.0	10.5
Words-NaiveBayes	51.3		25.0	95.0
Words-SVM	73.8		29.0	23.5
ϕ-SSLEA	59.8	3.8	44.0	33.5
Our method	73.7	1.7	23.5	22.5

Table 7.4: Results of our method and the 5 baselines.

We built two binary classifiers with this scheme, which we call *Words-NaiveBayes* and *Words-SVM*, respectively. Concerning the feature selection parameters k' and k, we set $k' = 1000$ for both the classifiers and we chose the value of k for which each classifier obtained the best accuracy on the learning sets, i.e., $k = 25$ for *Words-NaiveBayes* and $k = 50$ for *Words-SVM*. Concerning SVM parameters, we used a Gaussian radial kernel with the cost parameter set to 1.

7.4.2 Results

We executed a 5-fold cross-validation, i.e., we generated 5 different problem instances from the corpus at random. For each instance we used $\approx 80\%$ of the data for learning and $\approx 20\%$ for testing, i.e., we built the learning sets S_L^+, S_L^- (with $|S_L^+| = |S_L^-| = 188$) and left two testing sets S_T^+, S_T^- aside for assessing the accuracy of the generated classifiers (with $|S_T^+| = |S_T^-| = 40$). We used the very same learning sets and testing sets for all the considered methods—our method and the baseline methods described in the previous section. For ϕ-SSLEA and our method, which are stochastic, we repeated the execution 10 times for each fold.

Table 7.4 shows the results obtained by the 6 methods. The results are expressed in accuracy, FPR and FNR, averaged across the 5 folds: for the two stochastic methods, we also report the accuracy standard deviation of the 10 executions averaged across the 5 folds.

It can be seen that our method and Words-SVM obtain the highest accuracies ($\approx 74\%$) among those 4 methods which learn a classifier only from examples and dictionaries—i.e., without any problem-specific knowledge. GP is indeed able to learn syntax patterns from examples effectively, even in NLP scenarios of practical interest. We believe this result is particularly relevant.

The accuracy of the two problem-specific classifiers is 82.3% and 77.8% for Annotations-LLL05-patterns and Annotations-Co-occurrence, respectively. We believe that the better accuracy exhibited by these methods is not surprising having considered that these methods build on a substantial amount of problem-specific knowledge, as clarified above. Indeed, the accuracy of Annotations-Co-occurrence is only slightly better than our method.

It is interesting to note that the accuracy obtained by ϕ-SSLEA is much worse than ours although the two approaches are based, broadly speaking, on similar tools—evolutionary learning of a DFA vs. evolutionary learning of sets of regular expressions. While this result might appear somewhat surprising—as pointed out above, ϕ-SSLEA exhibits state-of-the-art performance in DFA learning—we believe the reason is because benchmark problems in DFA learning consider short sequences of binary symbols, with training data drawn uniformly from the input space. Settings of this sort do not fit the needs of practical NLP applications, which have to cope with much longer sequences of symbols, from a much larger alphabet, not drawn uniformly from the space of all possible sequences. Our interpretation is corroborated by earlier claims from different authors: benchmark problems for DFA learning from examples are not inspired by any real world application [60] and the applicability of the corresponding learning algorithms to other application domains is still largely unexplored [40].

Figure 7.2 shows one of the classifiers obtained during our experimentation, composed of two regular expressions. For the sake of comprehension, expressions are shown using annotations (A') instead of symbols of U. It can be seen that the generated expressions include salient annotations (`GENEPTN`

$r_1 =$ GENEPTN [^ RB] [^ NNS VBN GENEPTN] ++
GENEPTN [^ LRB DT NNS RRB] [^ LRB NNS]
$r_2 =$. INOUN IN GENEPTN . [^ DT NN]

Figure 7.2: An example of a generated classifier composed of two regular expressions, shown using annotations, instead of symbols of U, for readability.

and INOUN) although those annotations were not given any special status. The two patterns are not easily readable, which is not surprising since we did not include any mechanism for favoring readable expressions. Indeed, readability could be a valuable objective to be pursued and we plan to investigate this respect in future work.

7.5 Remarks

In this chapter we presented an evolutionary method for learning syntax patterns in natural text from examples. We applied this method to a sentence classification problem from the biomedical domain that is practically relevant and very challenging.

Our method is based on GP and builds on recent results for the automatic generation of regular expressions from examples of the desired behavior. We propose a technique for representing sentences as strings of symbols which can be manipulated by common regular expressions. Symbols correspond to POS annotations augmented with problem-specific dictionaries. Working at the abstraction level of POS annotations has many practical advantages, including modularity and the possibility of leveraging the steadily improving state-of-the-art in this area. The number of patterns to be learned from a set of example sentences is not known in advance, but is instead automatically determined by means of a separate-and-conquer procedure.

We assessed experimentally our method on a challenging corpus of 456 hand-labeled sentences and compared it against 5 significant baseline methods. We obtained good results which indicate that GP may indeed learn syntax patterns from examples effectively, even in NLP scenarios of practical interest.

Chapter 8

Syntactical Similarity Learning by means of Grammatical Evolution

8.1 Overview

Many solutions to practically relevant applications are based on techniques that rely on a form of *similarity* between data items, i.e., on a quantification of the difference between any pair of data items in a given feature space. Although such a similarity may be quantified by many different generic functions, i.e., distances or pseudo-distances, a wealth of research efforts have advocated the usage of similarity functions that are *learned* from collections of data pairs labelled as being either "similar" or "dissimilar" [36,123,215]. Indeed, similarity functions constructed by a *similarity learning* algorithm have proven very powerful in many different application domains, as such functions may capture the application-specific similarity criterion described by the available *examples* in a way that fits the application needs more effectively than a generic distance definition.

In this chapter, we focus on the problem of learning a similarity function suitable for syntax-based *entity extraction* from *unstructured text* streams. The identification of strings which adhere to a certain syntactic pattern is an essential component of many workflows leveraging digital data and such a task occurs routinely in virtually every sector of business, government, science, technology. Devising a similarity function capable of capturing syntactic patterns is an important problem as it may enable significant improvements in methods for constructing syntax-based entity extractors from examples automatically [15, 17, 21, 22, 23, 44, 55, 60, 84, 133, 152]. We are not aware of any similarity definition capable of (approximately) separating strings which adhere to a common syntactic pattern (e.g., telephone numbers, or email addresses) from strings which do not.

We propose an approach based on GE, in which we explore systematically a similarity definition space including all functions that may be expressed with a specialized, simple *language* that we have defined for this purpose. The language includes the basic flow control, arithmetic and relation operators. It is expressive enough to describe important, existing similarity definitions, that we use as baseline in our experimental evaluation. A candidate solution, i.e., an individual, represents a program in the language which takes a pair of strings as input and outputs a number quantifying their similarity. Programs are executed with a *virtual machine* that we designed and implemented. The virtual machine is necessary only for assessing the quality of candidate solutions during the evolutionary search: the final solution can obviously be implemented in a more compact and more efficient way based on the specific technology in which the learned similarity function will be inserted.

We assessed our proposal on several tasks representative of practical applications, each task being a large text stream annotated with the strings following a task-specific pattern. We emphasize that we did not learn one similarity definition for each task: instead, we learned a single similarity function from all tasks except for one and then evaluated the behavior of the learned similarity function on the remaining task—i.e., on a syntactic pattern that was not available while learning. The results, averaged across all

the tasks, demonstrate that the proposed approach is indeed feasible, i.e., it is able to learn a similarity function capable of (approximately) separating strings based on their adherence to a given syntactical pattern. Most importantly, the learned function is more effective than the Levenshtein distance and the Jaccard similarity index.

An evolutionary approach to metric learning can be found in [146]. The cited work proposes a general approach for multi-label clustering problems in a given feature space. We focus instead on a different and more specific problem: syntax-based entity extraction from unstructured text streams. Furthermore, we aim at learning a similarity function and do not insist in requiring that the learned function be a distance. Several proposals have advocated genetic approaches to similarity learning in the context of case-based reasoning [195, 211, 212]. In those cases, though, the problem was learning a meaningful similarity criterion between problem definitions, to enable effective comparison of a new problem to a library of known, already solved problems. We consider instead similarity between pairs of strings that are a small part of a problem instance. Our problem statement follows a common approach in similarity learning: input data consist of pairs of data points, where each pair is known to belong to either the same class (i.e., the same pattern) or to different classes [215]. An alternative framework is based on input data which consist of triplets of data points (a, b, c) labelled with the information regarding whether a is more similar to b or to c [103, 183, 213]. Such a *relative comparisons* framework has proven to be quite powerful, in particular, for clustering applications. A relative comparison approach could be applied also to our entity extraction problem and indeed deserves further investigation.

8.1.1 Problem statement

The problem input consists of a set of tasks $\{T_1, \ldots, T_n\}$ where each task describes a *syntactic pattern* by means of *examples*. Task T_i consists of a pair of sets of strings (P_i, N_i): P_i contains strings which adhere to the ith pattern while N_i contains strings which do not adhere to that pattern. The problem consists in learning a *similarity function* $\hat{m}(s, s')$ which, given two strings s, s', returns a *similarity index* capable of capturing to which degree s and s' adhere to the same (unknown) syntactic pattern. That is, intuitively, pairs of strings in P_i should be associated with a "large" similarity index, while pairs consisting of a string in P_i and a string in N_i should be associated with a "small" similarity index. Furthermore, this requirement should be satisfied for all tasks by the same function $\hat{m}$.

In details, the ideal learned function should satisfy the following requirement:

$$\forall i \in \{1, \ldots, n\}, \forall x \in M(P_i, N_i), \forall y \in M(P_i, P_i), x < y \tag{8.1}$$

where $M(S, S') = \{m(s, s') : s \in S, s \in S'\}$. For a given problem input, a function satisfying Equation 8.1 may or may not exist; and, even if it exists, a learning algorithm may or may not be capable of learning that function.

8.2 Our approach

8.2.1 Search space and solution quality

We consider a search space composed of functions that may be expressed with the language L described in Figure 8.1 in the Backus-Naur Form (BNF). The available mathematical operators are defined in the rule concerning the ⟨ValueReturningFunction⟩ non-terminal while relation operators are defined in rule concerning the ⟨Condition⟩ non-terminal. The language includes basic flow control operators and allows defining numeric variables and arrays dynamically. Access to variables and array elements occur by index.

The language is expressive enough to describe commonly used similarity indexes: in particular, we described the Levenshtein distance and the Jaccard similarity index—which we used in our experimental evaluation as baselines—using this language.

We propose an evolutionary approach based on *Grammatical Evolution* (GE) [161, 179]. GE is an evolutionary framework where candidate solutions (*individuals*) are represented as fixed-length numeric

Rules

1. ⟨BlockCode⟩ ::= ⟨RowOfBlockCode⟩
2. ⟨Statement⟩ ::= ⟨Assign⟩ | ⟨CreateArray⟩ | ⟨CreateVariable⟩ | ⟨For⟩ | ⟨If⟩ | ⟨Return⟩ | ⟨SetArrayItem⟩
3. ⟨ValueReturningFunction⟩ ::=⟨Constant⟩ | ⟨GetVariableValue⟩ | ⟨Add⟩ | ⟨Decrement⟩ | ⟨Maximum⟩ | ⟨Minimum⟩ | ⟨GetArrayItem⟩ | ⟨GetArrayLength⟩ | ⟨Division⟩ | ⟨Multiplication⟩
4 ⟨Assign⟩ ::= `var[`⟨ValueReturningFunction⟩`] =` ⟨ValueReturningFunction⟩
5. ⟨CreateArray⟩ ::= `newArray[`⟨ValueReturningFunction⟩`]`
6. ⟨CreateVariable⟩ ::= `createVariable()`
7. ⟨Division⟩ ::= `(`⟨ValueReturningFunction⟩ `/` ⟨ValueReturningFunction⟩`)`
8. ⟨For⟩ ::= `for(index0 = 0; index0 <` ⟨ValueReturningFunction⟩`; index0++)` ⟨BlockCode⟩
9. ⟨If⟩ ::= `if(`⟨Condition⟩`)` ⟨BlockCode⟩ `else` ⟨BlockCode⟩
10. ⟨Return⟩ ::= `return` ⟨ValueReturningFunction⟩
11. ⟨SetArrayItem⟩ ::= `array[`⟨ValueReturningFunction⟩`][`⟨ValueReturningFunction⟩`] =` ⟨ValueReturningFunction⟩
12. ⟨Add⟩ ::= ⟨ValueReturningFunction⟩ `+` ⟨ValueReturningFunction⟩
13. ⟨Subtract⟩ ::= ⟨ValueReturningFunction⟩ `-` ⟨ValueReturningFunction⟩
14. ⟨Maximum⟩ ::= `maximum(`⟨ValueReturningFunction⟩`,`⟨ValueReturningFunction⟩`)`
15. ⟨Minimum⟩ ::= `minimum(`⟨ValueReturningFunction⟩`,`⟨ValueReturningFunction⟩`)`
16. ⟨Multiplication⟩ ::= ⟨ValueReturningFunction⟩ `*` ⟨ValueReturningFunction⟩
17. ⟨GetArrayItem⟩ ::= `array[`⟨ValueReturningFunction⟩`][`⟨ValueReturningFunction⟩`]`
18. ⟨GetArrayLength⟩ ::= `array[`⟨ValueReturningFunction⟩`].length`
19. ⟨Constant⟩ ::= `0 | 1 | ... | 255`
20. ⟨GetVariableValue⟩ ::= `var[`⟨ValueReturningFunction⟩`]`
21. ⟨RowOfBlockCode⟩ ::= ⟨Statement⟩ | ⟨Statement⟩ `\n` ⟨RowOfBlockCode⟩
22. ⟨Condition⟩ ::= ⟨EqualCondition⟩ | ⟨NotEqualCondition⟩ | ⟨GreaterCondition⟩ | ⟨GreaterOrEqualCondition⟩
23. ⟨EqualCondition⟩ ::= ⟨ValueReturningFunction⟩ `==` ⟨ValueReturningFunction⟩
24. ⟨NotEqualCondition⟩ ::= ⟨ValueReturningFunction⟩ `!=` ⟨ValueReturningFunction⟩
25. ⟨GreaterCondition⟩ ::= ⟨ValueReturningFunction⟩ `>` ⟨ValueReturningFunction⟩
26. ⟨GreaterOrEqualCondition⟩ ::= ⟨ValueReturningFunction⟩ `>=` ⟨ValueReturningFunction⟩

Alternative rules

2. ⟨Statement⟩ ::= ⟨CreateVariable⟩
3. ⟨ValueReturningFunction⟩ ::= ⟨Constant⟩
21. ⟨RowOfBlockCode⟩ ::= ⟨Statement⟩

Figure 8.1: BNF grammar for the language L: below the set of alternative rules (see text).

sequences. Such sequences (*genotype*) are translated into similarity functions (*phenotype*) by means of a mapping procedure which uses the production rules in a grammar definition. After early experimentation, we chose to tailor several aspects of the general GE framework to our specific problem.

In our case, we represent an individual with a genotype consisting of a tuple $\mathbf{g} \in [0, 255]^{n_{\text{gen}}}$, where each g_i element is a positive 8-bit integer. We chose $n_{\text{gen}} = 350$ because with such value we were able to obtain, from two suitable genotypes, the phenotypes corresponding to the Levenshtein distance and the Jaccard similarity, according to the mapping procedure described below. Given a genotype, we obtain the corresponding phenotype, i.e., a similarity function expressed as a program l in the language L, according to an iterative *mapping procedure* which works as follows, starting with $l = \langle\text{BlockCode}\rangle$ and $i = 0$: (i) we consider the first occurrence of a non-terminal in l and the corresponding rule in the BNF grammar for L; (ii) among the $n_{\text{rule}} \geqslant 1$ alternatives (i.e., possible replacements separated by $|$ in the rule), we choose the $(j + 1)$th one, with j equals to the remainder between g_i and n_{rule}; (iii) we increment i by one: if i exceeds n_{gen}, we set to 1. The procedure is iterated until no more non-terminals exist in l: since it is not guaranteed that this condition is satisfied in a finite number of iterations, we implemented a mechanism to overcome this limitation. We associate a number c with each non-terminal x in l: the value of c is set to 0 for the starting non-terminal $\langle\text{BlockCode}\rangle$, or to $c' + 1$ otherwise, where c' is the number associated with the non-terminal whose replacement lead to the insertion of x in l. Whenever a non-terminal among $\langle\text{Statement}\rangle$, $\langle\text{ValueReturningFunction}\rangle$, and $\langle\text{RowOfBlockCode}\rangle$ has to be replaced, if its c exceeds a parameter $c_{\max} = 40$, we use the alternative rules shown at the bottom of Figure 8.1 instead of the original ones for those non-terminals—in other words, with this mechanism we pose a depth limit on the derivation trees.

We quantify the quality of an individual encoding a similarity function m by its *fitness* $f(m)$, that we define as follows. Given a numeric multiset I, let $I_{p\%}$ indicate the smallest element $i \in I$ greater or equal to the p percentile of elements in I. Given a pair of numeric multisets (X, Y), we define the *overlapness* function $o(X, Y; p) \in [0, 1]$ as follows:

$$o(X, Y; p) = \frac{|\{x \in X : x \geqslant Y_{p\%}\}| + |\{y \in Y : y \leqslant X_{(100-p)\%}\}|}{|X| + |Y|} \tag{8.2}$$

Intuitively, $o(X, Y; p)$ measures the degree of overlapping between elements of X and Y, assuming that elements in X are in general smaller than elements of Y: when X and Y are perfectly separated, $o(X, Y; p) = 0, \forall p$. The value of p is used to discard extreme (greatest for X and smallest for Y) elements in the multisets. The fitness $f(m) \in [0, 1]$ of m is given by:

$$f(m) = \frac{1}{2n} \sum_{i=1}^{n} o\big(M(P_i, N_i), M(P_i, P_i); 10\big) + o\big(M(P_i, N_i), M(P_i, P_i); 0\big) \tag{8.3}$$

where $M(S, S')$ is defined as for Equation 8.1. In other words, the fitness of m is the average overlapness over the tasks in $\{T_1, \ldots, T_n\}$: for each task, $f(m)$ takes into account the average between the overlapness of the two multisets $M(P_i, N_i)$ and $M(P_i, P_i)$ computed on the whole multisets and after discarding 10% extreme values. The rationale for the latter design choice is to avoid giving too much importance to possible outliers in the data. Note that a similarity function satisfying Equation 8.1 has zero fitness—i.e., fitness should be minimized.

During the evolutionary search, we evolve a fixed-size population of n_{pop} individuals for $n_{\text{iter}} = 200$ generations by means of the *mutation* and *two-point crossover* genetic operators, which are applied to individuals selected by means of a tournament of size 3.

8.2.2 Virtual Machine

We designed and implemented a *virtual machine* (VM) capable of executing programs in language L. A VM program execution takes a pair of strings (s, s') as input and returns the value $m(s, s')$, m being the similarity function represented by the program.

As described in Section 8.2.1, the language allows defining numeric variables and arrays dynamically with access occurring by index. VM provides a running program with a list of numeric variables and a list of numeric arrays. Indexes start from 0 and when a new variable is created the next free index is used: the actual variable/array being accessed is determined by the reminder of $\frac{i}{n_v}$. When execution starts, VM creates two arrays into the arrays list, one for s and the other for s': the ith element of each array contains the UTF-8 representation of the ith character in the corresponding string. The execution stops when a return statement is reached or when the last instruction has been executed: in the latter case, the returned value is $m(s, s') = 0$.

A VM program execution may fail, in which case execution terminates and the returned value is $m(s, s') = 0$. Failure occurs when one of the following conditions is met: division by zero; maximum number n_{max} of executed instructions exceeded; maximum array size n_{array} exceeded—we set $n_{\text{max}} = 40\,000$ and $n_{\text{max}} = 10 \operatorname{length}(s) \operatorname{length}(s')$.

8.3 Experimental evaluation

As described previously, a task describes a syntactic pattern by means of examples, i.e., each task consists of a pair of sets of strings (P_i, N_i): P_i contains strings which adhere to the pattern while N_i contains strings which do not adhere to the pattern. We assess our proposal on several datasets representative of possible applications of our similarity learning method (the name of each dataset describes the nature of the data and the type of the entities to be extracted): HTML-href [17, 22, 23], Log-MAC+IP [17, 22, 23], Email-Phone [17, 22, 23, 44, 133], Bills-Date [21, 23], Web-URL [17, 22, 23, 133], Twitter-URL [17, 22, 23]. Each dataset consists of a text annotated with all and only the snippets that should be extracted.

We constructed a task (P, N) for each such dataset, as follows. Let d denote the annotated text in the dataset. Set P contains all and only the strings that should be extracted from d. Set N contains strings obtained by splitting the remaining part of d. It follows that no pair of elements in $P \cup N$ overlap. The splitting procedure is based on a *tokenization* heuristics that (approximately) identifies the tokens that delimit P strings in d; those tokens are then used for splitting N strings in d as well. For example, if strings in P are delimited by a space, then we split the remaining part of d by spaces and insert all the resulting strings in N. The details of the heuristic are complex because different P strings could be delimited by different characters—we omit the details for ease of presentation.

We performed a cross-fold assessment of our proposed method, i.e., we executed one experiment for each of the 6 tasks resulting from the available datasets. In each ith experiment we executed our method on a *learning set* consisting of all but the ith task. We obtained the actual jth pair (P'_j, N'_j) of the learning set by sampling $2n_{\text{ex}}$ items of the corresponding (P_j, N_j), i.e., $|P'_j| = |N'_j| = n_{\text{ex}}$, with $P'_j \subseteq P_j$, $N'_j \subseteq N_j$, where n_{ex} is a parameter of the experiment which affects the amount of data available for learning.

We used the remaining task (P_i, N'_i) (i.e., all of the examples in P_i and a number $|N'_i| = |P_i|$ of examples sampled randomly from N_i) for quantifying the quality of the learned similarity function $m^\star$—$m^\star$ being the individual with the best fitness after the last generation. Note that we assessed $m^\star$ on a task *different* from the tasks that we used for learning it.

For each task, we repeated the experiment for 5 times, each time using a different random seed. We considered the following indexes for each experiment, which we averaged across the 5 repetitions: the learning fitness LF, i.e., the fitness of $m^\star$ on the learning set; the testing fitness TF, i.e., the fitness of $m^\star$ on (P_i, N'_i); the number #I of instructions in $m^\star$; the average number #S of executed instructions while processing pairs in (P_i, N'_i) with $m^\star$.

We explored two different values for the population size n_{pop}, 50 and 100 individuals, and three different values for the cardinality of sets of examples n_{ex}: 10, 25 and 50.

Table 8.1 provides the key results (with $n_{\text{ex}} = 50$ and $n_{\text{pop}} = 50$), separately for each dataset and averaged across all datasets. To place results in perspective, we provide all indexes (except for LF) also for two baseline definitions: the Levenshtein distance, which counts the minimum number of character insertions, replacements or deletions required to change one string into the other, and the Jaccard similarity

Table 8.1: Results of our method, with $n_{\text{ex}} = 50$ and $n_{\text{pop}} = 50$, and the baselines. Best TF figure highlighted.

Task	LF	TF			#I			#S [$\times 10^6$]		
	GE	GE	Jac.	Lev.	GE	Jac.	Lev.	GE	Jac.	Lev.
HTML-href	0.45	**0.42**	0.64	0.91	1877	174	103	0.22	3.49	2.25
Log-MAC+IP	0.44	**0.08**	0.82	0.91	179	174	103	0.06	0.42	0.75
Email-Phone	0.43	0.64	**0.56**	0.90	352	174	103	0.41	4.62	3.64
Bills-Date	0.49	0.85	**0.59**	0.90	1116	174	103	1.56	2.71	5.19
Web-URL	0.40	**0.30**	0.43	0.92	151	174	103	0.72	23.8	10.00
Twitter-URL	0.48	0.30	**0.29**	0.90	147	174	103	0.84	6.28	8.10
Average	0.45	**0.43**	0.55	0.90	637	174	103	0.64	6.90	4.99

Table 8.2: Results (including learning time t_l) for different values of n_{pop} and n_{ex}.

n_{pop}	n_{ex}	LF	TF	#I	#S [$\times 10^6$]	t_l [s]
50	10	0.37	0.45	552	0.59	52
	25	0.43	0.44	3076	0.56	245
	50	0.45	**0.43**	637	0.64	715
100	10	0.34	0.50	1138	2.76	110
	25	0.40	0.48	1224	0.94	326
	50	0.38	0.49	**443**	**0.44**	1056

index, which considers each string as a set of bigrams and is the ratio between the intersection and the union of the two sets. The key result is that, on average, the definitions synthesized by our method exhibit the best results. By looking at individual tasks, our synthesized definitions outperform Jaccard in three tasks, are nearly equivalent in one task and are worse or slightly worse in the two remaining tasks. Thus, the similarity functions synthesized by our method are more effective at separating strings based on their adherence at a certain syntactic pattern with respect to the traditional Levenshtein and Jaccard metrics.

Table 8.2 provides further insights into our method by providing results averaged across all tasks for various combinations of available examples n_{ex} and population size n_{pop}. It can be seen that, with a larger population ($n_{\text{pop}} = 100$), the amount of learning examples does not impact TF significantly, but more examples lead to more compact and more efficient solutions (smaller #I and #S, respectively). On the other hand, the configuration with smaller population ($n_{\text{pop}} = 50$) exhibits a slight but consistent improvement in TF when the amount of examples grows. It can also be observed that more examples lead to solutions with varying length but that tend to be more efficient (no clear trend in #I and decreasing #S, respectively). This observation suggests that our method might perhaps be improved further by a multiobjective optimization search strategy, where the fitness of an individual would take into account not only its ability of capturing similarity as specified in the learning examples (to be maximized) but also the length of the individual (to be minimized).

Table 8.2 also shows the learning time t_l, averaged across repetition: we performed the experiments on a platform equipped with an Intel Core i7-4720HQ (2.60 GHz) CPU and 16 GB of RAM.

8.4 Remarks

We have investigated the feasibility of learning a similarity function capable of (approximately) separating strings which adhere to a common syntactic pattern (e.g., telephone numbers, or email addresses) from strings which do not. We are not aware of any similarity function with this property, which could enable significant improvements in methods for constructing syntax-based entity extractors from examples automatically—in many application domains, similarity functions learned over labelled sets of data points

have often proven more effective than generic distance definitions.

We have proposed a method based on Grammatical Evolution which takes pairs of strings as input, along with an indication of whether they follow a similar syntactic pattern. The method synthesizes a similarity function expressed in a specialized, simple language that we have defined for this purpose.

We assessed our proposal on several tasks representative of practical applications, with an experimental protocol in which we learned a similarity function on a given set of tasks (i.e., patterns) and we assessed the learned function on a previously unseen task. The results demonstrate that the proposed approach is indeed feasible and that the learned similarity function is much more effective than the Levenshtein distance and the Jaccard similarity index.

We plan to extend our investigation in two ways: first, synthesize a more powerful similarity function, by using a broader set of patterns and a larger amount of labelled data points; in this phase there may certainly be room for further improvements to our Grammatical Evolution method; next, take advantage of the learned similarity function in order to improve methods for syntax-based entity extraction.

Chapter 9

Continuous and Non-Intrusive Reauthentication of Web Sessions based on Mouse Dynamics

9.1 Overview

Stealing of authentication credentials has become a first class security problem which cannot be considered an exceptional event.Mechanisms capable of complementing the traditional authentication procedures, which are based on knowledge of a certain password or possession of a certain cryptographic key, would be highly desirable. Approaches which have been proposed in this respect include usage of the stream of events generated at the human-machine boundary as a "behavioral biometric" property which can be univocally associated with each user—keystrokes [100, 201], mouse trajectories and clicks [5, 114, 151, 153, 188, 189, 201], touch-screen interactions [73]. By comparing the stream of events in a certain session to a previously collected ground truth, one may perform additional authentication checks with high accuracy. Approaches of this kind usually assume that the client machine is instrumented with software capable of collecting all the user-generated events of interest, either at login time or continuously in the background. We believe that this requirement may be too difficult to satisfy in practice, especially for large organizations, and in this work we investigate the feasibility of an alternative approach which is much simpler to implement and deploy.

In this chapter we consider the problem of using mouse dynamics for *continuous reauthentication* in a setting where no specific software may be installed on client machines and mouse-generated events are available only for web traffic. The model resulting from these assumptions allows integrating continuous reauthentication capabilities into the existing infrastructure of large organizations easily. In particular, our model fits several key scenarios: web applications hosted in the cloud, where users authenticate with standard mechanisms; organizations which allow locally authenticated users to access external web applications, and enterprise applications hosted in local servers or private cloud facilities.

The problem of identifying users by means of biometric data comes in two flavors [158]: *verification*, in which the system is required to check whether the current user is really the user he/she claims to be; and *identification*, in which the system is required to identify which is the current user amongst a population of known users. In principle, mouse dynamics might be used for both verification and identification. In this work, we consider a form of verification, because we assume that the connected user claims a certain identity by successfully executing some authentication procedure (the specifics of this procedure are irrelevant). The task of the system consists in continuously checking the actual mouse dynamics and generating an alert in case the observed data do not fit the mouse dynamics of the claimed user. In other words, we do not advocate usage of mouse dynamics as the only tool for authenticating users and suggest instead its use as a layer for a defense-in-depth strategy, i.e., as a complement to other forms of

authentication, intrusion detection, and so on. In this respect, mouse dynamics fit neatly into an emerging framework where authentication credentials are considered just one of the multiple signals to be used for authenticating humans [41]. The threat model assumes an attacker who impersonates a legitimate user in web browsing sessions which last for several minutes on a mouse-equipped platform. This model fits, in particular, credential stealing scenarios where an attacker occasionally or routinely accesses an account fraudulently. The model does not address attackers who perform a session lasting just a few seconds (more details in Section 9.3.3).

Our contribution is the following: (a) we describe a system for capturing GUI-related events for web traffic which does not require any specific software to be installed on client machines and is fully *transparent* to both users and web sites; (b) we describe a procedure for performing continuous reauthentication (i.e., frequent verification of the claimed user identity) based on the observed mouse-generated events; and, (c) we show, based on real data collected in two distinct working environments, that despite the intrinsic limitations of the collection procedure, with respect to the commonly adopted approach of instrumenting client machines for collecting all user activities, the system exhibits accuracy aligned with the state-of-the-art.

Our system consists of an HTTP proxy which may be deployed either close to servers or close to clients, and of a specialized *Collector* application. The proxy is configured for injecting JavaScript code which captures mouse-generated events into all web documents fetched through the proxy. The JavaScript code then sends the collected events in batches to the Collector, which performs continuous reauthentication in the background by comparing the observed mouse dynamics to the mouse dynamics of the claimed user. The proxy also rewrites the fetched web documents in order to circumvent the Same Origin Policy enforced by browsers [1], which would prevent the transmission of data to Collector while interacting with pages fetched from a different web site.

We construct mouse dynamics in a feature space similar to several earlier proposals (e.g., [5, 188]) and composed of 39 features related to position, speed, acceleration, and so on. In order to minimize the amount of events to be collected and sent to the Collector, we chose to: (a) construct a new feature vector only when there is a pause of at least 500 ms in mouse usage; and (b) use in each feature vector only a small amount of the events immediately preceding each pause. As it turns out from our experimental evaluation, this design choice does not harm detection accuracy. We execute continuous reauthentication whenever there is a new feature vector available, by applying a Support Vector Machine calibrated specifically for the user who claims to be connected. We generate an alert whenever the number of recent feature vectors which do not fit the expected profile exceeds a certain user-dependent threshold. This simple filtering boosts the accuracy which one would obtain by considering only the last feature vector generated, while at the same time keeping the time required for generating the first alert in the order of the tens of minutes—a reaction time which, in our opinion, is both reasonable and realistic for a reauthentication mechanism of this kind applied to the considered threat model. We remark that the choice of the threshold is based on training data only and is an integral part of our calibration methodology, that is, we do not focus our analysis on the threshold values which happen to provide the best performance on testing data.

We assess our proposal with real data from 24 users, collected during normal working activity for several working days. We obtain accuracy in the order of 97%, which is aligned with earlier proposals requiring instrumentation of client workstations for intercepting all mouse activity.

9.2 Data capture system

Our data capture system is fully *transparent* to both the user and the target web sites. That is, the user navigates with a normal browser and the target web sites do not need to be modified in any way. The system consists of: (a) a web application which we developed and that we call *Collector*; and (b) a *proxy* which must be placed in between the browser and the target web sites. The system captures all mouse-related events generated by web traffic which travels through the proxy. Each user may thus use his/her preferred browser. The two components need not be physically separated, that is, an organization might choose to integrate Collector in the proxy.

The Collector is composed of a server side code (Collector-S) and a client-side code executed by the browser (Collector-C). Collector-C records all the mouse-related DOM events generated by the user and periodically sends a description of these events to Collector-S, which performs continuous reauthentication in the background by comparing the observed mouse dynamics to the mouse dynamics of the claimed user and alerts administrators in case of a mismatch. Collector-C is able to record also keyboard-related events, which may potentially improve the quality of continuous reauthentication even further, but we have not exploited this possibility in this work. We developed Collector with the Google Web Toolkit (GWT). GWT is a Java framework which allows writing AJAX web applications entirely in Java and is able to transcompile Java code directly in JavaScript code ready to be executed by any web browser.

The proxy: (i) injects the Collector-C code into all the pages sent to the browser, to make it possible recording user's actions transparently to the target site; and (ii) redirects part of the web traffic so as to enable communication between Collector-C and Collector-S without violating the *same origin policy* (SOP) [1] implemented by modern browsers (see below). The code injected by the proxy takes this form: `<script type="text/javascript" src="/GWT-Observer/observer.js"></script>` Upon parsing the received page, the browser will fetch the JavaScript code—i.e., the Collector-C code—from the specified `src` location and execute that code locally. The results produced by Collector-C are sent to `/GWT-Observer`. Since both the `src` location and the `/GWT-Observer` are specified by means of a relative URL, the browser treats our code as if it was part of the monitored web application. In other words, the browser is tricked into believing that Collector-C is fetched from the target web site and communicates with that site. The proxy is configured so as to reroute any traffic to `/GWT-Observer` toward the server in our control which actually executes the Collector-S code.

For example, suppose the browser fetches the New York Times home page (`http://www.nytimes.com`). When the browser renders the received HTML page, it generates a request for fetching the JavaScript code from `http://www.nytimes.com/GWT-Observer/observer.js`. This request will *not* be served by the web server at `http://www.nytimes.com`; instead, it will be rerouted by the proxy to a server in our control. The same rerouting will be applied to all HTTP traffic generated by execution of the Collector-C code, which is directed toward `http://www.nytimes.com/GWT-Observer`.

The Collector-C code is obfuscated and its size is approximately 70 kB, which drops to approximately 22 kB if the browser accepts compressed content. Upload bandwidth usage is in the order of 2.5 kB/s. Although we have not performed a systematic performance analysis, we have not experienced any observable performance penalty during continued usage of the system for our normal working activity. This is not surprising having considered that navigation in modern web sites involves loading many thousands of JavaScript functions whose aggregate size is in the order of the MegaBytes [2]. Indeed, an indirect proof that our approach does not hurt performance is that several large providers are already recording mouse-generated events of their users [4].

The system is highly flexible and may be deployed in a variety of ways. It could be deployed at the organization hosting the web application or at the boundary of an organization for monitoring all outbound web traffic. In the former case the system would monitor a web application, while in the latter it would monitor client workstations. The system could also be placed within an organization and configured to monitor only accesses to certain web applications, which could be either local in the organization or hosted elsewhere.

The system is able to handle also encrypted HTTPS traffic. In case the target web application is not in the same administrative domain of our system, monitoring of HTTPS traffic requires the user to accept a self-signed certificate sent by the proxy in place of the certificate sent by the target application. For example, within our University, HTTPS connections to the exam registration system—an attractive target for credential stealing [3, 105]—would not require any specific actions from the user, while connections to GMail or Facebook, as example, would involve accepting a self-signed certificate generated by the proxy—which, of course, needs to be part of the trusted computing base.

9.3 Mouse Dynamics

Each mouse-related DOM *event* e captured by Collector-C is composed of the following information: (i) timestamp, denoted $t(e)$; (ii) cartesian coordinates of the mouse position with respect to the browser viewport, denoted $x(e)$ and $y(e)$; (iii) event type $s(e)$, which can be one among click, double-click and movement. Collector-C captures events with a time resolution which depends on several factors, including the browser, the client processing power, and so on: we found in our experimentation that the mean time resolution was 25 ms and the median time resolution was 8 ms. Collector-C sends the collected events to Collector-S in batches, every few seconds. Collector-S extracts certain features from the collected events (Section 9.3.1) and compares the observed mouse dynamics to the mouse dynamics of the claimed user (Section 9.3.2).

9.3.1 Features extraction

Given the stream of events $(e_1, e_2, \dots)$ generated by each user, we construct a sequence $\mathcal{T}$ of trajectories. Each *trajectory* T represents how the user moved the mouse in a time span delimited by pauses and is built from the stream of events as follows: (i) we identify all the events e_k such that $t(e_{k+1}) - t(e_k) > 500$ ms; (ii) we split the stream in one or more non-overlapping subsequences, each terminated by one of such events; (iii) we discard all subsequences composed of less than $N_T = 10$ events; (iv) in each remaining subsequence, we put all the N_T events immediately preceding the last event (included) in T and discard the other events; (v) we add T to $\mathcal{T}$ and sort $\mathcal{T}$ by the timestamp of the first event in each trajectory. In other words, all trajectories are composed of N_T events and we generate a new trajectory whenever there is a pause in the stream of mouse-generated events lasting at least 500 ms. We chose to split the mouse-generated stream of events based on pauses and to focus on the final part of each trajectory, based on the assumption that this segmentation allows keeping communication needs to a minimum while at the same time capturing the specific mouse-related tasks to be accomplished (i.e., menu selection, point-and-click, and so on).

We associate a feature vector $\mathbf{f}(T) \in \mathbb{R}^{39}$ with each trajectory $T = (e_1, \dots, e_{N_T})$, as follows. For each sample e_k in T, we compute:

- *direction* $d_k \in \{0, 45, 90, 135, \dots, 360\}$, computed as the direction of the vector $(x(e_k) - x(e_{k-1}), y(e_k) - y(e_{k-1}))$, rounded to multiples of 45 degrees; we set $d_k = 0$ if and only if the vector has exactly zero magnitude (i.e., the mouse did not move between e_{k-1} and e_k);
- *speed* $v_k = \frac{\sqrt{(x(e_k) - x(e_{k-1}))^2 + (y(e_k) - y(e_{k-1}))^2}}{t(e_k) - t(e_{k-1})}$;
- *acceleration* $a_k = \frac{v_k - v_{k-1}}{t(e_k) - t(e_{k-1})}$.

We compute direction and speed excluding $k = 1$ and acceleration excluding $k = 1$ and $k = 2$.

The feature vector $\mathbf{f}(T)$ consists of the following features:

- duration of the trajectory, i.e., $t(e_{N_T}) - t(e_1)$;
- x-extent, i.e., $\max_{k=1}^{N_T} x(e_k) - \min_{k=1}^{N_T} x(e_k)$;
- y-extent, i.e., $\max_{k=1}^{N_T} y(e_k) - \min_{k=1}^{N_T} y(e_k)$;
- number of direction changes, i.e., the number of events e_k for which $d_k \neq d_{k-1}$;
- total covered distance, i.e., $\sum_{k=2}^{N_T} \sqrt{(x(e_k) - x(e_{k-1}))^2 + (y(e_k) - y(e_{k-1}))^2}$;
- average speed, i.e., $\frac{1}{N_T} \sum_{k=2}^{N_T} v_k$
- prevalent direction, i.e., the direction occurring most often;

- prevalent direction, without considering zero values;
- maximum distance between the event position (i.e., $x(e_k), y(e_k)$) and the straight line connecting $(x(e_1), y(e_1))$ to $(x(e_{N_T}), y(e_{N_T}))$;
- global direction, i.e., the direction of the vector $(x(e_1) - x(e_{N_T}), y(e_1) - y(e_{N_T}))$, rounded to multiples of 45 degrees as we did for d_k;
- distance between the first and the last position, i.e., $\sqrt{(x(e_1) - x(e_{N_T}))^2 + (y(e_1) - y(e_{N_T}))^2}$;
- average speed of the last five events, i.e., $\frac{1}{5}\sum_{k=N_T-4}^{N_T} v_k$;
- average speed of the first five events, i.e., $\frac{1}{5}\sum_{k=1}^{5} v_k$;
- counts of the 9 possible directions, i.e., for each n-th direction sector, with $n = 1, \ldots, 9$ being the index for the set $\{0, 45, 90, 135, \ldots, 360\}$, the n-th feature counts the number of events in T for which $d_k = n$;
- $N_T - 2$ features corresponding to the acceleration values $a_3, \ldots, a_{N_T}$;
- $N_T - 1$ features corresponding to the speed values $v_2, \ldots, v_{N_T}$.

We hence obtain a sequence F of feature vectors from $\mathcal{T}$, which is itself obtained from the stream of events.

9.3.2 Detection methodology

We construct, in an initial off-line training phase, off-line a classifier for each authorized user to be used in the actual on-line continuous reauthentication phase.

In the *training phase*, we proceed as follows. Let U^- be the authorized user and $U_1^+, U_2^+, \ldots$ a set of other users. We collect a stream of events for each user and build the corresponding sequences of feature vectors F^-_{train} and $F^+_{1,\text{train}}, F^+_{2,\text{train}}, \ldots$, as explained in Section 9.3.1. Next, we train a Support Vector Machine (SVM) binary classifier for the authorized user, the training data being composed of F^-_{train}, which contains training *negative* instances, and $F^+_{\text{train}} = F^+_{1,\text{train}} \cup F^+_{2,\text{train}} \cup \ldots$, which contains training *positive* instances. We truncate each $F^+_{i,\text{train}}$ so as to obtain a balanced training data, i.e., such that $|F^+_{\text{train}}| = |F^-_{\text{train}}|$ and $|F^+_{i,\text{train}}| = |F^+_{i,\text{train}}|, \forall i, j$. We denote the size of the training data by $n_{\text{train}} = |F^-_{\text{train}}| + |F^+_{\text{train}}|$.

We then compute a predefined threshold $\hat{w}$ based on the training data, as follows:

$$\hat{w} = \frac{1}{2} w \left(\frac{\left|\{\mathbf{f} \in F^-_{\text{train}} : \text{SVM}(\mathbf{f}) = \text{positive}\}\right|}{|F^-|} + \frac{\left|\{\mathbf{f} \in F^+_{\text{train}} : \text{SVM}(\mathbf{f}) = \text{positive}\}\right|}{|F^+|} \right) \tag{9.1}$$

where w is a system parameter described below. In other words, $\hat{w}$ is set to the mid-point between the False Rejection Rate (FRR) and the True Rejection Rate (TRR) on the training data, rescaled from $[0, 1]$ to $[0, w]$.

In the *reauthentication phase*, we proceed as follows. Let U^- be the claimed user (i.e., the one who successfully performed an initial authentication) and U the actual connected user who should be reauthenticated: the system collects the stream of events generated by U and constructs the corresponding sequence of feature vectors F. Whenever the system generates a new feature vector $\mathbf{f}$, i.e., whenever there is a pause larger than 500 ms in the mouse-generated stream of events, the system: (i) classifies $\mathbf{f}$ with the classifier trained for the claimed user U^-—we say that the classification outcome $\text{SVM}(\mathbf{f})$ is a *positive* if $\mathbf{f}$ does not fit the mouse dynamics profile for U^-; (ii) counts the number of positives generated in the last w classifications; and (iii) in case the number of positives exceeds the predefined threshold $\hat{w}$

(with $0 \leqslant \hat{w} \leqslant w$), generates an alert. Our filtering strategy has some similarity with the "trust" value associated with each individual classification in [151]. The cited work, though, provides an experimental assessment where a threshold value is chosen based on *all* the available data—including testing data—and does not detail the procedure which should be applied in practice, when operating solely on training data.

The technical details for informing Collector-S of which is the claimed user U^-, as well as those for selecting the corresponding classifier, are irrelevant to this discussion.

9.3.3 Discussion

Usage of mouse dynamics for continuous reauthentication involves solving a key design question. Let $M(U^-)$ denote the mouse dynamics profile for a certain user U^-. Shall the construction of $M(U^-)$ be based solely on data generated by U^-? In this respect, two approaches have been proposed: 1. labelled data of mouse dynamics generated by U^- and by all the other users $U_1^+, U_2^+, \ldots$ known to the system are used [5, 153]; or, 2. only data generated by U^- may be used [188, 189]. The rationale of approach 2 is that, in a system involving a large and dynamic set of users, assuming that the profile of each user requires data from every other user is not realistic. In this work, we followed approach 1, but we believe that the features available to our system would make approach 2 feasible as well.

In this respect, we note that, actually, a third option exists which deserves to be investigated, namely constructing $M(U^-)$ based on labelled data from U^- and from *some* users different from U^-, as opposed to *all* other users. We believe this option is worth exploring because, in our opinion, neither approach 1 nor approach 2 is intrinsically superior to the other. The essential issue is the separability of the features generated by U^- from those generated by impostors. In this respect, since an accurate representation of the mouse dynamics of impostors is not available, we believe it is not possible to tell a priori which of the two approaches results in better separability. In this work, we chose not to follow this third option because the available number of users is too small to perform a meaningful analysis—train the classifier for U^- with the users most similar to U^-, or those who are most different from U^-, or a randomly selected set.

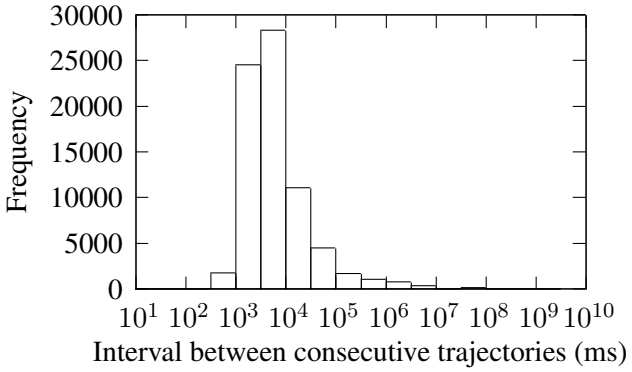
As pointed out in the introduction, the threat model assumes an attacker who impersonates a legitimate user on a mouse-equipped platform in web browsing sessions which last for several minutes. In principle the model could address attackers who perform a short web session lasting just a few seconds, but we believe that in these cases the number of events available to the detection machinery would be too small to generate meaningful alerts, i.e., alerts associated with a reasonable level of accuracy. The threat model could be extended to partly address injection of fraudulent requests in web sessions generated by the legitimate user, for example through malware executing in the browser or in the client machine, or through hijacking of HTTP sessions from a different node. To this end, the system should learn the set of HTTP requests generated during mouse trajectories or shortly after them, and then it should generate an alert when those requests occur at time instants which are too far away from the time intervals of the observed mouse trajectories. Indeed, the ability to discriminate bot-generated traffic from human-generated traffic based solely on webpage-embedded loggers of keyboard and mouse events with excellent accuracy and negligible overhead, has been proven [59]. We have not addressed this extension in this work, though. Of course, an attacker capable of accurately mimicking the traffic generated by Collector-C in a web session originated by the victim user would be able to circumvent the system.

9.4 Experimental evaluation

9.4.1 Dataset

We collected two datasets in different environments and operating conditions, one consisting of a stream of events generated by 6 users and the other by 18 users. In both cases data were collected during normal working activity for several working days. Users of Dataset 1 were monitored for 4 weeks on the average and operated on different hardware equipment, in terms of screen size, screen resolution, actual mouse device—the data for each user being collected entirely on the same hardware. The average number of trajectories per user in Dataset 1 is $|\mathcal{T}| = 2927$. Users of Dataset 2 were monitored for 2 weeks and

Figure 9.1: Histogram of time intervals between consecutive trajectories (i.e., difference between the timestamp of the corresponding first events) considering both the datasets.



operated on homogeneous hardware equipment. The average number of trajectories per user in Dataset 2 is $|\mathcal{T}| = 3229$.

Concerning the way in which we collected the datasets, Dataset 1 reflects the scenario in which the system is deployed by a web application provider: in this scenario, legitimate users access the application mainly from their platform, whereas impostors will likely access from different platforms. Dataset 2 reflects the scenario in which the system is deployed within an organization where platforms are homogeneous.

Figure 9.1 shows an histogram of the time intervals between consecutive trajectories, i.e., difference between the timestamp of the corresponding first events. It is important to remark that this distribution has a long tail (note the log scale on the x axis): the reason is because the user interacts with the browser and with other applications from which our machinery cannot collect mouse events. The considered scenario is thus more challenging than one in which all mouse events can be captured: this affects the trade-off between the classification accuracy which can be achieved and the observation time needed to achieve it. In our dataset, if we consider a *session* of interaction with the browser a time span without any pause longer than 10 min, then the average interval between consecutive trajectories within a session is $\overline{t} = 16.2\,\text{s}$.

9.4.2 Procedure and results

For each user U_i in a dataset, (i) we built the sequence of events F_i^- and the set of sequences of events $F_j^+, i \neq j$, from the other users U_j in the dataset; (ii) we built the training data $F_{\text{train}}^-, F_{\text{train}}^+$ as described in Section 9.3.2, trained the SVM classifier and computed $\hat{w}$; (iii) we applied the classifier to the sequence of feature vectors $F_{\text{test}}^- = F^- \backslash F_{\text{train}}^-$ and measured the False Rejection Rate (FRR); and (iv) finally, for each $j \neq i$, we applied the classifier to the sequence of feature vectors $F_{j,\text{test}}^+ = F_j^- \backslash F_{j\text{train}}^-$ and measured the False Acceptance Rate (FAR)—we thus simulated that U_j is an impostor. We repeated the above procedure 2 times for each user U_i by changing the training set composition, in order to average disadvantageous or advantageous random choices. We experimented on Dataset 1 only, Dataset 2 only, and union of Dataset 1 and Dataset 2—the latter only for one combination of w, n_{train}. That is, we tested the profile of each user against data of other 5, 17 and 23 users, respectively. We measured the performance in terms of FRR, FAR (averaged across the simulated impostors U_j for the sake of brevity) and accuracy (i.e., $1 - \frac{\text{FRR}+\text{FAR}}{2}$).

We experimented with $n_{\text{train}} \in \{1000, 1500, 2000\}$ and with $w = \{50, 100, 200, 350, 500\}$. The number of users was not uniform across all the experiments because, in some configurations, the amount of data available for some users was smaller than the amount of training data required by that configuration.

Table 9.1 shows the results for varying size of the training dataset n_{train}, with $w = 500$. As one would expect, increasing n_{train} indeed delivers better results. In a realistic scenario the user could be asked to train the system for a longer time to improve performances. The second column of Table 9.1 shows the Time to Train (TtT), which is the estimated time needed to collect the training data: TtT is computed

Table 9.1: Results with different values of n_{train} and $w = 500$ for the two datasets. TtT is the Time to Train (see text).

	TtT	Dataset 1			Dataset 2		
n_{train}	(h)	Acc.	FAR	FRR	Acc.	FAR	FRR
1000	4.5	91.4	8.9	8.2	88.2	9.7	14.0
1500	6.8	95.0	6.9	3.1	89.6	8.1	12.7
2000	9.0	96.5	6.1	0.8	92.2	9.5	6.1

Table 9.2: Results with different values of w and $n_{\text{train}} = 2000$ for the two datasets. TtD is the Time to Detection (see text).

	TtD	Dataset 1			Dataset 2		
w	(min)	Acc.	FAR	FRR	Acc.	FAR	FRR
50	13.5	83.3	16.6	16.7	76.4	21.8	25.4
100	27.1	88.5	12.8	10.2	81.4	17.5	19.6
200	54.1	93.5	9.2	3.8	86.6	13.5	13.3
350	94.7	95.6	7.9	1.0	90.6	10.8	8.0
500	135.3	96.5	6.1	0.8	92.2	9.5	6.1

as $n_{\text{train}}\bar{t}$ and represents the total duration of the sessions needed to collect n_{train} trajectories, one being generated each $\bar{t}$ s. The values (up to 9 h) for this figure appear to fit the considered application scenario.

Table 9.2 shows the results for varying size of the filtering window w, with $n_{\text{train}} = 2000$. It can be seen that increasing w improves accuracy significantly: in other words, the larger w, the smaller the variance of the system output. Of course, increasing w results in a longer time to detection (TdT, see second column in Table 9.2, computed as $w\bar{t}$), because more trajectories (and hence more mouse-generated events) have to be observed by the system in order to output the first classification outcome on the connected user. For example, for $w = 350$ the accuracy is larger than 90% for both datasets and the time to detection is in the order of 1.5 h: this figure appears to be practical with respect to the considered threat model. It should be noted, however, that whenever at least w trajectories have been observed, a classification outcome is output soon after each trajectory.

Table 9.3 sums up the results in the configuration which delivers the best average performance ($w = 500$ and $n_{\text{train}} = 2000$) and includes the results for the union of the two datasets. It can be seen that the performance are better for Dataset 1. We think that this is due to the fact that users of Dataset 1 operated on different hardware platforms (see Section 9.4.1) resulting in observed trajectories (and corresponding feature vectors) which are intrinsically more separable. This finding fits the original aim of our proposal, since (i) Dataset 1 corresponds to the scenario of the system deployed by the web application provider and (ii) the architecture of our system is designed to accommodate this scenario, posing no requirements on the clients.

In order to place these results in perspective, Table 9.3 (right side) shows the corresponding results computed according to the methodology usually adopted in the literature (e.g., [5]), which consists in

Table 9.3: Results with different values of $w = 500$ and $n_{\text{train}} = 2000$ (dataset 1 above, dataset 2 below).

	With $\hat{w}$ set as in Eq. 9.1			In EER point		
User	Acc.	FAR	FRR	Acc.	FAR	FRR
Dataset 1	96.5	6.1	0.8	98.1	2.1	1.7
Dataset 2	92.2	9.5	6.1	95.9	4.1	4.1
Dataset 1 $\cup$ Dataset 2	93.5	6.8	6.2	96.0	4.1	4.0

applying the method, for each user, with varying values of the threshold $\hat{w}$ and by selecting the threshold value associated with the Equal Error Rate (EER) point—that is, the $\hat{w}$ values for which FRR = FAR. It can be seen that, according to this evaluation methodology, the numerical values for accuracy are significantly better. Since the EER point cannot be computed with training data only, though, we believe the left side of the table considers a more realistic scenario.

9.5 Remarks

Usage of mouse dynamics as an authentication signal for complementing traditional authentication procedures and for constructing a further layer for a defense-in-depth strategy has been proposed several times in the past. We believe there are at least two major obstacles for a wide adoption of such approaches in practice. First, all published experimental evaluations considered a user base composed of only a few tens of users, hence the effectiveness of the approach over thousands or millions of users is still to be demonstrated. Second, earlier proposals assumed that client machines are instrumented for collecting all user-generated events of interest, which in many cases, including continuous reauthentication of accesses to web applications hosted in the cloud, is unfeasible.

Our work indicates that the latter issue may actually be overcome, thereby greatly improving the potential scope of mouse dynamics as a continuous reauthentication tool. In this chapter, we have shown a system for capturing GUI-related events for web traffic which does not require any specific software to be installed on client machines and is fully transparent to both users and web sites. We have also shown that, despite the intrinsic limitations of the collection procedure, the system exhibits accuracy in terms of FAR and FRR which is aligned with the state-of-the-art. The event capture machinery is sufficiently lightweight to not harm the actual user experience and may be deployed in a variety of scenarios, either close to the client or close to the server, and, perhaps most importantly, also in scenarios where clients and servers belong to distinct administrative domains. The details of the events actually captured could be easily modified to include, e.g., keystroke dynamics, and the detection procedure could be modified in order to take advantage of such additional information [100, 201]. We hope that the architectural advantages illustrated by our work may help in promoting further research aimed at understanding what can be realistically achieved on a very large user population.

Chapter 10

An Author Verification Approach Based on Differential Features

10.1 Overview

In this chapter we describe the approach that we submitted to the 2015 PAN competition [196] for the author identification task[1]. The task consists in determining if an unknown document was authored by the same author of a set of documents with the same author.

We propose a machine learning approach based on a number of different features that characterize documents from widely different points of view. We construct non-overlapping groups of homogeneous features, use a random forest regressor for each features group, and combine the output of all regressors by their arithmetic mean. We train a different regressor for each language.

Our approach achieved the first position in the final rank for the Spanish language.

10.2 Problem statement

A problem instance is a tuple $\langle K, u, L \rangle$ where K is a set of documents $\{k_1, \ldots, k_n\}$ authored by the same author (called known documents), u is a document whose authorship must be ascertained (called unknown document), L is an enumerated value specifying the language of the documents: English, Dutch, Greek or Spanish. All documents in a problem instance are in the same language.

The author verification procedure consists in generating an answer in the form of a real number in $[0, 1]$ which quantifies the degree of confidence of being u authored by the same author of the documents in K: 0 indicates absolute certainty that u was not authored by the same author of documents in K, while 1 indicates absolute certainty that all documents were authored by the same author.

A set of solved problem instances (the *training set*) is available in which, for each problem instance $\langle K, u, L \rangle$, the solution consisting in one between 0 and 1 is provided.

The effectiveness of a method for author identification is assessed using a *testing set* of solved problem instances, as follows. The answers generated by the method for the problem instances in the testing set are compared against the actual values and the comparison outcome is expressed in terms of two indexes: area under the ROC curve (AUC) and $c@1$. AUC is computed basing on the ROC curve plotted by comparing the generated answers against a threshold moving between 0 and 1, hence obtaining a binary classification task. The latter index is computed as $c@1 = \frac{n_c}{n} + \frac{n_u n_c}{n^2}$, where n is the size of the testing set, n_u is the number of unanswered problem instances (i.e., those for which the generated answer was exactly 0.5), n_c

[1] During the competition we discovered several opportunities for fraudulently boosting the accuracy of our method during the evaluation phase. We described these opportunities in a report. We notified the organizers which promptly acknowledged the high relevance of our concerns and took measures to mitigate the corresponding vulnerabilities. The organizers acknowledged our contribution publicly. We submitted for evaluation an honestly developed method—the one described in this document—that did not exploit such unethical procedures in any way.

is the number of correct answers (i.e., those for which the generated answer > 0.5 and the actual answer is 1 and those for which the generated answer < 0.5 and the actual answer is 0).

10.3 Our approach

We propose a machine learning approach based on a number of different features that characterize documents from widely different points of view: character, word, part-of-speech, sentence length, punctuation. We construct non-overlapping groups of homogeneous features and use a random forest regressor for each features group. The output of the resulting ensemble of regressors is the arithmetic mean of the output generated by each random forest.

We train a different regressor for each language. Based on extensive experimention on the training set, we decided to use the same features for problem instances in Dutch, Greek, Spanish but a different set of features for problem instances in English.

10.3.1 Features

We extract a number of different features from each document. For ease of presentation, we group homogeneous features together, as described below.

Word $n$grams (WG) We convert all characters to lowercase and then we transform the document to a sequence of words. We consider white spaces, punctuation characters and digits as word separators. We count all word $n$grams, with $n \leqslant 3$, and we obtain a feature for each different word $n$gram which occurs in the training set documents of a given language.

Character $n$grams (CG) We replace punctuation characters and digits with blank spaces and then sequences of blank spaces with a single blank space. We count all character $n$grams, with $n \leqslant 3$, and we obtain a feature for each different character $n$gram which occurs in the training set documents of a given language.

POS (part-of-speech) tag $n$grams (PG) We apply a *part of speech (POS) tagger* on each document, which assigns words with similar syntactic properties to the same POS tag. For English and Dutch we use the Apache OpenNLP Tools[2], for Greek we use the tagger developed by the Department of Informatics at Athens University of Economics and Business[3] while for Spanish we use TreeTagger[4] [182]. We count all POS $n$grams, with $n \leqslant 3$, and we obtain a feature for each different POS $n$gram which occurs in the training set documents of a given language.

Word lengths (WL) We convert all characters to lowercase and then we transform the document to a sequence of words. We consider white spaces, punctuation characters and digits as word separators. We count the number of words whose length in characters is n, with $n \in \{1, \ldots, 16\}$: we obtain a feature for each value of n.

Sentence lengths (SL) We transform the document to a sequence of tokens, a token being a sequence of characters separated by one or more blank spaces. Next, we transform the sequence of tokens to a sequence of sentences, a sentence being a sequence of tokens separated by any of the following characters: `.`,`;`,`:`,`!`,`?`. We count the number of sentences whose length in tokens is n, with $n \in \{1, \ldots, 40\}$: we obtain a feature for each value of n.

Sentence length $n$grams (SG) We transform each document to a sequence of labels, where each label represents a full sentence and is chosen based on the sentence length (as described in the following). Next, we compute the $n$grams of the resulting labels, with $n \leqslant 2$. In detail, we execute a preliminary

[2] http://opennlp.apache.org
[3] http://nlp.cs.aueb.gr/software.html
[4] http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger

analysis of all documents of a given language in the training set, as follows. For each document, we transform the document to a sequence of sentences as illustrated for the SL features group. Next, we compute the distribution of sentence length across all sentences in the training set and determine the length values associated with the following percentile values: 10%, 25%, 75%, and 90%. In other words, we divide the range of sentence lengths observed in the training set in 5 intervals, with boundaries between intervals determined by the specified percentiles. The label we assign to each sentence corresponds to one of the 5 lenght intervals, i.e., $]0\%, 10\%]$, $]10\%, 25\%]$, and so on: we obtain a feature for each label $n$grams which occurs in the training set documents of a given language.

Word richness (WR) We transform the document to a sequence of words as for the WG features group. Then we compute the ratio between the number of distinct words and the number of total words in the document—this features group contains only one feature.

Punctuation $n$grams (MG) We transform the document by removing all characters not included in the following set: {`,`, `.`, `;`, `:`, `!`, `?`, `"`}—the resulting document thus consists of a (possibly empty) sequence of characters in that set. We then count all character $n$grams of the resulting document, with $n \leqslant 3$, and we obtain a feature for each different punctuation $n$gram which occurs in the training set documents of a given language.

Text shape $n$grams (TG) We transform the document as follows: sequences of digits are replaced by the single character `n`; sequences of alphabetic characters are replaced by a single character: `l` if all the characters in the sequence are lowercase, `u` if only the first character is uppercase, `w` if at least two characters are uppercase; sequences of blank spaces are replaced by a single blank space; other characters are left unchanged. We then count all character $n$grams of the resulting document, with $n \leqslant 3$, and we obtain a feature for each different character $n$gram which occurs in the training set documents of a given language.

10.3.2 Feature selection, normalization, and aggregation

We perform a simple *feature selection* for features in groups WG, CG, PG, and TG. To this end, we apply the following procedure to each of the 4 partitions of the training set for which the language of the documents was the same—in other words, we select different features for each language. We compute the feature values for all the documents in the training set partition. Next, among each group, we sort the features according to their average values on the documents of the partition—greater values coming first. Finally, for each group, we keep the n_{sel} top features. We set $n_{\text{sel}} = 500$ for WG, CG and PG and $n_{\text{sel}} = 100$ for TG.

After the feature selection, we perform a normalization of the features values, as follows. Let $f_i(d)$ be the value of the ith feature for the document d and let G be the group of features (as defined in Section 10.3.1) which includes the feature f_i, we set $f_i(d) := \frac{f_i(d)}{\sum_{f_j \in G} f_j(d)}$. We execute this procedure for all the groups of features, except for WR, which consists of a single feature.

Finally, for the purpose of obtaining a single feature vector for each problem instance, rather than one feature vector for each document, we build a new feature f'_i whose value is obtained from the values of the corresponding f_i for the documents in K and the document u, as follows:

$$f'_i(\langle K, u, L\rangle) = \text{abs}\left(f_i(u) - \frac{\sum_{k \in K} f_i(k)}{|K|}\right) \tag{10.1}$$

In other words, we consider the absolute difference between the feature value for the unknown document u and the average of the feature values for the known documents in K. We also consider a variant of our approach in which the difference is divided by the feature value for u:

$$f''_i(\langle K, u, L\rangle) = \frac{f'_i(\langle K, u, L\rangle)}{f_i(u)} \tag{10.2}$$

	$c@1$				AUC			
Method	EN	DU	GR	SP	EN	DU	GR	SP
RF-abs	0.67	0.74	0.77	0.94	0.718	0.707	0.808	0.992
RF-rel	0.58	0.66	0.77	0.95	0.584	0.776	0.796	0.989
SVM-abs	0.48	0.67	0.69	0.92	0.513	0.707	0.754	0.978
SVM-rel	0.45	0.62	0.66	0.86	0.584	0.645	0.732	0.936
Tree-abs	0.69	0.70	0.53	0.94	0.725	0.708	0.557	0.951
Tree-rel	0.56	0.62	0.69	0.97	0.526	0.595	0.699	0.992

Table 10.1: $c@1$ and AUC for 6 methods.

10.3.3 Regressor

We explored three different regressor algorithms: trees (Tree), random forests (RF), and support vector machines (SVM). In particular, we use the algorithm proposed in [138] for Tree, we use the gaussian kernel and $C = 1$ for SVM [56], and we use the algorithm for regression proposed in [47] with $n_{\text{tree}} = 500$ for RF.

We apply each regressor, both in training and actual regression phase, only to the feature values of the same group. For obtaining an answer in $[0, 1]$ for a problem instance, we average the predictions obtained by the trained regressors on the features groups. In other words, we built an *ensemble* of *group regressors*.

10.4 Analysis

As described in the previous section, we considered two set of features (f' and f'') and 3 regressors. We systematically assessed the effectiveness of all the 6 resulting combinations by means of a *leave-one-out* procedure applied on the training set, separately for each language. That is, for each language, type of feature, and regressor, (i) we built the subset T of the problem instances of the training with that language, (ii) we removed one element t_0 from T, (iii) we computed the feature values for the problem instances in T and trained the regressor, (iv) we applied the trained regressor to the problem instance t_0 and compared the generated answer against the known one. We repeated all but first steps $|T| = 100$ times, i.e., by removing each time a different element, and computed the performance of the method in terms of the indexes defined in Section 10.2: $c@1$ and AUC.

The results are in Table 10.1: the table shows $c@1$ and AUC values for each method, the method name being composed by the regressor acronym and one among abs or rel indicating the use of f' or f'' features, respectively. It can be seen that RF provides in general better results than the other regressors; moreover, RF-abs appears to be the best performing method. In order to further validate the latter finding, we performed a Wilcoxon signed-rank test [34] with a significance level of 5% and Bonferroni correction [38]: the outcome is that RF-abs is significantly better than all the other methods, except Tree-rel, for a little gap, and RF-rel; RF-rel is not better than the other methods except SVM-rel; Tree-rel is not better than all the other methods.

In order to gain insights about which features group appeared to be more suitable for accomplishing the considered task, we applied the RF-abs method (with the leave-one-out procedure described above) 9 times, each time removing one of the 9 features groups—i.e., we performed a features group ablation analysis. The results (in terms of $c@1$) are reported in Table 10.2. It can be seen, by comparing results of method RF-abs with those of Table 10.1, that the largest decrease of $c@1$ occurs by removing features group MG, while the smallest one occurs by removing features group WR—on the average around 3% and 1%, respectively. It can also be observed that feature ablation may actually lead to some improvements: for English, we obtain 0.69, rather than 0.67, by removing WG; for Spanish, we obtain 0.96, rather than 0.94, by removing either WG or WR.

Then, we analyzed the performance of RF-abs in terms of feature addition. We considered RF-abs using only features group MG (which showed to be the most relevant, according to the ablation analysis)

Features groups	EN	DU	GR	SP
All-WG	0.69	0.68	0.75	0.96
All-CG	0.66	0.71	0.75	0.95
All-PG	0.68	0.70	0.75	0.94
All-WL	0.67	0.71	0.75	0.95
All-SL	0.65	0.70	0.73	0.95
All-SG	0.66	0.69	0.75	0.95
All-WR	0.67	0.71	0.75	0.96
All-MG	0.62	0.71	0.74	0.94
All-TG	0.63	0.72	0.75	0.93

Table 10.2: $c@1$ with RF-abs by removing one features group at once.

Features groups	EN	DU	GR	SP
MG	0.71	0.63	0.66	0.89
MG+WG	0.67	0.71	0.75	0.94
MG+CG	0.73	0.63	0.68	0.93
MG+PG	0.71	0.67	0.68	0.94
MG+WL	0.72	0.65	0.66	0.91
MG+SL	0.73	0.65	0.72	0.90
MG+SG	0.73	0.58	0.71	0.87
MG+WR	0.59	0.56	0.60	0.74
MG+TG	0.72	0.64	0.68	0.91

Table 10.3: $c@1$ with RF-abs by using MG features and zero or one other features group.

and then using only MG and each of the 8 other features groups in isolation. The results are reported in Table 10.3. It can be seen that, for English, there are combinations that improve $c@1$ with respect to the baseline value 0.67: MG+CG, MG+SL, and MG+SG reach 0.73. Since such improvement is not negligible, we inspected the mutual effect of these features groups more closely by analyzing the $c@1$ values resulting from all their combinations. The results are: 0.78 with MG+CG+SL, 0.65 with CG+SG+SL, 0.71 with MG+SL+SG, and 0.73 with MG+CG+SL+SG. Based on these results, which improved the 0.67 baseline (all feature groups), we chose to use RF-abs with only 3 features groups (MG+CG+SL), only for the English language. On the other hand, we did not notice significant improvements for specific sets of features groups for the other languages: hence, for Dutch, Greek, and Spanish, we chose to use RF-abs with all the features groups.

We observed that the results for the Spanish language tend to be much better than for the other languages. We believe that such good results depend more on the peculiarity of this dataset rather than to the quality of our method: indeed the training set for Spanish contained 100 problem instances with 5 documents each, but the number of distinct documents, though, was only 42.

Method	Language	$c@1$	AUC	Score	Ranking
RF-abs on MG+CG+SL	EN	0.56	0.578	0.323	10/18
RF-abs on all	DU	0.69	0.751	0.518	4/17
RF-abs on all	GR	0.66	0.698	0.459	7/14
RF-abs on all	SP	0.83	0.932	0.773	1/17

Table 10.4: Final results.

10.4.1 Final results

Table 10.4 reports the final results obtained in the competition, as released by the organizers[5]. The table shows the performance indexes computed on a separated testing set which was not available during the method design phase. Besides $c@1$ and AUC, the table also reports a score, according to which a ranking for each language has been compiled: the score is the product of $c@1$ and AUC.

10.4.2 Remarks

In this chapter we have described the approach for the author identification that we submitted to the 2015 PAN competition [196]. Our machine learning performed very well in the PAN competition, obtaining the first position in the final rank for the Spanish language.

[5] http://www.tira.io/task/authorship-verification/

Chapter 11

An Author Profiling Approach Based on Language-dependent Content and Stylometric Features

11.1 Overview

In this chapter we describe the approach that we submitted to the 2015 PAN competition [171] for the author profiling task[1]. The task consists in predicting some attributes of an author analyzing a set of his/her Twitter tweets.

We consider several sets of stylometric and content features, and different decision algorithms: we use a different combination of features and decision algorithm for each language-attribute pair, hence treating it as an individual problem.

11.2 Problem statement

A problem instance consists of a tuple $\langle D, L \rangle$, where D is a set of tweets written by the same author and L is a value of enumerated type that describes the language of the tweets—English, Spanish, Italian, or Dutch.

The author profiling consists in generating, given a problem instance, the value for several attributes with respect to the author of the tweets: gender, age group (only for English and Spanish), and 5 personality traits. Age group is an enumerated value among the following: 18–24, 25–34, 35–49 or $\geqslant 50$. The 5 *personality traits* are widely accepted characteristics used to describe human personality (also known as Big Five [193]): extroversion, neuroticism, agreeableness, conscientiousness, and openness to experience. For each trait, the attribute value consists of a score in $[-0.5, +0.5]$.

A set of solved problem instances (the *training set*) is available in which, for each problem instance $\langle D, L \rangle$, the tuple of the attributes values is provided.

The effectiveness of a method for author profiling is assessed using a *testing set* of solved problem instances. In particular, the effectiveness is assessed separately for each attribute as follows: the attribute values generated by the method for the problem instances in the testing set are compared against the actual values and the comparison outcome is expressed in terms of accuracy for gender and age, and in terms of Root-mean-square error (RMSE) for the personality traits.

[1] During the competition we discovered several opportunities for fraudulently boosting the accuracy of our method during the evaluation phase. We described these opportunities in a report. We notified the organizers which promptly acknowledged the high relevance of our concerns and took measures to mitigate the corresponding vulnerabilities. The organizers acknowledged our contribution publicly. We submitted for evaluation an honestly developed method—the one described in this document—that did not exploit such unethical procedures in any way.

Language	Original	Merged
English	152	83
Spanish	100	50
Italian	38	19
Dutch	34	18

Table 11.1: Number of problem instances in the original training set and in the new training built by merging repetitions.

11.3 Our approach

We chose to handle the prediction of each attribute for each language as an individual problem: in particular, we consider gender and age group prediction as 2 classification tasks and personal traits prediction as 5 regression tasks. Since we had tweets written in four languages and we had to predict age groups for those written in English and Spanish only, we hence considered 26 different problems.

We propose a machine learning approach based on a number of different *stylometric* and *content* features which are processed by one among three different decision algorithms—we used SVM and random forests as classifiers and regressors. We carried out an extensive experimental campaign for systematically assessing a large number of the possible combinations, through *leave-one-out* cross validation on the available training data.

11.3.1 Training set analysis and repetitions

During preliminary analysis, we noticed that the training set included some subsets of problem instances for which L and the solution were the same, i.e., the attributes values for all the problem instances in a subset were the very same, despite being D different. We call *repetitions* those problem instances. We argued that the tweets of the problem instances in each of those subsets were authored by the same person. For this reason, we decided to build a new training set by replacing each of those subsets with a single problem instance in which D is the union of all the tweet sets of the subset—i.e., we merged the repetitions. Table 11.1 shows the sizes of the training set portions corresponding to each language before and after merging repetitions. We later experimentally verified that this transformation did affect the learned classifiers and regressors.

11.3.2 Features

The feature extraction procedure requires a language-dependent *dictionary* in which words are grouped according to their prevalent topic (e.g., "money", "sports", or "religion") or their function (e.g., "prepositions", "articles", or "negations"). To this end, we used an English dictionary similar to the one used by LIWC [164]. For the other 3 languages, we proceeded as follows. For Spanish and Dutch, we built the dictionary by automatically translating the English dictionary with Google Translate. For Italian, we manually built the dictionary, by using the English one as guideline. Moreover, for each language, we augmented the dictionary with a new category of words ("chat acronyms") containing the top fifty most popular chat acronyms exposed on NetLingo[2].

The feature extraction procedure is also based on the notion of *automatic tweet*, that we define as follows. We determined a set of ordered sequences of $n = 1, \ldots, 4$ words, that we call *templates*, based on an analysis of the full training set:

1. we automatically extracted from the full training set all tweets starting with the same ordered sequence of n words;

2. we automatically constructed a set including all word sequences that were the starting sequence of at least 3 different tweets;

[2]http://www.netlingo.com/top50/popular-text-terms.php

Template	EN	ES	IT	NL
`# Move más reciente`		✓		
`Photo:`	✓	✓	✓	
`I'm at`	✓	✓	✓	
`I liked a`	✓		✓	
`I favorited a`	✓	✓	✓	
`Ik vind een`	✓			✓
`#in`	✓		✓	
Total number of templates	29	8	12	1

Table 11.2: Some examples of templates and the languages for which at least one automatic tweet with that template were found. The first row corresponds to a template found only in Spanish problem instances, while the other rows are templates found in problem instances of multiple languages. The last row contains, for each language, the count of templates for which at least one automatic tweet with that template was found.

3. we manually analyzed each sequence and retained only those which appeared to be the beginning of an automatically-generated tweet.

We say that a tweet is an automatic tweet if its first words correspond to a template. Table 11.2 provides some examples of templates, along with the presence or absence of corresponding automatic tweets of different languages in the training sets.

The feature extraction procedure is as follows. Given a problem instance $\langle D, L \rangle$, we denote by D_M the set of tweets obtained by D by removing all the automatic tweets. We extract several numerical features from each problem instance: the value of all (except of 3) features is obtained by averaging the corresponding computation outcomes on the tweets in D or D_M—the remaining three feature values are computed on the whole D and/or D_M. For ease of presentation, we group conceptually similar features together; the full list is given in Table 11.3.

Stylometric These features tend to capture the structural properties of a tweet in a way largely independent of both the language and the specific semantic content; therefore, they are not based on the dictionaries. Stylometric features are computed on tweets in D_M: the reason is because we assume that automatic tweets are not really representative of the tweet writing style of the author.

Content These features are based on the dictionaries categories related to word topic and are computed on tweets in D: the reason is because we assume that the content of automatic tweets is indeed informative of the author profile.

Hybrid These features are based on the dictionaries categories related to word function and are computed on tweets in D_M.

11.3.3 Feature selection

Past studies on author profiling report several correlations between gender, age, personality traits and writing style. In particular, [181] showed that stylometric features are more predicitve than content features for determining the gender, and viceversa for the age group, but the combination of both stylometric and content features can offer better results. In [95], the authors provided a list of correlations between some LIWC and non-LIWC features and the five personality traits. We constructed 40 different feature groups based on this knowledge and we assessed each of the resulting feature groups as described in the next section.

11.3.4 Classifier and regressor

We decided to build a different model for each language-problem pair, for a total of 26, as described in Section 11.2. We explored the usage of SVM [56] and Random Forest [47] with different configurations,

	Feature name	Description
stylometric	allpunc	Number of `.,:;`
	commas	Number of `,`
	exclmar	Number of `!`
	questma	Number of `?`
	parenth	Number of parenthesis
	numbers	Number of numbers
	wocount	Number of words
	longwor	Number of words longer than 6 letters
	upcawor	Number of uppercase words
	carrret	Number of carriage returns (`\n`, `\r`, `\r\n`)
	atmenti	Number of `@` mentions
	extlink	Number of links
	hashtag	Number of `#`
	posemot	Number of positive emoticons
	negemot	Number of negative emoticons
	emotico	Number of emoticons
	emotiyn	Presence of emoticons in D (binary feature)
content	moneywo	Number of words in the "money" category
	jobword	Number of words in the "job or work" category
	sportwo	Number of words in the "sports" category
	televwo	Number of words in the "tv or movie" category
	sleepwo	Number of words in the "sleeping" category
	eatinwo	Number of words in the "eating" category
	sexuawo	Number of words in the "sexuality" category
	familwo	Number of words in the "family" category
	frienwo	Number of words in the "friends" category
	posemwo	Number of words in the "positive emotion" category
	negemwo	Number of words in the "negative emotion" category
	emotiwo	Number of words in the "positive emotion" or "negative emotion" category
	swearwo	Number of words in the "swear words" category
	affecwo	Number of words in the "affective process" category
	feeliwo	Number of words in the "feeling" category
	religwo	Number of words in the "religion" category
	schoowo	Number of words in the "school" category
	occupwo	Number of words in the "occupation" category
	autotwe	Automatic tweets ratio, i.e., $\frac{\lvert D \setminus D_M \rvert}{\lvert D \rvert}$
	autweyn	Presence of automatic tweets in D (binary feature)
hybrid	fsipron	Number of words in the "I" category
	fplpron	Number of words in the "we" category
	ssipron	Number of words in the "you" category
	selfref	Number of words in the "self" category
	negpart	Number of words in the "negations" category
	asspart	Number of words in the "assents" category
	article	Number of words in the "articles" category
	preposi	Number of words in the "prepositions" category
	pronoun	Number of words in the "pronoun" category
	slangwo	Number of words in the "chat acronyms" category

Table 11.3: Features list.

as these methods can be used both as classifiers and as regressors. In particular, we considered:

- *svm*: SVM with default gaussian kernel and $C = 1$;
- *rf500*: Random Forest with 500 trees;
- *rf2000*: Random Forest with 2000 trees.

11.4 Analysis

As described in the previous sections, we considered 40 sets of features and 3 classifiers/regressors. We systematically assessed the effectiveness of all the 120 resulting combinations by means of a *leave-one-out* procedure applied on the training set, separately for each language-attribute pair. That is, for each language-attribute pair, set of features, and classifier/regressor, (i) we built the subset T of the problem instances of the training set with that language, (ii) we removed one element t_0 from T, (iii) we computed the values for the features set on the problem instances in T and trained the classifier/regressor, (iv) we applied the trained classifier/regressor to the problem instance t_0 and compared the generated answer against the known one. We repeated all but first steps $|T|$ times, i.e., by removing each time a different element, and computed the performance of the method in terms of the indexes defined in Section 11.2. Finally, we chose, for each language-attribute pair, the best performing combination, in terms of accuracy or RMSE, as appropriate for that attribute. The resulting configurations are summarized in Table 11.4.

In order to provide a synthetic baseline, we built 3 baseline methods using each of the 3 classifiers/regressors with all the features. The results, obtained by means of the same leave-one-out procedure, are shown in Table 11.5.

It can be seen from Table 11.4 that our procedure lead us to chose a different configuration of classifier/regressor and features set for each language-attribute pair. There could be several reason to explain that. First, every language has its own writing rules and culture, so it is possible that a middle aged English man could not have the same interests and the same writing style of a middle aged Italian man. Second, the Spanish, Dutch, and Italian dictionaries we used were not as good as the LIWC English one. Finally, the number of problem instances in the training set was not the same for every language, and so was the number of tweets in the instances within each language subset.

11.5 Remarks

In this chapter we have described our machine learning method for author identification that is based on SVM and random forest as classifiers and regressors. We submitted this method to the 2015 PAN competition for the author profiling task.

L	Attribute	Class./Regr.	Chosen features set
EN	Gen	rf2000	commas negemot exclmar
	Age	rf2000	allpunc commas exclmar questma parenth numbers wocount longwor upcawor carrret atmenti extlink hashtag posemot negemot emotico autotwe
	Ext	svm	wocount questma parenth familwo
	Neu	svm	selfref fsipron chatacr affecwo emotiwo hashtag posemot pronoun wocount
	Con	rf500	extlink longwor numbers hashtag fsipron selfref
	Agr	svm	questma atmenti allpunc ssipron article longwor jobword chatacr
	Ope	rf2000	commas extlink hashtag exclmar questmar parenth wocount ssipron negpart article feeliwo moneywo jobword eatinwo familwo negemwo religwo
ES	Gen	svm	allpunc commas exclmar questma parenth numbers wocount longwor upcawor carrret atmenti extlink hashtag posemot negemot fsipron fplpron ssipron selfref negpart asspart article preposi pronoun slangwo moneywo jobword sportwo televwo sleepwo eatinwo sexuawo familwo frienwo posemwo negemwo affecwo feeliwo
	Age	svm	extlink hashtag numbers sleepwo sexuawo
	Ext	rf2000	longwor carrret questma preposi autweyn emotico
	Neu	rf2000	posemot ssipron exclmar selfref extlink
	Con	rf500	extlink longwor numbers hashtag fsipron selfref affecwo emotiwo
	Agr	svm	allpunc commas exclmar questma parenth numbers wocount longwor upcawor carrret atmenti extlink hashtag posemot negemot + fsipron fplpron ssipron selfref negpart asspart article preposi pronoun slangwo moneywo jobword sportwo televwo sleepwo eatinwo sexuawo familwo frienwo posemwo negemwo swearwo religwo
	Ope	rf2000	autotwe hashtag preposi wocount religwo
IT	Gen	rf500	asspart fsipron selfref exclmar extlink hashtag emotiyn
	Ext	svm	allpunc wocount hashtag questma
	Neu	rf2000	commas longwor fplpron chatacr autweyn
	Con	svm	commas extlink hashtag exclmar questmar parenth wocount ssipron negpart article feeliwo moneywo jobword eatinwo familwo negemwo religwo
	Agr	svm	posemot exclmar moneywo hashtag pronoun autweyn
	Ope	svm	negpart hashtag atmenti exclmar longwor
NL	Gen	rf2000	negemot upcawor preposi
	Ext	svm	questma atmenti allpunc ssipron article longwor jobword chatacr extlink autweyn
	Neu	rf2000	atmenti preposi longwor emotiyn
	Con	svm	hashtag questma exclmar atmenti posemot wocount extlink longwor
	Agr	svm	atmenti commas exclmar hashtag autweyn emotiyn
	Ope	svm	negpart hashtag atmenti exclmar longwor

Table 11.4: Chosen classifier/regressor and features set for each language-attribute pair.

		Baselines			
L	Attribute	svm	rf500	rf2000	Our conf.
EN	Gen	0.566	0.619	0.619	0.735
	Age	0.614	0.617	0.605	0.692
	Ext	0.185	0.182	0.181	0.165
	Neu	0.243	0.226	0.226	0.208
	Con	0.167	0.158	0.158	0.146
	Agr	0.173	0.183	0.183	0.162
	Ope	0.157	0.149	0.149	0.143
ES	Gen	0.760	0.760	0.760	0.820
	Age	0.400	0.404	0.416	0.580
	Ext	0.185	0.177	0.176	0.156
	Neu	0.243	0.220	0.220	0.202
	Con	0.161	0.163	0.162	0.154
	Agr	0.162	0.169	0.169	0.157
	Ope	0.183	0.183	0.183	0.168
IT	Gen	0.632	0.705	0.737	0.853
	Ext	0.159	0.162	0.162	0.121
	Neu	0.202	0.215	0.215	0.170
	Con	0.126	0.135	0.136	0.113
	Agr	0.159	0.165	0.165	0.150
	Ope	0.186	0.178	0.177	0.102
NL	Gen	0.611	0.344	0.333	0.633
	Ext	0.131	0.140	0.139	0.105
	Neu	0.206	0.205	0.204	0.156
	Con	0.122	0.125	0.125	0.101
	Agr	0.163	0.161	0.162	0.130
	Ope	0.121	0.122	0.122	0.104

Table 11.5: Results of our configuration and the synthetic baselines. Accuracy is reported for Gen and Age, RMSE is reported for Ext, Neu, Con, Agr, and Ope.

Chapter 12

Automatic Generation of Scientific Paper Reviews

12.1 Overview

Peer review, i.e., the process of subjecting a work to the scrutiny of experts in order to determine whether the work deserves publication, is a keystone in scholarly publishing. The review process should ensure that a published paper is of high scientific quality, which in its turn preserves the reputation of the corresponding publishing venue and improves the prestige of its author. On the other hand, peer review is just a piece of broader process involving several entities whose incentives may or may not actually drive the overall process toward those ideal goals. Authors are increasingly subject to strong pressures in the form of research evaluation procedures in which the indicators that play a key role are often mostly numerical [32]. Reviewers tend to be overworked and often receive little credit for their hard work [67], while at the same time being interested in increasing some counter of program committees or editorial boards in which they are involved. Commercial publishers may find in scholarly publishing excellent opportunities for profit [107], even in the form of journals with little or no scrutiny: a periodically updated list of *predatory publishers* has grown by 50 times in the last 5 years, including 923 publishers in its latest release [35].

While there is no doubt that most published research follows a rigorous and honest path, it is evident that actors involved in research may now find ways to maximize their personal benefits disregarding the ideal objective of the scientific environment as a whole, by following practices that are questionable or simply fraudulent [43, 68]. Unfortunately, this claim is not a mere theoretical possibility. Questionable operators have emerged that run bogus journals and conferences which have no other purpose than generating profit while uttering worthless scientific literature [51]. Supposedly peer-reviewed journals accept for publication papers that have been randomly generated [80] or publish papers which clearly have not been proof-read by anyone [162]. Misbehaving researchers attempt to inflate their records by ghostwriting papers on nonexistent research [170]. Not surprisingly, the critical reviewing step has been exploited as well. Computer intrusions on the editorial system of a major commercial publisher have forced the publisher to retract several published papers [173]. In the last few years, hundreds of published papers have been retracted by several commercial publishers in many independent events [53, 83, 88], due to the discovery of reviews fabricated by the authors themselves which provided journals with suggested reviewers along with fake contact information which actually routed communication to the authors or their colleagues.

In this chapter, we investigate the feasibility of more fraud opportunities in the form of a procedure for *automatic generation of fake reviews*. We propose a method for generating automatically text which (a) looks like the typical scientific paper review, (b) is tailored to the specific paper being reviewed, and (c) conveys a recommendation specified as input. A tool that is capable of generating fake reviews systematically and at *no cost* may be misused in several ways. Busy people which want to be involved in

as many reviewing committees as possible might choose a recommendation and then generate reviews very quickly, perhaps without even reading the paper or after just a superficial look. Predatory publishers might attempt to improve their credibility by sending many reviews to authors. Of course, reviews generated by our tool will certainly be detected as being fake by any decent editorial process. On the other hand, as pointed out above, perverse incentives and unethical conducts might find a role for a tool of this kind, which may potentially magnify the scale of frauds in the reviewing process in several ways. In this respect, it is important to keep in mind that a few years ago Springer and IEEE retracted more than 120 published papers which were computer-generated nonsense [156]. Our proposed tool could find more constructive applications, though. For example, the steering committee of a conference could inject fake reviews in the discussion phase without informing the program committee and then observe the outcome.

Our proposed method constructs a review tailored to a specific paper, with a specified recommendation, based solely on the paper text and a corpus of reviews written by humans for other papers. A key aspect of our proposal is that it builds upon a relatively small knowledge base (some tens of reviews) while commonly used methods for text generation, such as Artificial Neural Networks (ANN), typically require a very large amount of data in order to build an effective generative model. Applying those methods in the context of scientific review generation is difficult because of the difficulty in finding a large amount of samples of scientific reviews, in particular, of negative reviews.

An important contribution of our work is the experimental campaign performed involving human subjects. We performed an *intrinsic* evaluation aimed at assessing the ability of our method to generate reviews which look like as being written by a real human reviewer. Moreover, we performed an *extrinsic* evaluation aimed at assessing the impact on the decision about accepting or rejecting a paper under review. Although our experimental campaign is not a replica of a real editorial process and thus may provide only a preliminary assessment, our results do provide interesting insights.

We like to emphasize that the scientific article [28] that describes this work has also attracted much interest on magazines[1].

12.2 Related work

To the best of our knowledge, no method for the automatic generation of reviews of scientific papers has been proposed before. From a broader point of view, our proposal is a form of Natural Language Generation (NLG), which is widely used in many different fields such as spoken dialogue systems [206], machine translation [198], and as a mean for creating editorial content by turning structured data into prose [208].

A notable use of NLG for scientific purpose, which is particularly relevant to our work, is the software SCIgen[2]. This tool generates pdf files consisting of syntactically correct random text which is formatted like a scientific publication, including randomly generated figures, plots, and code fragments. Later and independently from its creators, SCIgen has been used in order to test the submissions standard of conferences and to prove that nonsense papers may actually be published, even by respected publishers [156]. This phenomenon has been investigated also in [124], which studies the spread of fakes and duplicates through notable publishers. The fact that a tool which was born as a "toy" for Computer Science researchers led to actual malicious behaviors suggests that other types of cheating may arise, including the creation of false reviews: this consideration is indeed the main motivation of our work.

In the work described in this chapter, we propose a *corpus-based* NLG method. Corpus-based methods aim at training text generation rules automatically from text examples of the desired text generator output. An example of corpus-based method applied to text generation in dialogue is the work in [159]. The cited work proposes a class-based n-gram language model (LM) that improves over template-based and rule-

[1] https://www.timeshighereducation.com/news/robot-written-reviews-fool-academics, http://retractionwatch.com/2016/09/02/weve-seen-computer-generated-fake-papers-get-published-now-we-have-computer-generated-fake-peer-reviews/,http://www.powerlineblog.com/archives/2016/09/academic-absurdity-of-the-week-fake-peer-reviews.php?,https://www.insidehighered.com/news/2016/09/22/many-academics-are-fooled-robot-written-peer-reviews

[2] http://pdos.csail.mit.edu/scigen/

based text generation systems. Belz [37] proposes a corpus-based probabilistic generation methodology and apply it to the automatic generation of weather forecast texts. The work in [174] assesses a new model for NLG in dialogue systems by maximizing the expected reward using reinforcement learning.

A different approach to NLG is based on Artificial Neural Networks (ANN). Kukich [121] implemented a stock reporter system where text generation is done at phrase level using an ANN-based approach. A recent work demonstrated the effectiveness of Recurrent Neural Networks (RNN) for natural language generation at character level [149]. A variant of RNN, Long Short-Term Memory (LSTM) [111], proved its ability to generate characters sequences with long-range structure [97]. The authors of [167] showed the ability of a LSTM framework to automatically generate rap lyrics tailored to the style of a given rapper. Zhang and Lapta [219] proposed an RNN-based work for generating Chinese poetry. Beyond unbounded text generation, LSTM for NLG has also been used in the generation of image descriptions [115, 143, 203] and in the generation of descriptive captions for video sequences [202].

All the generative methods based on neural networks require a huge amount of learning data, usually orders of magnitude more than the amount of data that we could find in our scenario (i.e., scientific reviews). Methods for *data augmentation* capable of decreasing the amount of learning data required for training a neural network effectively certainly deserve investigation in our context [57].

12.3 Our approach

The problem consists in generating, given a paper a and an overall recommendation $o \in \{\text{accept}, \text{neutral}, \text{reject}\}$, a review r which (i) appears as generated by a human (ii) for the paper a and (iii) which expresses a recommendation o for a. In our work, we assume that the paper a is a plain text which consists of the concatenation of the paper title, abstract and main content.

Our method requires a set R of real paper reviews, i.e., each review $r \in R$ has been written by humans. We pre-process each review in R as follows: (i) we split the document in a sequence $\{t_1, t_2, \dots\}$ of tokens according to the Penn-Treebank procedure; (ii) we execute a *Named-entity Recognition* (NER)[3] [87] on the token sequence; and (iii) we execute a *Part-of-Speech* (POS) annotation[4] [200] on the token sequence; finally (iv) we classify each token in $\{t_1, t_2, \dots\}$ as being or not being a specific term, according to an heuristic procedure (see below).

When generating a review for a paper a with a specified recommendation o, our method performs 3 steps, described below in full detail: (i) it builds a set S of sentences from reviews in R and replaces each specific term in each sentence with a specific term of a; (ii) it removes from S the sentences which express a sentiment which is not consistent with o; (iii) it reorders and concatenates the sentences in S obtaining a review for a.

Specific terms identification. With this procedure, we aim at identify the *specific terms* of a document d—i.e., those terms which are relevant to d. To this end, we defined a simple heuristic. Let $\{t_1, t_2, \dots\}$ the sequence of tokens for d, where each token has been annotated with NER and POS taggers. A token $t \in \{t_1, t_2, \dots\}$ is a specific term if it meets all the following criteria: (i) t has been annotated as a noun (`NN`) or as an adjective (`JJ`); (ii) the length in characters of t is at least 2; (iii) t contains at least one letter.

Specific terms replacement. In this step, we aim at constructing a set S of review sentences tailored to a. To this end, we proceed as follows, starting with $S = \varnothing$. For each review $r \in R$, we split the review in a set S_r of sentences. We obtain (according to the procedure described above) the set W'_a of specific terms of a, retrieve the set W'_r of specific terms of r, and set $W_a = W'_a \backslash W'_r$ and $W_r = W'_r \backslash W'_a$. Then, for each sentence $s_r \in S_r$, we generate a random mapping from items in the set W^s_r of specific terms of W_r which occur in s_r to items in W_a such that: (a) each item in W^s_r is mapped to exactly one item in

[3] http://nlp.stanford.edu/software/CRF-NER.shtml
[4] http://nlp.stanford.edu/software/tagger.shtml

W_a, (b) no items in W_r^s exist such that they are mapped to the same item in W_a, and (c) for each item w_r^s mapped to an item w_a, the POS and NER annotations of w_r^s are the same of respective annotations of w_a. If such mapping is possible, we replace each occurrence of a term of W_r^s in s_r with the mapped term in W_a and add the modified sentence to S; otherwise, we proceed to the next sentence.

In other words, after this procedure, S contains all the suitable sentences generated by iterating the term replacement procedure for all the reviews in R.

Sentiment analysis. In this step, we aim at selecting the sentences of S which express a sentiment consistent with the specified overall recommendation o. To this end, we apply a pre-trained Naive Bayes sentiment classifier[5] [154] to each sentence $s \in S$, basing on the assumption that a positive sentiment can be associated with an accept recommendation, a negative sentiment with a reject recommendation, and a neutral sentiment with a neutral recommendation.

After the application of the sentiment classifier, we retain in S only the sentences for which the outcome is consistent with o.

Sentences reordering. In this step, we aim at generating the final output of our method (the automatically generated review) by selecting, reordering, and concatenating a subset of sentences of S. The rationale for the selection and reordering is to obtain a review (a) whose length is realistic, w.r.t. a typical review, and (b) which has an overall structure which resembles a typical review—e.g., an opening sentence, some considerations, a conclusive remark.

Concerning the reordering, we based on the assumption that sentences may be classified as suitable for opening part, central content, and closing part. Accordingly, we built a classifier which takes as input a single sentence and outputs a label in $\{\text{opening}, \text{central}, \text{closing}\}$. We took the general purpose text classifier based on maximum entropy[6] described in [142] and trained it using all the sentences of the reviews in R, which we automatically labeled as follows: if the sentence was the first sentence in its review, we associated it with the label opening; otherwise, if it was the last sentence, we associated it with closing; otherwise, we associated it with central.

When generating a review, we apply the classifier to each sentence in S and then randomly select 1 opening sentence, 3 central sentences, and 1 closing sentence. Finally, we concatenate those 5 sentences and obtain the review for a.

12.4 Experimental evaluation

We performed two experimental evaluations involving human subjects for assessing our proposed method ability to generate reviews which (a) look like as they have been written by real human reviewers for the specified paper, and (b) can affect the decision about accepting or rejecting the specified paper. That is, we performed an intrinsic evaluation and an extrinsic evaluation, respectively.

We built a dataset composed of 48 papers and 168 reviews, which we obtained from the F1000Research, Elifescience, Openreview and PeerJ web sites—which publish reviews of accepted papers along with corresponding full texts—and from our lab publication records; we used the reviews of the dataset as the set R while running our method. Moreover, for the purpose of performing our evaluations, we associated an overall recommendation (i.e., a label in $\{\text{accept}, \text{neutral}, \text{reject}\}$ with each review in the dataset. Since the sources we considered vary in the way, if any, they classify reviews according to overall recommendation, we proceeded as follows. If a review was explicitly associated with an overall recommendation by its author, we associated it with the suitable label—e.g., positive recommendations to accept, negative recommendations to reject, and all the other recommendations to neutral. Otherwise, if a review was not explicitly associated with an overall recommendation, we considered the outcome of the publishing process which, for published papers, was always acceptance.

[5] http://sentiment.vivekn.com

[6] http://nlp.stanford.edu/software/classifier.shtml

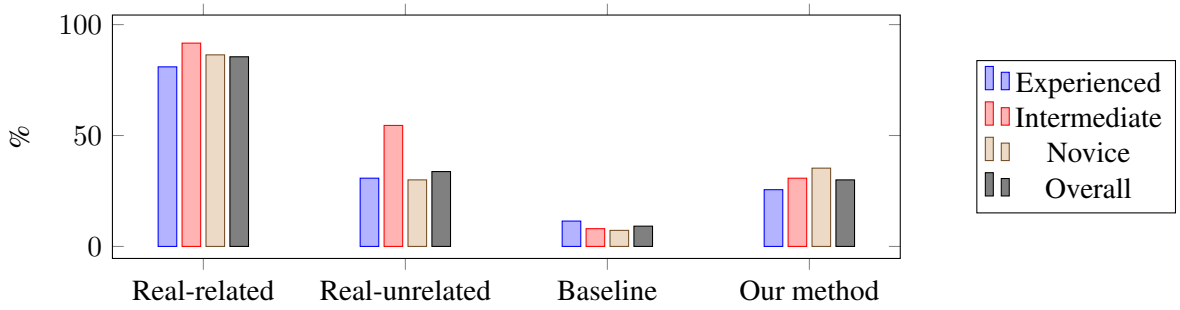


Figure 12.1: Percentage of reviews considered as written by a human for the specified paper.

In order to provide a comparison baseline for our review generation method, we designed and built a simple baseline generation method based on Markov chains. To this end, we trained a second order Markov chain, operating on tokens, on all the reviews in the dataset: before the training, we added a special token t_{end} at the end of each review. When generating a review with the baseline method, the specified paper a and the overall recommendation o are not considered and the following steps are performed. First, a review in the dataset is randomly chosen and its first two tokens are fed into the Markov chain generative model. Then, the generative model is run until the token t_{end} is obtained. Finally, the output is obtained by concatenating all the generated tokens.

In our experimentation, we involved a number of human subjects, who were asked to examine the generated reviews and then to answer some questions. In order to gain more insights about our method effectiveness, we grouped the subjects according to their presumed familiarity with scholarly publishing, resulting in 3 classes. The experienced class is composed of professors, PhD student, and postdocs; the intermediate class is composed of undergraduate students; the novice class is composed of all the remaining subjects (who were anyway sufficiently proficient with English).

12.4.1 Intrinsic evaluation

In the intrinsic evaluation, we built a number of forms, each showing the title of a paper a randomly chosen from our dataset and a set of 10 reviews randomly sampled for the following sets: (a) the real reviews in the dataset actually related to a, (b) the real reviews in the dataset not related to a, (c) a set of reviews generated using the baseline method, and (d) a set of reviews generated using our method with a and a random overall recommendation o as input. Since the size in characters of the real reviews can widely vary, we limited the number of sentences presented to the subject to 5, as for our generated reviews, randomly sampled from the corresponding reviews while maintaining the original ordering. We asked the subject to say, for each review in the form, if "it appeared as a genuine review written by a human reviewer for the paper with the shown title". We gathered results from 16 subjects—5 novice, 3 intermediate, and 8 experienced.

Figure 12.1 shows the key findings of the intrinsic evaluation: the figure plots the percentage of positive answers (on the y axis) to the form questions for each kind of review (bar group) and for each class of subjects (bar fill pattern). It can be seen that our method generates reviews that are considered as written by a human in almost one case on three—the figure being greater for novice subjects an smaller for experienced subjects. Moreover, the deceiving ability is larger than the baseline: approximately 30% vs. 10%. Concerning the real reviews, Figure 12.1 shows that, as expected, they are properly recognized $\approx 85\%$ of the times: this finding suggests that the truncation of real reviews does not severely affect their appearance.

12.4.2 Extrinsic evaluation

In the extrinsic evaluation, we built a number of forms, each showing the title of a paper a randomly chosen from our dataset and a set of 3 reviews randomly sampled for the sets described at points a, b, and d in the previous section. Real reviews were possibly limited in length as in the intrinsic evaluation.

The form also showed, next to each review, the corresponding overall recommendation. We asked the subject to answer the following two questions: 1. "basing on these 3 reviews, would you recommend to accept or reject the paper?"; 2. "while taking your decision, in which order the 3 reviews influenced you?" We gathered results from 13 subjects—3 novice, 3 intermediate, and 7 experienced.

Table 13.4 summarizes the key findings of the extrinsic evaluation. In the left portion the table shows, for each subject class and for all the subjects, the number of forms in which at least a real and a generated reviews were discordant w.r.t. the recommendation (Discordant column), the number of discordant forms for which the subject took a decision in line with the generated reviews (and hence against the real reviews, Subverted column), and the ratio among Subverted and Discordant. In the right portion it shows the number of forms, for each kind of reviews, in which a review of the corresponding type were stated to be the most influencing by the subject; moreover it shows the percentage of forms in which the generated reviews were stated to be the most influencing.

Subject class	Subverted	Discordant	%	Our method	Original	Others	%
Experienced	4	16	25.0	10	21	4	28.6
Intermediate	4	15	26.7	11	18	14	25.6
Novice	5	21	23.8	11	25	9	24.4
Overall	13	52	25.0	32	64	27	26.0

Table 12.1: Results of the extrinsic evaluation (see text).

The most interesting, and somewhat surprising, finding is that in the 25% of cases the decision of an experienced subject agreed with the generated reviews and disagreed with the real reviews: from another point of view, through a generated review we were able to manipulate the outcome of the (simulated) peer review process. Table 13.4 also shows that, in 26% of cases, a generated review was stated to be the most influencing by the subjects.

12.5 Remarks

In this chapter we proposed a method for the automatic generation of scientific reviews. The method is able to generate a review of a given research paper with a specified overall recommendation. To this end, it performs multiple steps aimed at generating reviews which resemble human written reviews and hence might potentially induce the reader to accept or reject the reviewed paper.

A key contribution of our work is the experimental evaluation, which involved 16 human subjects. The results show that in $\approx 30\%$ of cases a generated review is considered genuine by the human subjects; moreover, in about 1 among 4 cases, we were able to manipulate the outcome of a (simulated) peer review process through generated reviews which we mixed with genuine reviews.

Beyond these promising results, our proposal needs further investigation and, in this respect, we plan to compare it with other NLG methods, such as ANN, for which, however, a much larger amount of data need to be collected. Finally, it could be interesting to investigate if and how an ontology can improve the review generation process.

Chapter 13

Automatic Generation of Restaurant Reviews with LSTM-RNN

13.1 Overview

Online product reviews play a crucial role in both the electronic and conventional commerce [125]. Many websites and user forums allow online communities to share their experience about products, touristic destinations, cultural offerings, and so on. Such information may be very useful to both users interested in a certain item and sellers interested in increasing their revenue. Since users tend to trust the opinion of other users, online reviews strongly influence decisions.

In this scenario, the opinion of a user can be biased by malicious sellers who try to gain unfair competitive advantages for their products, by disseminating either fake positive reviews for their products, or fake negative reviews for the products of their competitors. This phenomenon, called *opinion spamming*, is well known by web-oriented business companies which forbid or strongly discourage such practice. Despite being forbidden, the economic returns potentially involved in committing review fraud can be so high to motivate users in devoting time and resources for praising or discrediting a specific target. It is clear that a tool capable of automatically generating a large number of false and diverse reviews with the desired bias may be potentially disruptive, as it might allow manipulating the opinions of consumers on a large scale. Although the services hosting product reviews do apply filters and procedures aimed at limiting the proliferation of false reviews, an attacker able to generate thousands of fake reviews quickly and cheaply could be able to generate a sufficient amount of reviews which slip through the sanity checks. Such reviews could suffice to manipulate the opinion of at least a fraction of the interested users and, more broadly, could undermine the confidence in the overall ecosystem of online product reviews. In this work, we aim at investigating the feasibility of a tool of this sort. We focus only on the actual content of a review. Systems which attempt to identify non genuine reviews usually consider also ancillary information such as, e.g., number and temporal distribution of reviews submitted from the same user or IP address. These features are beyond the scope of this work.

The contribution of our work, that is illustrated in this chapter, is two-fold: (i) we propose a method for generating a review, given a restaurant category and a rating; (ii) we perform an experimental campaign involving human users in which we evaluate the impact of our automatically generated deceptive reviews when mixed with genuine reviews.

Our method is based on a *Long Short-Term Memory based Recurrent Neural Network (LSTM-RNN)*. We train the network with a set of genuine reviews in order to obtain a tool capable of generating text which looks like a restaurant review. Then, in order to tailor the review to the desired rating and category, we use a set of classifiers (also previously trained with genuine reviews) in order to pick from the text generated by the network only the portions which matches the desired rating and category.

The experimental campaign is performed on a cohort of 39 users, who were not aware of the fact that they were dealing with automatically-generated reviews. We performed an *extrinsic* evaluation aimed

at assessing the impact on the decision about whether to go to a specific restaurant, and an *intrinsic* evaluation aimed at assessing the ability of generating a review which looks like as a review generated by an human author.

13.2 Related work

Methods for *Natural Language Generation* (NLG) are widely used in spoken dialogue systems [206], machine translation [198], and image caption generation [115]. We are not aware of any proposal for automatic generation of product reviews.

Artificial Neural Networks (ANN) are largely used in the field of NLG. The first ANN-based approach to NLG is the system presented in [121], which implements a stock reporter system where text generation is done at phrase level. A recent work [149] has shown the effectiveness of *Recurrent Neural Networks* (RNN) for NLG at character level. A key aspect of character-level generation with RNN is the ability of these models to autonomously learn grammatical and punctuation rules—e.g., opening and closing parentheses. Furthermore, character-level RNN tend to be more efficient than word-level RNN in terms of computational cost, which grows with the size of the input and output dictionaries. The works [150, 197] show that character-level RNN provide slightly worse performance than the equivalent word-based model, but the character-level approach allows to prediction and generation of new words and strings.

Long Short-term Memory (LSTM) networks [93, 111] are a form of RNN which has proven able to effectively generate characters sequences with long-range structures [97]. The work [198] bases on character-level LSTM RRN for machine translation tasks and proves their superiority over other statistical approaches.

An interesting NLG application of RNN is abstractive summarization [178], where the system produces a condensed representation of an input text that maintains its original meaning. The work in [218] employs RNN for question answering, with the NLG system producing correct answers to questions expressed in natural language. The work [216] provides a conversation system—a generator—for more fluent responses as part of a conversation.

A remarkable use of LSTM for NLG has been done in the generation of image descriptions [115, 143, 203] and in the generation of descriptive captions for video sequences [202]. Concerning the text generation for artistic purpose, Zhang and Lapta [219] proposed an RNN-based work for generating Chinese poetry. In [167], the authors show the ability of a LSTM framework to automatically generate rap lyrics tailored to the style of a given rapper.

13.3 Our approach

A restaurant is associated with a possibly empty set $C \subset \mathcal{C}$ of *categories*, with $\mathcal{C} = \{\text{italian}, \text{pub}, \text{spanish}, \dots\}$ (Table 13.2 shows the list of all categories in $\mathcal{C}$). A review for a restaurant is associated with a *rating* $s \in \{1, 2, 3, 4, 5\}$. We address the problem of automatically generating a review with a specified rating s for a restaurant with a specified set C of categories. The generated review should look like a review written by a human author and should be tailored to the rating and categories specified as input.

Our method is based on 3 steps: (i) a generative phase based on a LSTM character-level recurrent neural network, (ii) a category classification phase and (iii) a rating classification phase.

In the step i), we use LSTM RNN for generating text as follows. We first train the network (see next section) to predict the probability of the next *token* for a fixed-length sequence of tokens given as input—a token being a single character. Then, we generate text with the trained network by starting, as input, with an input sentence selected from a dataset of genuine reviews at random. We stochastically sample a token from the output of the network and append to the starting sequence. We then set the next input sequence for the network by shifting the starting sequence of one token, that is, we exclude the first token and we include the newly generated. We repeat this iterative procedure until a predefined number of reviews has been generated. We detect this condition by counting the number of occurrence of a special token t_{end},

Rating	# of reviews
1	111 218
2	111 833
3	245 896
4	671 610
5	1 028 707

Table 13.1: Number of reviews grouped by rating.

Category	# reviews
french	346 175
spanish	334 429
italian	310 816
american	306 499
pizza	239 311
mediterranean	238 141
british	224 719
asian	178 640
pub	169 564
european	169 362

Table 13.2: Number of reviews grouped by the 10 most frequent categories.

which we included at the end of each genuine review in the training text. We remove the first review from each set of generated reviews in order to neutralize any strong dependence from the starting sentence.

Concerning step ii, given the set of R reviews generated by the network, we use a set of binary classifiers to remove from R those reviews that are not coherent with the categories C specified as input. To this end, we input all reviews in R to a set of a 10 binary Naive Bayes classifiers, one for each category (we chose to consider only the 10 most frequent categories in our dataset). These classifiers were previously trained with genuine reviews (see next section) and use frequencies of 1-grams, 2-grams and 3-grams as features. We discard from R those reviews deemed to belong to less than half of the categories in C.

Finally, concerning step iii, we assign to each review in R a rating s' between 1 and 5 using a previously trained multi-class classifier. This classifier is Naive Bayes and uses the same features as those in the previous step. We remove from R all the reviews for which $|s - s'| > 1$ and pick one element from R at random as output of the procedure. If, at any point, $R = \varnothing$ then the overall procedure is aborted and restarted.

13.4 Experimental evaluation

We collected a dataset composed of 2 169 264 reviews distributed over 66 700 restaurants. Table 13.1 shows the number of reviews for each rating while Table 13.2 reports the number of reviews for the 10 most frequent categories.

We used a LSTM-RNN implementation based on the char-rnn library[1], configured with 3 layers composed of 700 neurons each—as suggested by the library authors. We trained the LSTM-RNN with a randomly chosen subset of the full dataset, composed of 500 000 reviews with 100 000 reviews for each rating. The training phase lasted about 1 month on a Intel Xeon E5-2440 (2.40 GHz) CPU equipped with 32 GB of RAM. Once trained, the time spent by the neural network to generate a review is in the order of

[1] https://github.com/karpathy/char-rnn

	Useful		Not useful	
	#	%	#	%
Genuine	138	80	35	20
Artificial	51	29	127	71

Table 13.3: Number and percentage of reviews considered as useful or not useful by human users (question a of extrinsic evaluation).

seconds.

The category and rating classifiers are based on the Naive Base implementation of the Stanford Classifier[2]. We trained each category classifier with $100\,000$ reviews and the rating classifier with $500\,000$ elements, all randomly selected from the dataset. Training time of classifiers was negligible with respect to the training time of LSTM-RNN.

In order to assess the effectiveness of our proposal we performed two different evaluations with human users, an *extrinsic* evaluation and an *intrinsic* evaluation, illustrated below. The evaluations were executed by presenting to each user a suite of *forms*, each including a set of reviews and few questions to be answered. This activity was carried out in our laboratory: it is important to remark that users were *not* aware that some revisions were artificially generated.

13.4.1 Extrinsic evaluation

In the extrinsic evaluation we assessed the ability of a review generated with our method to influence the decision of a user about whether to go or not to go to the reviewed restaurant. To this end, we constructed a set of forms, each composed of the name of a restaurant, its categories and 3 reviews. Reviews were randomly picked from a set R_g containing *genuine reviews* written by humans for that restaurant and from a set R_a of *artificial reviews*, using our method with inputs given by the categories of the restaurant and a random rating. We make sure that each form included at least one genuine review and one artificial review. In each form, we asked the user (a) for each review, if it was useful for his decision, and (b) if, basing on the 3 reviews, he would have decided to go to that restaurant. We proposed 3 forms to each user and we collected the evaluations of 39 different users.

The results of the extrinsic evaluation concerning question a are shown in Table 13.3 which reports the number of reviews marked by users as useful, separately for genuine and artificial. The key, and somewhat surprising, result shown in Table 13.3 is that around 30% of the artificial reviews (generated using our method) are indeed considered by a human users as useful for their decision.

Concerning question b, we categorized the forms submitted to users as follows. For each form, we computed the mean genuine rating $\overline{s}_g$ and the mean artificial rating $\overline{s}_a$ of the ratings associated with the genuine and artificial review, respectively. Next, we partitioned the 117 collected forms in 4 partitions, according to whether each of the two mean ratings was (denoted by $\mathcal{P}$) or was not (denoted by $\mathcal{N}$) $\geqslant 3$: two partitions include forms for which genuine and artificial reviews agree (denoted as $\mathcal{PP}$ and $\mathcal{NN}$), two include forms for which genuine and artificial reviews disagree (denoted as $\mathcal{PN}$ and $\mathcal{NP}$)—the two symbols concern $\overline{s}_g$ and $\overline{s}_a$, respectively. We were particularly interested in users answer to question b in the cases in which $\overline{s}_g$ and $\overline{s}_a$ disagree, i.e., in partitions $\mathcal{PN}$ and $\mathcal{NP}$. Table 13.4 shows the results concerning question b for those two partitions. It can be seen that the answer of the user conflicts with the polarity of genuine reviews in 29% of the cases for $\mathcal{PN}$ (genuine reviews are positive) and in 24% of the cases for $\mathcal{NP}$ (artificial reviews are negative).

A different point of view about this finding is given by Figure 13.1, which has one point for each form. The x and y coordinates of each point are the sums s_g^{tot} and s_a^{tot} of the ratings for the genuine and artificial reviews in the form, respectively. A circle represents a negative answer (not going) while a cross represents a positive answer (going). In a scenario in which the user decision conflicts with genuine

[2]http://nlp.stanford.edu/software/classifier.shtml

Genuine	Artificial	Going #	Going %	Not going #	Not going %
$\mathcal{P}$	$\mathcal{P}$	21	47	23	53
$\mathcal{P}$	$\mathcal{N}$	10	71	4	29
$\mathcal{N}$	$\mathcal{P}$	9	24	28	76
$\mathcal{N}$	$\mathcal{N}$	5	23	17	77

Table 13.4: Number and percentage of forms resulting in decision to go or not to go to a restaurant (question b of extrinsic evaluation).

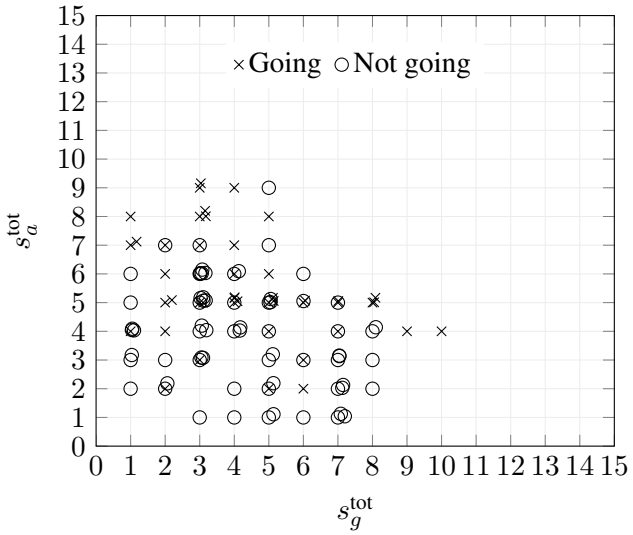


Figure 13.1: Answers to question b of extrinsic evaluation, one mark for each form.

reviews, there would be a concentration of circles (not going) answers in the bottom-right corner of the scatter plot and a concentration of crosses (going) in the left part of the image. In our case, Figure 13.1 highlights a concentration of crosses (going) answers in the top-left portion, i.e., positive answers in a region with high ratings of false reviews and low ratings of true reviews.

13.4.2 Intrinsic evaluation

With the intrinsic evaluation we aimed at evaluating if a human user is able to discriminate between genuine and generated reviews. In other words, we wanted to evaluate the effectiveness of our method in generating human-like reviews.

To this end, we constructed a set of forms, each composed of the name of a restaurant and 5 reviews. In each form, reviews could be partitioned in 4 classes according to the way we chose them (at least one review in each class): R_{gs} refers to reviews written by a human for the restaurant in the form; R_{gd} refers to reviews written by a human for a different restaurant; R_{as} refers to reviews generated with our method with the categories of the restaurant and a random rating as inputs; and R_{ad} of reviews generated using only the first step of our method. We asked the user, for each review, if the review was written by a human for the restaurant in the form. Since the nature of the question could suggest that some reviews were not written by humans, the forms of intrinsic evaluation were presented to each user after the forms for extrinsic evaluation. We proposed 4 forms to each user and we collected the evaluations of 39 different users.

The main results of the intrinsic evaluation are reported in Table 13.5. Each row corresponds to one of the classes described above and shows the distribution of number and percentage of answers, i.e., either genuine or artificial. The main finding of this evaluation is that a review generated with our

	Looks genuine		Looks artificial	
	#	%	#	%
R_{gs}	158	81	37	19
R_{gd}	102	52	93	48
R_{as}	47	24	148	76
R_{ad}	46	24	149	76

Table 13.5: Number and percentage of answers for the intrinsic evaluation, grouped by class of reviews.

method is considered genuine more frequently than a genuine review is considered artificial (24% for R_{as} vs. 19% for R_{gs}). On the other hand, the percentage of answers for R_{as} and R_{ad} are essentially identical, suggesting that, for this kind of evaluation, the contribution of steps ii and iii of our method is not significant.

13.5 Remarks

In this chapter we have proposed a method for the automatic generation of restaurant reviews based on LSTM-RNN. The method is able to generate reviews tailored to a rating and a set of categories specified as input.

A key contribution of our work is the experimental evaluation involving 39 human users. The results are promising (or should we say worrisome?): about 30% of reviews generated by our method are considered useful by human users; the opinion of a user on a restaurant, when presented with a mix of genuine and automatically-generated reviews, conflicts with the polarity of genuine reviews in $\approx 25\%$ of the times; automatically-generated reviews are considered truthful more frequently than genuine reviews are considered artificial.

Although our approach is certainly to be investigated further, and although we focus only on the textual content of reviews while systems for detection of non-genuine reviews consider also other features describing users activities, we believe that our work provides strong indications that machine-generated reviews may soon become a real threat for the integrity of review-based systems.

Bibliography

[1] Same origin policy. http://www.w3.org/Security/wiki/Same_Origin_Policy.

[2] JSMeter: characterizing real-world behavior of JavaScript programs. Technical Report MSR-TR-2009-173, Microsoft Research, 2009.

[3] Students busted for hacking computers, changing grades. http://www.theregister.co.uk/2012/01/27/students_hack_teachers_computers/, Jan. 2012.

[4] ISPs scramble to explain mouse-sniffing tool. http://www.theregister.co.uk/2013/08/27/isps_scramble_to_explain_away_mousesniffing/, Aug. 2013.

[5] A. Ahmed and I. Traore. A new biometric technology based on mouse dynamics. *IEEE Transactions on Dependable and Secure Computing*, 4(3):165–179, July 2007.

[6] D. Angluin. Queries and concept learning. *Machine learning*, 2(4):319–342, 1988.

[7] L. Araujo. Symbiosis of evolutionary techniques and statistical natural language processing. *Trans. Evol. Comp*, 8(1):14–27, Feb. 2004.

[8] L. Araujo. How evolutionary algorithms are applied to statistical natural language processing. *Artif. Intell. Rev.*, 28(4):275–303, Dec. 2007.

[9] J. Atserias, M. Simi, and H. Zaragoza. H.: Active learning for building a corpus of questions for parsing. In *In: Proceedings of LREC 2010*, 2010.

[10] R. Babbar and N. Singh. Clustering based approach to learning regular expressions over large alphabet for noisy unstructured text. In *Proceedings of the Fourth Workshop on Analytics for Noisy Unstructured Text Data*, AND '10, pages 43–50, New York, NY, USA, 2010. ACM.

[11] D. Barrero, D. Camacho, and M. R-Moreno. Automatic Web Data Extraction Based on Genetic Algorithms and Regular Expressions. *Data Mining and Multi-agent Integration*, pages 143–154, 2009.

[12] D. F. Barrero, M. D. R-Moreno, and D. Camacho. Adapting searchy to extract data using evolved wrappers. *Expert Systems with Applications*, 39(3):3061–3070, Feb. 2012.

[13] R. C. Barros, M. P. Basgalupp, A. C. de Carvalho, and A. A. Freitas. A hyper-heuristic evolutionary algorithm for automatically designing decision-tree algorithms. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, pages 1237–1244. ACM, 2012.

[14] A. Bartoli, A. Dagri, A. De Lorenzo, E. Medvet, and F. Tarlao. An Author Verification Approach Based on Differential Features—Notebook for PAN at CLEF 2015. In Cappellato et al. [171].

[15] A. Bartoli, G. Davanzo, A. De Lorenzo, M. Mauri, E. Medvet, and E. Sorio. Automatic generation of regular expressions from examples with genetic programming. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '12, pages 1477–1478, New York, NY, USA, 2012. ACM.

[16] A. Bartoli, G. Davanzo, A. De Lorenzo, E. Medvet, and E. Sorio. Automatic synthesis of regular expressions from examples. *Computer*, 47(12):72–80, Dec 2014.

[17] A. Bartoli, G. Davanzo, A. De Lorenzo, E. Medvet, and E. Sorio. Automatic synthesis of regular expressions from examples. *Computer*, 47(12):72–80, Dec 2014.

[18] A. Bartoli, G. Davanzo, A. De Lorenzo, E. Medvet, and E. Sorio. Automatic synthesis of regular expressions from examples. *Computer*, 47:72–80, 2014.

[19] A. Bartoli, A. De Lorenzo, A. Laderchi, E. Medvet, and F. Tarlao. An author profiling approach based on language-dependent content and stylometric features. In *Proceedings of CLEF*, 2015.

[20] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Playing regex golf with genetic programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 1063–1070. ACM, 2014.

[21] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Learning text patterns using separate-and-conquer genetic programming. In *European Conference on Genetic Programming*, pages 16–27. Springer, 2015.

[22] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Active learning approaches for learning regular expressions with genetic programming. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 97–102. ACM, 2016.

[23] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Inference of regular expressions for text extraction from examples. *IEEE Transactions on Knowledge and Data Engineering*, 28(5):1217–1230, 2016.

[24] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. On the automatic construction of regular expressions from examples (gp vs. humans 1-0). In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 155–156. ACM, 2016.

[25] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Predicting the effectiveness of pattern-based entity extractor inference. *Applied Soft Computing*, 46:398–406, 2016.

[26] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Regex-based entity extraction with active learning and genetic programming. *ACM SIGAPP Applied Computing Review*, 16(2):7–15, 2016.

[27] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Syntactical similarity learning by means of grammatical evolution. In *International Conference on Parallel Problem Solving from Nature*, pages 260–269. Springer, 2016.

[28] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Your paper has been accepted, rejected, or whatever: Automatic generation of scientific paper reviews. In *International Conference on Availability, Reliability, and Security*, pages 19–28. Springer, 2016.

[29] A. Bartoli, A. De Lorenzo, E. Medvet, F. Tarlao, and D. Morello. "best dinner ever!!!": Automatic generation of restaurant reviews with lstm-rnn. In *2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. IEEE, 2016.

[30] A. Bartoli, A. De Lorenzo, E. Medvet, F. Tarlao, and M. Virgolin. Evolutionary learning of syntax patterns for genic interaction extraction. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1183–1190. ACM, 2015.

[31] A. Bartoli, A. D. Lorenzo, E. Medvet, and F. Tarlao. Data quality challenge: Toward a tool for string processing by examples. *Journal of Data and Information Quality (JDIQ)*, 6(4):13, 2015.

[32] A. Bartoli and E. Medvet. Bibliometric evaluation of researchers in the internet age. *The Information Society*, 30(5):349–354, 2014.

[33] A. Bartoli, E. Medvet, A. D. Lorenzo, and F. Tarlao. Can a machine replace humans in building regular expressions? a case study. *IEEE Intelligent Systems*, PP(99):1–1, 2016.

[34] D. F. Bauer. Constructing confidence sets using rank statistics. *Journal of the American Statistical Association*, 67(339):687–690, 1972.

[35] J. Beall. List of predatory publishers 2016. https://scholarlyoa.com/2016/01/05/bealls-list-of-predatory-publishers-2016. Accessed: 2016-29-04.

[36] A. Bellet, A. Habrard, and M. Sebban. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*, 2013.

[37] A. Belz. Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models. *Natural Language Engineering*, 14(04):431–455, 2008.

[38] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 289–300, 1995.

[39] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, (5):16–23, 2003.

[40] J. Bongard and H. Lipson. Active coevolutionary learning of deterministic finite automata. *The Journal of Machine Learning Research*, 6:1651–1678, 2005.

[41] J. Bonneau. Authentication is machine learning. *Light Blue Touchpaper - Security Research, University of Cambridge*, Dec. 2012.

[42] J. Bootkrajang, S. Kim, and B.-T. Zhang. Evolutionary hypernetwork classifiers for protein-proteininteraction sentence filtering. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 185–192, New York, NY, USA, 2009. ACM.

[43] J. D. Bowman. Predatory publishing, questionable peer review, and fraudulent conferences. *American journal of pharmaceutical education*, 78(10), 2014.

[44] F. Brauer, R. Rieger, A. Mocan, and W. Barczynski. Enabling information extraction by inference of regular expressions from sample entities. In *ACM International Conference on Information and knowledge management*, pages 1285–1294. ACM, 2011.

[45] A. Brāzma. Efficient identification of regular expressions from representative examples. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, COLT '93, pages 236–242, New York, NY, USA, 1993. ACM.

[46] A. Bràzma. Efficient identification of regular expressions from representative examples. In *Conference on Computational learning theory*, volume 1, pages 236–242. ACM, 1993.

[47] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[48] D. D. A. Bui and Q. Zeng-Treitler. Learning regular expressions for clinical text classification. *Journal of the American Medical Informatics Association*, 21(5):850–857, 2014.

[49] M. Bundschus, M. Dejori, M. Stetter, V. Tresp, and H.-P. Kriegel. Extraction of semantic biomedical relations from text using conditional random fields. *BMC Bioinformatics*, 9(1):207, 2008.

[50] E. K. Burke, S. Gustafson, and G. Kendall. Diversity in genetic programming: An analysis of measures and correlation with fitness. *Evolutionary Computation, IEEE Transactions on*, 8(1):47–62, 2004.

[51] D. Butler et al. The dark side of publishing. *Nature*, 495(7442):433–435, 2013.

[52] W. Cai, M. Zhang, and Y. Zhang. Active learning for ranking with sample density. *Information Retrieval Journal*, 18(2):123–144, 2015.

[53] E. Callaway. Faked peer reviews prompt 64 retractions. *Nature*, aug 2015.

[54] J.-R. Cano. Analysis of data complexity measures for classification. *Expert Systems with Applications*, 40(12):4820 – 4831, 2013.

[55] A. Cetinkaya. Regular expression generation through grammatical evolution. In *Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, pages 2643–2646. ACM, 2007.

[56] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[57] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

[58] M. Cheatham and P. Hitzler. String similarity metrics for ontology alignment. In *The Semantic Web–ISWC 2013*, pages 294–309. Springer, 2013.

[59] Z. Chu, S. Gianvecchio, A. Koehl, H. Wang, and S. Jajodia. Blog or block: Detecting blog bots through behavioral biometrics. *Computer Networks*, 57(3):634–646, Feb. 2013.

[60] O. Cicchello and S. C. Kremer. Inducing grammars from sparse data sets: a survey of algorithms and results. *The Journal of Machine Learning Research*, 4:603–632, 2003.

[61] F. Ciravegna et al. Adaptive information extraction from text by rule induction and generalisation. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 1251–1256. LAWRENCE ERLBAUM ASSOCIATES LTD, 2001.

[62] R. A. Cochran, L. D'Antoni, B. Livshits, D. Molnar, and M. Veanes. Program boosting: Program synthesis via crowd-sourcing. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 677–688, New York, NY, USA, 2015. ACM.

[63] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string metrics for matching names and records. In *Kdd workshop on data cleaning and object consolidation*, volume 3, pages 73–78, 2003.

[64] W. Cohen, P. Ravikumar, and S. Fienberg. Secondstring: An open source java toolkit of approximate string-matching techniques. http://secondstring.sourceforge.net, 2003.

[65] W. R. Cohen and P. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IJCAI-03 Workshop on Information Integration on the Web*, 2003.

[66] S. Cronen-Townsend, Y. Zhou, and W. B. Croft. Predicting query performance. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '02, pages 299–306, New York, NY, USA, 2002. ACM.

[67] A. Csiszar. Peer review: Troubled from the start. *Nature*, 532(7599):306–308, apr 2016.

[68] M. Dadkhah, A. M. Alharbi, M. H. Al-Khresheh, T. Sutikno, T. Maliszewski, M. D. Jazi, and S. Shamshirband. Affiliation oriented journals: Don't worry about peer review if you have good affiliation. *International Journal of Electrical and Computer Engineering*, 5(4):621, 2015.

[69] K. Davydov and A. Rostamizadeh. Smart autofill - harnessing the predictive power of machine learning in google sheets. http://googleresearch.blogspot.it/2014/10/smart-autofill-harnessing-predictive.html, Oct. 2014.

[70] J. De Freitas, G. L. Pappa, A. S. Da Silva, M. Gonçalves, E. Moura, A. Veloso, A. H. Laender, M. G. De Carvalho, et al. Active learning genetic programming for record deduplication. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[71] E. D. De Jong and J. B. Pollack. Multi-objective methods for tree size control. *Genetic Programming and Evolvable Machines*, 4(3):211–233, 2003.

[72] C. De La Higuera. A bibliographical study of grammatical inference. *Pattern recognition*, 38(9):1332–1348, 2005.

[73] A. De Luca, A. Hang, F. Brudy, C. Lindner, and H. Hussmann. Touch Me Once and I Know It's You!: Implicit Authentication Based on Touch Screen Patterns. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 987–996, 2012.

[74] G. De Pauw. Evolutionary computing as a tool for grammar development. In *Proceedings of the 2003 International Conference on Genetic and Evolutionary Computation: PartI*, GECCO'03, pages 549–560, Berlin, Heidelberg, 2003. Springer-Verlag.

[75] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182 –197, apr 2002.

[76] B. Decadt, , B. Decadt, V. Hoste, W. Daelemans, and A. V. D. Bosch. Gambl, genetic algorithm optimization of memory-based wsd. In *In Proceedings of ACL/SIGLEX Senseval-3*, pages 108–112, 2004.

[77] F. Denis. Learning regular languages from simple positive examples. *Machine Learning*, 44(1-2):37–66, 2001.

[78] B. Dunay, F. Petry, and B. Buckles. Regular language induction with genetic programming. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 396–400 vol.1, Jun 1994.

[79] A. D'Ulizia, F. Ferri, and P. Grifoni. A survey of grammatical inference methods for natural language learning. *Artificial Intelligence Review*, 36(1):1–27, 2011.

[80] N. Eldredge. Mathgen paper accepted! Technical report, That's Mathematics, 2012.

[81] P. G. Espejo, S. Ventura, and F. Herrera. A survey on the application of genetic programming to classification. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 40(2):121–144, 2010.

[82] A. Esuli, D. Marcheggiani, and F. Sebastiani. Sentence-based active learning strategies for information extraction. In *IIR*, pages 41–45, 2010.

[83] C. Ferguson, A. Marcus, and I. Oransky. Publishing: The peer-review scam. *Nature*, 515(7528):480–482, nov 2014.

[84] H. Fernau. Algorithms for learning regular expressions from positive data. *Information and Computation*, 207(4):521 – 541, 2009.

[85] H. Fernau. Algorithms for learning regular expressions from positive data. *Information and Computation*, 207(4):521–541, Apr. 2009.

[86] R. L. Figueroa, Q. Zeng-Treitler, L. H. Ngo, S. Goryachev, and E. P. Wiechmann. Active learning for clinical text classification: is it better than random sampling? *Journal of the American Medical Informatics Association*, 19(5):809–816, 2012.

[87] J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[88] J. Fischman. Fake peer reviews, the latest form of scientific fraud, fool journals. Technical report, The Chronicle of Higher Education, 2012.

[89] J. Friedl. *Mastering Regular Expressions*. O'Reilly Media, Inc., 2006.

[90] K. Fundel, R. Küffner, and R. Zimmer. Relex—relation extraction using dependency parse trees. *Bioinformatics*, 23(3):365–371, 2007.

[91] J. Fürnkranz. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1):3–54, 1999.

[92] P. García, M. V. d. Parga, G. I. Álvarez, and J. Ruiz. Universal automata and {nfa} learning. *Theoretical Computer Science*, 407(1–3):192 – 202, 2008.

[93] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[94] L. Goeuriot, L. Kelly, and J. Leveling. An analysis of query difficulty for information retrieval in the medical domain. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 1007–1010. ACM, 2014.

[95] J. Golbeck, C. Robles, M. Edmondson, and K. Turner. Predicting personality from twitter. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*, pages 149–156. IEEE, 2011.

[96] M. Graff, R. Poli, and J. J. Flores. Models of performance of evolutionary program induction algorithms based on indicators of problem difficulty. *Evolutionary computation*, 21(4):533–560, 2013.

[97] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[98] S. N. Group. TokensRegex. http://nlp.stanford.edu/software/tokensregex.shtml, 2011.

[99] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 317–330, New York, NY, USA, 2011. ACM.

[100] D. Gunetti and C. Picardi. Keystroke analysis of free text. *ACM Trans. Inf. Syst. Secur.*, 8(3):312–347, Aug. 2005.

[101] S. Gupta, D. L. MacLean, J. Heer, and C. D. Manning. Induced lexico-syntactic patterns improve information extraction from online medical forums. *Journal of the American Medical Informatics Association*, 21(5):902–909, 2014.

[102] J. Hakenberg, C. Plake, U. Leser, H. Kirsch, and D. Rebholz-Schuhmann. LLL'05 challenge: Genic interaction extraction-identification of language patterns based on alignment and finite state automata. In *Proceedings of the 4th Learning Language in Logic workshop (LLL05)*, pages 38–45, 2005.

[103] S. Hao, P. Zhao, S. C. Hoi, and C. Miao. Learning relative similarity from data streams: Active online learning approaches. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1181–1190. ACM, 2015.

[104] C. Hauff, D. Hiemstra, and F. de Jong. A survey of pre-retrieval query performance predictors. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 1419–1420, New York, NY, USA, 2008. ACM.

[105] J. Hawes. Jail time for university hacker who changed his grades to straight as. http://nakedsecurity.sophos.com/2014/02/28/jail-time-for-university-hacker-who-changed-his-grades-to-straight-as/, Feb. 2014.

[106] J. He, C. Reeves, C. Witt, and X. Yao. A note on problem difficulty measures in black-box optimization: Classification, realizations and predictability. *Evolutionary Computation*, 15(4):435–443, 2007.

[107] HEFC. Identification and dissemination of lessons learned by institutions participating in the research excellence framework (ref) bibliometrics pilot. Technical report, Higher Education Funding Council for England, 2009.

[108] A. Heise, G. Kasneci, and F. Naumann. Estimating the number and sizes of fuzzy-duplicate clusters. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 959–968. ACM, 2014.

[109] L. Hirsch, R. Hirsch, and M. Saeedi. Evolving lucene search queries for text classification. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1604–1611. ACM, 2007.

[110] T. K. Ho and M. Basu. Complexity measures of supervised classification problems. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(3):289–300, 2002.

[111] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[112] E. J. Hughes. Evolutionary many-objective optimisation: many once or one many? In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 222–227. IEEE, 2005.

[113] R. Isele and C. Bizer. Active learning of expressive linkage rules using genetic programming. *Web Semantics: Science, Services and Agents on the World Wide Web*, 23:2–15, 2013.

[114] Z. Jorgensen and T. Yu. On mouse dynamics as a behavioral biometric for authentication. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 476–482, 2011.

[115] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

[116] G. Katz, A. Shtock, O. Kurland, B. Shapira, and L. Rokach. Wikipedia-based query performance prediction. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 1235–1238. ACM, 2014.

[117] E. Kinber. Learning regular expressions from representative examples and membership queries. *Grammatical Inference: Theoretical Results and Applications*, pages 94–108, 2010.

[118] E. Kinber. Learning regular expressions from representative examples and membership queries. *Grammatical Inference: Theoretical Results and Applications*, pages 94–108, 2010.

[119] P. Kluegl, M. Toepfer, P.-D. Beck, G. Fette, and F. Puppe. Uima ruta: Rapid development of rule-based information extraction applications. *Natural Language Engineering*, FirstView:1–40, 4 2015.

[120] J. R. Koza. Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems). 1992.

[121] K. Kukich. Where do phrases come from: Some preliminary experiments in connectionist phrase generation. In *Natural language generation*, pages 405–421. Springer, 1987.

[122] S. Kulick, A. Bies, M. Liberman, M. Mandel, R. McDonald, M. Palmer, A. Schein, L. Ungar, S. Winters, and P. White. Integrated annotation for biomedical information extraction. In *Proc. of HLT/NAACL*, pages 61–68, 2004.

[123] B. Kulis. Metric learning: A survey. *Foundations and Trends in Machine Learning*, 5(4):287–364, 2012.

[124] C. Labbé and D. Labbé. Duplicate and fake publications in the scientific literature: how many scigen papers in computer science? *Scientometrics*, 94(1):379–396, 2013.

[125] G. Lackermair, D. Kailer, and K. Kanmaz. Importance of online product reviews from a consumer's perspective. *Advances in Economics and Business*, 1(1):1–5, 2013.

[126] K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference*, page 1–12. Springer, 1998.

[127] K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference*, pages 1–12. Springer, 1998.

[128] W. Langdon and M. Harman. Optimizing existing software with genetic programming. *Evolutionary Computation, IEEE Transactions on*, 19(1):118–135, Feb 2015.

[129] V. Le and S. Gulwani. Flashextract: A framework for data extraction by examples. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, pages 542–553, New York, NY, USA, 2014. ACM.

[130] H. Lee, R. T. Ng, and K. Shim. Extending q-grams to estimate selectivity of string matching with low edit distance. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 195–206. VLDB Endowment, 2007.

[131] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12. Springer-Verlag New York, Inc., 1994.

[132] J. Li, Z. Zhang, X. Li, and H. Chen. Kernel-based learning for biomedical relation extraction. *Journal of the American Society for Information Science and Technology*, 59(5):756–769, 2008.

[133] Y. Li, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Jagadish. Regular expression learning for information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 21–30. Association for Computational Linguistics, 2008.

[134] Y. Li, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. V. Jagadish. Regular expression learning for information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 21–30, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[135] A. Liefooghe, S. Verel, F. Daolio, H. Aguirre, and K. Tanaka. A feature-based performance analysis in evolutionary multiobjective optimization. In *Evolutionary Multi-Criterion Optimization*, pages 95–109. Springer, 2015.

[136] M. Litvak, M. Last, and M. Friedman. A new approach to improving multilingual summarization using a genetic algorithm. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 927–936, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[137] C. Liu, J. Liu, and N. J. Belkin. Predicting search task difficulty at different search stages. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 569–578. ACM, 2014.

[138] W.-Y. Loh. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):14–23, 2011.

[139] Z. Lu, X. Wu, and J. C. Bongard. Active learning through adaptive heterogeneous ensembling. *Knowledge and Data Engineering, IEEE Transactions on*, 27(2):368–381, 2015.

[140] S. M. Lucas and T. J. Reynolds. Learning deterministic finite automata with a smart state labeling evolutionary algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1063–1074, 2005.

[141] N. A. H. Mamitsuka. Query learning strategies using boosting and bagging. In *Machine Learning: Proceedings of the Fifteenth International Conference (ICML'98)*, page 1. Morgan Kaufmann Pub, 1998.

[142] C. Manning and D. Klein. Optimization, maxent models, and conditional estimation without magic. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: Tutorials-Volume 5*, pages 8–8. Association for Computational Linguistics, 2003.

[143] J. Mao, W. Xu, Y. Yang, J. Wang, Z. Huang, and A. Yuille. Deep captioning with multimodal recurrent neural networks (m-rnn). *arXiv preprint arXiv:1412.6632*, 2014.

[144] E. Medvet, A. Bartoli, F. Boem, and F. Tarlao. Continuous and non-intrusive reauthentication of web sessions based on mouse dynamics. In *Availability, Reliability and Security (ARES), 2014 Ninth International Conference on*, pages 166–171. IEEE, 2014.

[145] E. Medvet, A. Bartoli, B. Carminati, and E. Ferrari. Evolutionary inference of attribute-based access control policies. In *Evolutionary Multi-Criterion Optimization*, page to appear. Springer, 2015.

[146] T. Megano, K.-i. Fukui, M. Numao, and S. Ono. Evolutionary multi-objective distance metric learning for multi-label clustering. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 2945–2952. IEEE, 2015.

[147] P. Melville and R. J. Mooney. Diverse ensembles for active learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 74. ACM, 2004.

[148] A. Menon, O. Tamuz, S. Gulwani, B. Lampson, and A. Kalai. A machine learning framework for programming by example. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 187–95, 2013.

[149] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, volume 2, page 3, 2010.

[150] T. Mikolov, I. Sutskever, A. Deoras, H.-S. Le, S. Kombrink, and J. Cernocky. Subword language modeling with neural networks. *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)*, 2012.

[151] S. Mondal and P. Bours. Continuous authentication using mouse dynamics. In *Biometrics Special Interest Group (BIOSIG), 2013 International Conference of the*, pages 1–12, Sept. 2013.

[152] K. Murthy, D. P., and P. M. Deshpande. Improving recall of regular expressions for information extraction. In *Web Information Systems Engineering - WISE 2012*, volume 7651 of *Lecture Notes in Computer Science*, pages 455–467. Springer Berlin Heidelberg, 2012.

[153] Y. Nakkabi, I. Traore, and A. Ahmed. Improving mouse dynamics biometric performance using variance reduction via extractors with separate features. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 40(6):1345–1353, Nov. 2010.

[154] V. Narayanan, I. Arora, and A. Bhatia. Fast and accurate sentiment classification using an enhanced naive bayes model. In *Intelligent Data Engineering and Automated Learning–IDEAL 2013*, pages 194–201. Springer, 2013.

[155] A.-C. N. Ngomo and K. Lyko. Eagle: Efficient active learning of link specifications using genetic programming. In *The Semantic Web: Research and Applications*, pages 149–163. Springer, 2012.

[156] R. V. Noorden. Publishers withdraw more than 120 gibberish papers. *Nature*, feb 2014.

[157] P. Norvig. xkcd 1313: Regex golf. http://nbviewer.ipython.org/url/norvig.com/ipython/xkcd1313.ipynb, Jan. 2014.

[158] L. O'Gorman. Comparing passwords, tokens, and biometrics for user authentication. *Proceedings of the IEEE*, 91(12):2021–2040, Dec. 2003.

[159] A. H. Oh and A. I. Rudnicky. Stochastic natural language generation for spoken dialog systems. *Computer Speech & Language*, 16(3):387–407, 2002.

[160] F. Olsson. A literature survey of active machine learning in the context of natural language processing. 2009.

[161] M. O'Neill and C. Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.

[162] W. Oremus. This is what happens when no one proofreads an academic paper. http://www.slate.com/blogs/future_tense/2014/11/11/_crappy_gabor_paper_overly_honest_citation_slips_into_peer_reviewed_journal.html, 2016.

[163] G. L. Pappa and A. A. Freitas. Evolving rule induction algorithms with multi-objective grammar-based genetic programming. *Knowledge and information systems*, 19(3):283–309, 2009.

[164] J. W. Pennebaker, M. E. Francis, and R. J. Booth. Linguistic inquiry and word count (liwc): A computerized text analysis program. *Mahwah (NJ)*, 7, 2001.

[165] J. Pihera and N. Musliu. Application of machine learning to algorithm selection for tsp. In *Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on*, pages 47–54. IEEE, 2014.

[166] H. Poon, C. Quirk, C. DeZiel, and D. Heckerman. Literome: Pubmed-scale genomic knowledge base in the cloud. *Bioinformatics*, 2014.

[167] P. Potash, A. Romanov, and A. Rumshisky. Ghostwriter: Using an lstm for automatic rap lyric generation. pages 1919–1924, 2015.

[168] P. Prasse, C. Sawade, N. Landwehr, and T. Scheffer. Learning to Identify Regular Expressions that Describe Email Campaigns. In *International Conference on Machine Learning (ICML)*, 2012.

[169] A. Project. UIMA-Ruta rule-based text annotation. https://uima.apache.org/ruta.html.

[170] J. Qiu, M. Schrope, N. Jones, B. Borrell, J. Tollefson, M. Kaplan, R. A. Lovett, R. Dalton, and Z. Merali. News publish or perish in china. *Nature*, 463:142–143, 2010.

[171] F. Rangel, P. Rosso, M. Potthast, B. Stein, and W. Daelemans. Overview of the 3rd author profiling task at PAN 2015. In L. Cappellato, N. Ferro, J. Gareth, and E. San Juan, editors, *CLEF 2015 Labs and Workshops, Notebook papers*, CEUR Workshop Proceedings. CLEF and CEUR-WS.org, Sept. 2015.

[172] H. Raviv, O. Kurland, and D. Carmel. Query performance prediction for entity retrieval. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 1099–1102. ACM, 2014.

[173] T. Reller. Faking peer reviews. Technical report, Elsevier Connect, 2012.

[174] V. Rieser and O. Lemon. Natural language generation as planning under uncertainty for spoken dialogue systems. In *Empirical methods in natural language generation*, pages 105–120. Springer, 2010.

[175] L. Rodríguez, I. García-Varea, and J. A. Gámez. On the application of different evolutionary algorithms to the alignment problem in statistical machine translation. *Neurocomput.*, 71(4-6):755–765, Jan. 2008.

[176] J. A. Rodriguez Perez and J. M. Jose. Predicting query performance in microblog retrieval. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 1183–1186. ACM, 2014.

[177] B. Rozenfeld and R. Feldman. Self-supervised relation extraction from the web. *Knowledge and Information Systems*, 17(1):17–33, 2008.

[178] A. M. Rush, S. Chopra, and J. Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.

[179] C. Ryan, J. Collins, and M. O. Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *Genetic Programming*, pages 83–96. Springer, 1998.

[180] T. Scheffer, C. Decomain, and S. Wrobel. Active hidden markov models for information extraction. In *Advances in Intelligent Data Analysis*, pages 309–318. Springer, 2001.

[181] J. Schler, M. Koppel, S. Argamon, and J. W. Pennebaker. Effects of age and gender on blogging. *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*, 6:199–205, 2006.

[182] H. Schmid. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of the international conference on new methods in language processing*, volume 12, pages 44–49. Citeseer, 1994.

[183] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. *Advances in neural information processing systems (NIPS)*, page 41, 2004.

[184] F. Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[185] J. I. Serrano. Evolutionary algorithm for noun phrase detection in natural language processing. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computing (IEEE Computer Society*, 2005.

[186] B. Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11, 2010.

[187] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 287–294, New York, NY, USA, 1992. ACM.

[188] C. Shen, Z. Cai, and X. Guan. Continuous authentication for mouse dynamics: A pattern-growth approach. In *2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12, 2012.

[189] C. Shen, Z. Cai, X. Guan, Y. Du, and R. Maxion. User authentication through mouse dynamics. *IEEE Transactions on Information Forensics and Security*, 8(1):16–30, 2013.

[190] D. C. Smith, A. Cypher, and L. Tesler. Programming by example: Novice programming comes of age. *Commun. ACM*, 43(3):75–81, Mar. 2000.

[191] K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis. Towards objective measures of algorithm performance across instance space. *Computers & Operations Research*, 45:12–24, 2014.

[192] K. Smith-Miles and L. Lopes. Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research*, 39(5):875–889, 2012.

[193] S. Soldz and G. E. Vaillant. The big five personality traits and the life course: A 45-year longitudinal study. *Journal of Research in Personality*, 33(2):208–232, 1999.

[194] D. Spina, M.-H. Peetz, and M. de Rijke. Active learning for entity filtering in microblog streams. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 975–978. ACM, 2015.

[195] A. Stahl and T. Gabel. Using evolution programs to learn local similarity measures. In *Case-Based Reasoning Research and Development*, pages 537–551. Springer, 2003.

[196] E. Stamatatos, W. Daelemans, B. Verhoeven, P. Juola, A. Lopez Lopez, M. Potthast, and B. Stein. Overview of the Author Identification Task at PAN 2015. In *Working Notes Papers of the CLEF 2015 Evaluation Labs*, CEUR Workshop Proceedings. CLEF and CEUR-WS.org, Sept. 2015.

[197] I. Sutskever, J. Martens, and G. E. Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[198] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[199] B. Svingen. Learning Regular Languages Using Genetic Programming. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998 Conference*, pages 374–376. Morgan Kaufmann, 1998.

[200] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics, 2003.

[201] I. Traore, I. Woungang, M. S. Obaidat, Y. Nakkabi, and I. Lai. Online risk-based authentication using behavioral biometrics. *Multimedia Tools and Applications*, pages 1–31, June 2013.

[202] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko. Translating videos to natural language using deep recurrent neural networks. *arXiv preprint arXiv:1412.4729*, 2014.

[203] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.

[204] M. Volk, B. Ripplinger, Š. Vintar, P. Buitelaar, D. Raileanu, and B. Sacaleanu. Semantic annotation for concept-based cross-language medical information retrieval. *International Journal of Medical Informatics*, 67(1):97–112, 2002.

[205] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2):275–309, 2013.

[206] T.-H. Wen, M. Gasic, N. Mrkšić, P.-H. Su, D. Vandyke, and S. Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. pages 1711–1721, September 2015.

[207] W. Wieczorek. Induction of non-deterministic finite automata on supercomputers. *Journal of Machine Learning Research-Proceedings Track*, 21:237–242, 2012.

[208] A. Wright. Algorithmic authors. *Communications of the ACM*, 58(11):12–14, 2015.

[209] T. Wu and W. M. Pottenger. A semi-supervised active learning algorithm for information extraction from textual data. *Journal of the American Society for Information Science and Technology*, 56(3):258–271, 2005.

[210] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming botnets: signatures and characteristics. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 171–182. ACM, 2008.

[211] N. Xiong. Learning fuzzy rules for similarity assessment in case-based reasoning. *Expert systems with applications*, 38(9):10780–10786, 2011.

[212] N. Xiong and P. Funk. Building similarity metrics reflecting utility in case-based reasoning. *Journal of Intelligent & Fuzzy Systems*, 17(4):407–416, 2006.

[213] S. Xiong, Y. Pei, R. Rosales, and X. Z. Fern. Active learning from relative comparisons. *Knowledge and Data Engineering, IEEE Transactions on*, 27(12):3166–3175, 2015.

[214] A. Yakushiji, Y. Miyao, T. Ohta, Y. Tateisi, and J. Tsujii. Automatic construction of predicate-argument structure patterns for biomedical information extraction. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 284–292. Association for Computational Linguistics, 2006.

[215] L. Yang and R. Jin. Distance metric learning: A comprehensive survey. *Michigan State Universiy*, 2, 2006.

[216] K. Yao, G. Zweig, and B. Peng. Attention with intention for a neural network conversation model. *arXiv preprint arXiv:1510.08565*, 2015.

[217] L. Yao, C.-J. Sun, X.-L. Wang, and X. Wang. Relationship extraction from biomedical literature using maximum entropy based on rich features. In *Machine Learning and Cybernetics (ICMLC), 2010 International Conference on*, volume 6, pages 3358–3361, July 2010.

[218] J. Yin, X. Jiang, Z. Lu, L. Shang, H. Li, and X. Li. Neural generative question answering. *arXiv preprint arXiv:1512.01337*, 2015.

[219] X. Zhang and M. Lapata. Chinese poetry generation with recurrent neural networks. In *EMNLP*, pages 670–680, 2014.