Evolutionary Learning with Kernels: A Generic Solution for Large Margin Problems

Ingo Mierswa
Artificial Intelligence Unit
Department of Computer Science
University of Dortmund
ingo.mierswa@uni-dortmund.de

ABSTRACT

In this paper we embed evolutionary computation into statistical learning theory. First, we outline the connection between large margin optimization and statistical learning and see why this paradigm is successful for many pattern recognition problems. We then embed evolutionary computation into the most prominent representative of this class of learning methods, namely into Support Vector Machines (SVM). In contrast to former applications of evolutionary algorithms to SVMs we do not only optimize the method or kernel parameters. We rather use both evolution strategies and particle swarm optimization in order to directly solve the posed constrained optimization problem. Transforming the problem into the Wolfe dual reduces the total runtime and allows the usage of kernel functions. Exploiting the knowledge about this optimization problem leads to a hybrid mutation which further decreases convergence time while classification accuracy is preserved. We will show that evolutionary SVMs are at least as accurate as their quadratic programming counterparts on six real-world benchmark data sets. The evolutionary SVM variants frequently outperform their quadratic programming competitors. Additionally, the proposed algorithm is more generic than existing traditional solutions since it will also work for non-positive semidefinite kernel functions and for several, possibly competing, performance criteria.

Track: Learning Classifier Systems and other Genetics-Based Machine Learning

Categories and Subject Descriptors: I.2.6 [Computing Methodologies]: Learning

General Terms: Algorithms, Theory, Experimentation

Keywords: Support vector machines, machine learning, kernel methods, evolution strategies, particle swarms

1. INTRODUCTION

In this paper we will discuss how evolutionary algorithms can be used to solve large margin optimization problems. We explore the intersection of three highly active research areas, namely machine learning, statistical learning theory, and evolutionary algorithms. While the connection between statistical learning and machine learning was analyzed before, embedding evolutionary algorithms into this connection will lead to a more generic algorithm which can deal with problems today's learning schemes cannot cope with.

Supervised machine learning is often about classification problems. A set of data points is divided into several classes and the machine learning method should learn a decision function in order to decide into which class an unseen data point should be classified.

The maximization of a margin between data points of different classes, i. e. the distance between a decision hyperplane and the nearest data points, interferes with the ideas of statistical learning theory. This allows the definition of an error bound for the generalization error. Furthermore, the usage of kernel functions allows the learning of non-linear decision functions. We focus on Support Vector Machines (SVM) as they are the most prominent representatives for large margin problems. Since SVMs guarantee an optimal solution for the given data set they are currently one of the mostly used learning methods. Furthermore, many other optimization problems can also be formulated as large margin problem [26]. The relevance of large margin methods can be measured by the number of submissions to the main machine learning conferences over the past years[1].

Usually, the optimization problem posed by SVMs is solved with quadratic programming. However, there are some drawbacks. First, for kernel functions which are not positive semidefinite no unique global optimum exists. In these cases quadratic programming is not able to find satisfying solutions at all. Moreover, most implementations do not even terminate [8]. There exist several useful non-positive kernels [15], among them the sigmoid kernel which simulates a neural network [3, 23]. A more generic optimization scheme should allow such non-positive kernels without the need for omitting the more efficient dual optimization problem [17].

Second, SVMs should be able to optimize several performance measures at the same time. Traditional SVMs try to maximize the prediction accuracy alone. However, depending on the application area other specific performance criteria should be optimized instead of or additionally to prediction accuracy. Although first attempts were made to incorporate multivariate performance measures into SVMs [13], the problem is not generally solved and no solution exist

[1]More than 30% of all accepted papers for ICML 2005 dealt with SVMs and other large margin methods.

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

for competing criteria. This problem as well as the general trade-off between training error and capacity could be easily solved by an (multi-objective) evolutionary optimization approach.

Former applications of evolutionary algorithms to SVMs include the optimization of method and kernel parameters [6, 19], the selection of optimal feature subsets [7], and the creation of new kernel functions by means of genetic programming [10]. The latter is particularly interesting since it cannot be guaranteed that the resulting kernel functions are again positive semi-definite.

Replacing the traditional optimization techniques by evolution strategies or particle swarm optimization can tackle both problems mentioned above. We will extract as much information as possible from the optimization problem at hand and develop and compare different search point operations. We will show that the proposed implementation leads to as good results as traditional SVMs on all real-world benchmark data sets. Additionally, the optimization is more generic since it also allows non-positive semi-definite kernel functions and the simultaneous optimization of different, maybe competing, criteria.

1.1 Outline

In Section 2 we give a short introduction into the concept of structural risk minimization and the ideas of statistical learning theory. We will also discuss an upper bound for the generalization error. This allows us to formalize the optimization problem of large margin methods in Section 3. We will introduce SVMs for the classification of given data points in Section 3.1 and extend the separation problem to non-separable datasets (see Section 3.2) with non-linear hyperplanes (see Section 3.3). This leads to a constrained optimization problem for which we utilize evolution strategies and particle swarm optimization in Section 4. We discuss several enhancements and a new type of mutation before we evaluate the proposed methods on real-world benchmark datasets in Section 5.

2. STRUCTURAL RISK MINIMIZATION

In this section we discuss the idea of structural risk minimization. Machine learning methods following this paradigm have a solid theoretical foundation and it is possible to define bounds for prediction errors.

Let $X \in \mathbb{R}^m$ be a real-valued vector of random variables. Let $Y \in \mathbb{R}$ be another random variable. X and Y obey a fixed but unknown probability distribution $P(X,Y)$. *Machine Learning* tries to find a function $f(x,\gamma)$ which predict the value of Y for a given input $x \in X$. The function class f depends on a vector of parameters γ, e. g. if f is the class of all polynomials, γ might be the degree. We define a *loss function* $L(Y, f(X,\gamma))$ in order to penalize errors during prediction [9]. Every convex function with arity 2, positive range, and $L(x,x) = 0$ can be used as loss function [22]. This leads to a possible criterion for the selection of a function f, the *expected risk*:

$$R(\gamma) = \int L(y, f(x,\gamma)) dP(x,y). \tag{1}$$

Since the underlying distribution is not known we are not able to calculate the expected risk. However, instead of estimating the probability distribution in order to allow this calculation, we directly estimate the expected risk by using a set of known data points $T = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subseteq X \times Y$. T is usually called *training data*. Using this set of data points we can calculate the *empirical risk*:

$$R_{emp}(\gamma) = \frac{1}{n} \sum_{i=1}^{n} L\left(y_i, f\left(x_i, \gamma\right)\right). \tag{2}$$

If training data is sampled according to $P(X,Y)$, the empirical risk approximates the expected risk if the number of samples grows:

$$\lim_{n\to\infty} R_{emp}(\gamma) = R(\gamma). \tag{3}$$

It is, however, a well known problem that for a finite number of samples the minimization of $R_{emp}(\gamma)$ alone does not lead to a good prediction model [27]. For each loss function L, each candidate γ, and each set of tuples $T' \subseteq X \times Y$ with $T \cap T' = \emptyset$ exists another parameter vector γ' so that $L(y, f(x,\gamma)) = L(y, f(x,\gamma'))$ for all $x \in T$ and $L(y, f(x,\gamma)) > L(y, f(x,\gamma'))$ for all $x \in T'$. Therefore, the minimization of $R_{emp}(\gamma)$ alone does not guarantee the optimal selection of a parameter vector γ for other samples according to the distribution $P(X,Y)$. This problem is often referred to as overfitting.

At this point we use one of the main ideas of statistical learning theory. Think of two different functions perfectly approximating a given set of training points. The first function is a linear function, i. e. a simple hyperplane in the considered space $\mathbb{R}^m$. The second function also hits all training points but is strongly wriggling in between. Naturally, if we had to choose between these two approximation functions, we tend to select the more simple one, i. e. the linear hyperplane in this example. This derives from the observation that more simple functions behave better on unseen examples than very complicated functions. Since the mere minimization of the empirical risk according to the training data is not appropriate to find a good generalization, we incorporate the *capacity*[2] of the used function into the optimization problem (see Figure 1). This leads to the minimization of the *structural risk*

$$R_{struct}(\gamma) = R_{emp}(\gamma) + \lambda\Omega(\gamma). \tag{4}$$

Ω is a function which measures the capacity of the function class f depending on the parameter vector γ. Since the empirical risk is usually a monotonically decreasing function of Ω, we use λ to manage the trade-off between training error and capacity. Methods minimizing this type of risk function are known as *shrinkage estimators* [11].

2.1 Bound on the generalization performance

For certain functions Ω the structural risk is an upper bound for the empirical risk. The capacity of the function f for a given γ can for example be measured with help of the *Vapnik-Chervonenkis dimension* (VC dimension) [27, 28]. The VC dimension is defined as the cardinality of the biggest set of tuples which can separated with help of f in all possible ways. For example, the VC dimension of linear hyperplanes in an m-dimensional space is $m+1$. Using the VC dimension as a measure for capacity leads to a probabilistic bound for the structural risk [27]. Let f be a function class with finite VC dimension h and $f(\gamma)$ the best solution for the

[2]Although not the same, the capacity of a function resembles a measurement of the function complexity. In our example we measure the ability to "wriggle". More details in [27].

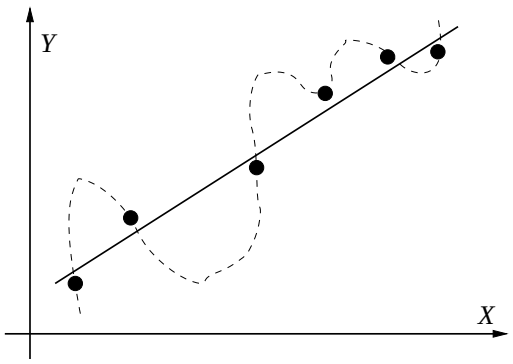

Figure 1: The simultaneous minimization of empirical risk and model complexity gives a hint which function should be used in order to generalize the given data points.

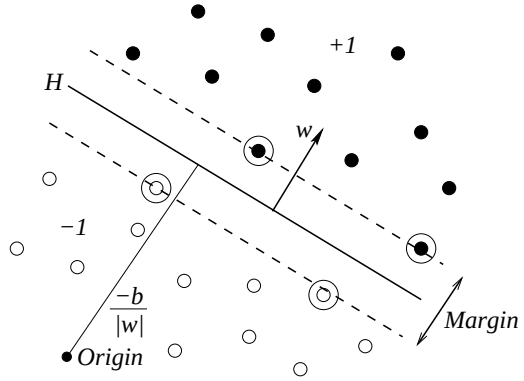

Figure 2: A simple binary classification problem for two classes -1 (empty bullets) and $+1$ (filled bullets). The separating hyperplane is defined by the vector w and the offset b. The distance between the nearest data point(s) and the hyperplane is called *margin*.

empirical risk minimization for T with $|T| = n$. Now choose some η such that $0 \leq \eta \leq 1$. Then for losses smaller than some number B, the following bound holds with probability $1 - \eta$:

$$R(\gamma) \leq R_{emp}(\gamma) + B\sqrt{\frac{h\left(\log\frac{2l}{h}+1\right) - \log\frac{\eta}{4}}{l}}. \qquad (5)$$

Surprisingly, this bound is independent of $P(X, Y)$. It only assumes that both the seen and the unseen data points are independently sampled according to some $P(X, Y)$. Please note that this bound also no longer contains a weighting factor λ or any other trade-off at all. The existence of a guaranteed error bound is the reason for the great success of structural risk minimization in a wide range of applications.

3. LARGE MARGIN METHODS

As discussed in the previous section we need to use a class of functions whose capacity can be controlled. In this section we will discuss a special form of structural risk minimization, namely large margin approaches. All large margin methods have one thing in common: they embed structural risk minimization by maximizing a margin between a linear function and the nearest data points. The most prominent large margin method for classification tasks is the *Support Vector Machine* (SVM).

3.1 Support Vector Machines

We constrain the number of possible values of Y to 2, without loss of generality these values should be -1 and $+1$. In this case, finding a function f in order to decide which of both predictions is correct for an unseen data point is referred to as *classification learning* for the classes -1 and $+1$. We start with the simplest case: learning a linear function from perfectly separable data. As we shall see in Section 3.2 and 3.3, the general case - non-linear functions derived from non-separable data - leads to a very similar problem.

If the data points are linearly separable, a linear hyperplane must exist in the input space $\mathbb{R}^m$ which separates both classes. This hyperplane is defined as

$$H = \{x | \langle w, x\rangle + b = 0\}, \qquad (6)$$

where w is normal to the hyperplane, $|b|/||w||$ is the perpendicular distance of the hyperplane to the origin (offset or bias), and $||w||$ is the Euclidean norm of w. The vector w and the offset b define the position and orientation of the hyperplane in the input space. These parameters correspond to the function parameters γ. After the optimal parameters w and b were found, the prediction of new data points can be calculated as

$$f(x, w, b) = \operatorname{sgn}\left(\langle w, x\rangle + b\right), \qquad (7)$$

which is one of the reasons why we constrained the classes to -1 and $+1$.

Figure 2 shows some data points and a separating hyperplane. If all given data points are correctly classified by the hyperplane at hand the following must hold:

$$\forall i : y_i\left(\langle w, x_i\rangle + b\right) \geq 0. \qquad (8)$$

Of course, an infinite number of different hyperplanes exist which perfectly separate the given data points. However, one would intuitively choose the hyperplane which has the biggest amount of safety margin to both sides of the data points. Normalizing w and b in a way that the point(s) closest to the hyperplane satisfy $|\langle w, x_i\rangle + b| = 1$ we can transform equation 8 into

$$\forall i : y_i\left(\langle w, x_i\rangle + b\right) \geq 1. \qquad (9)$$

We can now define the *margin* as the perpendicular distance of the nearest point(s) to the hyperplane. Consider two points x_1 and x_2 on opposite sides of the margin. That is $\langle w, x_1\rangle + b = +1$ and $\langle w, x_2\rangle + b = -1$ and $\langle w, (x_1 - x_2)\rangle = 2$. The margin is then given by $1/||w||$.

It can be shown, that the capacity of the class of separating hyperplanes decreases with increasing margin [21]. Maximizing the margin of a hyperplane therefore formalizes the structural risk minimization discussed in the previous section. Instead of maximizing $1/||w||$ we could also minimize $\frac{1}{2}||w||^2$ which will result into more simple equations later. This leads to the optimization problem

$$\text{minimize } \tfrac{1}{2}||w||^2 \qquad (10)$$

$$\text{subject to } \forall i : y_i\left(\langle w, x_i\rangle + b\right) \geq 1. \qquad (11)$$

Function 10 is the *objective function* and the constraints from equation 11 are called *inequality constraints*. They form a *constrained optimization problem*. We will use a Lagrangian formulation of the problem. This allows us to replace the inequality constraints by constraints on the Lagrange multipliers which are easier to handle. The second reason is that after the transformation of the optimization problem, the training data will only appear in dot products. This will allow us to generalize the optimization to the non-linear case (see Section 3.3). We will now introduce positive Lagrange multipliers $\alpha_i, i = 1, \ldots, n$, one for each of the inequality constraints. The Lagrangian has the form

$$L_P(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i y_i \left(\langle w, x_i \rangle + b\right). \quad (12)$$

Finding a minimum of this function requires that the derivatives

$$\frac{\partial L_P(w,b,\alpha)}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i \quad (13)$$

$$\frac{\partial L_P(w,b,\alpha)}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i \quad (14)$$

are zero, i. e.

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i \quad (15)$$

$$0 = \sum_{i=1}^{n} \alpha_i y_i. \quad (16)$$

The Wolfe dual, which has to be maximized, results from the Lagrangian by substituting 15 and 16 into 12, thus

$$L_D(w, b, \alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle. \quad (17)$$

This leads to the dual optimization problem which must be solved in order to find a separating maximum margin hyperplane for given set of data points:

$$\text{maximize } \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle \quad (18)$$

$$\text{subject to } \alpha_i \geq 0 \text{ for all } i = 1, \ldots, n \quad (19)$$

$$\text{and } \sum_{i=1}^{n} \alpha_i y_i = 0. \quad (20)$$

From an optimal vector α^* we can calculate the optimal normal vector w^* using equation 15. The optimal offset can be calculated with help of equation 11. Please note, that w is a linear combination of those data points x_i with $\alpha_i \neq 0$. These data points are called *support vectors*, hence the name support vector machine. Only support vectors determine the position and orientation of the separating hyperplane, other data points might as well be omitted during learning. In Figure 2 the support vectors are marked with circles. The number of support vectors is usually much smaller than the total number of data points.

3.2 Non-separable data

We now consider the case that the given set of data points is not linearly separable. The optimization problem discussed in the previous section would not have a solution since in this case constraint 11 could not be fulfilled for all i. We relax this constraint by introducing positive slack variables $\xi_i, i = 1, \ldots, n$. Constraint 11 becomes

$$\forall i : y_i \left(\langle w, x_i \rangle + b\right) \geq 1 - \xi_i. \quad (21)$$

In order to minimize the number of wrong classifications we introduce a correction term $C \sum_{i=1}^{n} \xi_i$ into the objective function. The optimization problems then becomes

$$\text{minimize } \tfrac{1}{2}||w||^2 + C \sum_{i=1}^{n} \xi_i \quad (22)$$

$$\text{subject to } \forall i : y_i \left(\langle w, x_i \rangle + b\right) \geq 1 - \xi_i. \quad (23)$$

The factor C determines the weight of wrong predictions as part of the objective function. As in the previous section we create the dual form of the Lagrangian. The slacking variables ξ_i vanish and we get the optimization problem

$$\text{maximize } \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle \quad (24)$$

$$\text{subject to } 0 \leq \alpha_i \leq C \text{ for all } i = 1, \ldots, n \quad (25)$$

$$\text{and } \sum_{i=1}^{n} \alpha_i y_i = 0. \quad (26)$$

It can easily be seen that the only difference to the separable case is the additional upper bound C for all α_i.

3.3 Non-linear learning with kernels

The optimization problem described with equations 24, 25, and 26 will deliver a linear separating hyperplane for arbitrary datasets. The result is optimal in a sense that no other linear function is expected to provide a better classification function on unseen data according to $P(X, Y)$. However, if the data is not linearly separable at all the question arises how the described optimization problem can be generalized to non-linear decision functions. Please note that the data points only appear in the form of dot products $\langle x_i, x_j \rangle$. A possible interpretation of this dot product is the similarity of these data points in the input space $\mathbb{R}^m$. Now consider a mapping $\Phi : \mathbb{R}^m \to H$ into some other Euclidean space H (called *feature space*) which might be performed before the dot product is calculated. The optimization would depend on dot products in this new space H, i. e. on functions of the form $\langle \Phi(x_i), \Phi(x_j) \rangle$. A function $k : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ with the characteristic

$$k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle \quad (27)$$

is called *kernel function* or *kernel*. Figure 3 gives a rough idea how transforming the data points can help to solve non-linear problems with the optimization in a (higher dimensional) space where the points can be linearly separated.

A fascinating property of kernels is that for some mappings Φ a kernel k exists which can be calculated without actually performing Φ. Since often the dimension of H is greater than the dimension m of the input space and H sometimes is even infinite dimensional, the usage of such kernels is a very efficient way to introduce non-linear decision functions into large margin approaches. Prominent examples for such efficient non-linear kernels are polynomial kernels with degree d

$$k(x_i, x_j) = (\kappa \langle x_i, x_j \rangle + \delta)^d, \quad (28)$$

radial basis function kernels (RBF kernels)

$$k(x_i, x_j) = e^{-\frac{||x_i - x_j||^2}{2\sigma^2}} \quad (29)$$

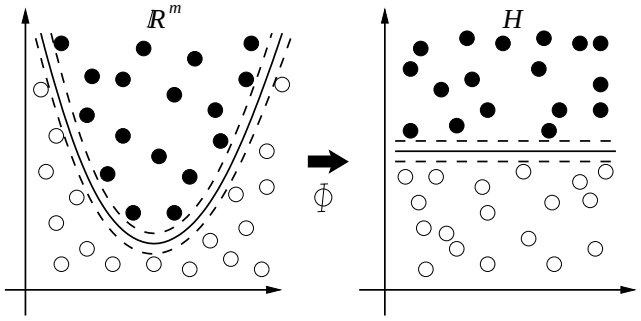

Figure 3: After the transformation of all data points into the feature space H the non-linear separation problem can be solved with a linear separation algorithm. In this case a transformation in the space of polynomials with degree 2 was chosen.

for a $\sigma > 0$, and the sigmoid kernel

$$k(x_i, x_j) = \tanh(\kappa \langle x_i, x_j \rangle - \delta) \tag{30}$$

which can be used to simulate a neural network. κ and δ are scaling and shifting parameters. Since the RBF kernel is easy interpretable and often yields good prediction performance, it is used in a wide range of applications. We will also use the RBF kernel for our experiments described in section 5 in order to demonstrate the learning ability of the proposed SVM.

We replace the dot product in the objective function by kernel functions and achieve the final optimization problem for finding a non-linear separation for non-separable data points

$$\text{maximize } \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j k(x_i, x_j) \tag{31}$$

$$\text{subject to } 0 \leq \alpha_i \leq C \text{ for all } i = 1, \ldots, n \tag{32}$$

$$\text{and } \sum_{i=1}^{n} \alpha_i y_i = 0. \tag{33}$$

It can be shown that if the kernel k, i. e. it's kernel matrix, is positive definite, the objective function is concave [2]. The optimization problem therefore has a global unique maximum. However, in some cases a specialized kernel function must be used to measure the similarity between data points which is not positive definite, sometimes not even positive semidefinite [21]. In these cases the usual quadratic programming approaches might not be able to find a global maximum in feasible time.

4. EVOLUTIONARY COMPUTATION FOR LARGE MARGIN OPTIMIZATION

Since traditional SVMs are not able to optimize for non-positive semidefinite kernel function, it is a very appealing idea to replace the usual quadratic programming approaches by an *evolution strategies* (ES) approach [1] or by *particle swarm optimization* (PSO) [14]. In this section we will describe both a straightforward application of these techniques and how we can exploit some information about our optimization problem and incorporate that information into our search operators.

4.1 Solving the dual problem and other simplifications

The used optimization problem is the dual problem for non-linear separation of non-separable data developed in the last sections (equations 31, 32, and 33). Of course it would also be possible to directly optimize the original form of our optimization problem depicted in equations 22 and 23. That is, we could directly optimize the weight vectors and the offset. As mentioned before, there are two drawbacks: first, the costs of calculating the fitness function would be much higher for the original optimization problem since the fulfillment of all n constraints must be recalculated for each new hyperplane. It is a lot easier to check if all $0 \leq \alpha_i \leq C$ apply. Second, it would not be possible to allow non-linear learning with efficient kernel functions in the original formulation of the problem. Furthermore, the kernel matrix K with $K_{ij} = k(x_i, x_j)$ can be calculated beforehand and the training data is never used during optimization again. This further reduces the needed runtime for optimization since the kernel matrix calculation is done only once.

This is a nice example for a case, where transforming the objective function beforehand is both more efficient and allows enhancements which would not have been possible before. Transformations of the fitness functions became a very interesting topic recently [25].

Another efficiency improvement can be achieved by formulating the problem with $b = 0$. All solution hyperplanes must then contain the origin and the constraint 33 will vanish. This is a mild restriction for high-dimensional spaces since the number of degrees of freedom is only decreased by one. However, during optimization we do not have to cope with this equality constraint which would take an additional runtime of $O(n)$.

4.2 EvoSVM and PsoSVM

We developed a support vector machine based on evolution strategies optimization (EvoSVM). We utilized three different types of mutation which will be described in this section. Furthermore, we developed another SVM based on particle swarm optimization (PsoSVM) which is also described.

The first approach (*EvoSVM-G*, *G* for *G*aussian mutation) merely utilizes a standard ES optimization. Individuals are the real-valued vectors α and mutation is performed by adding a Gaussian distributed random variable with standard deviation $C/10$. In addition, a variance adaptation is conducted during optimization (1/5 rule [18]). Crossover probability is high (0.9). We use tournament selection with a tournament size of 0.25 multiplied by the population size. The initial individuals are random vectors with $0 \leq \alpha_i \leq C$. The maximum number of generations is 1000 and the optimization is terminated if no improvement occurred during the last 5 generations. The population size is 10.

The second version is called *EvoSVM-S* (*S* for *s*witching mutation). Here we utilize the fact that only a small amount of input data points will become support vectors (*sparsity*). On the other hand, one can often observe that non-zero alpha values are equal to the upper bound C and only a very small amount of support vectors exists with $0 < \alpha_i < C$. Therefore, we just use the well known mutation of genetic algorithms and switch between 0 and C with probability $1/n$ for each α_i. The other parameters are equal to those described for the EvoSVM-G.

```
for i = 1 to n do {
    if (random(0, 1) < 1/n) do {
        if (alpha_i > 0) do {
            alpha_i = 0;
        } else do {
            alpha_i = random(0, C);
        }
    }
}
```

Figure 4: A simple hybrid mutation which should speed-up the search for sparser solutions. It contains elements from standard mutations from both genetic algorithms and evolution strategies.

Using this switching mutation inspired by genetic algorithms only allow $\alpha_i = 0$ or $\alpha_i = C$. Instead of a complete switch between 0 and C or a smooth change of all values α_i like the Gaussian mutation does, we developed a hybrid mutation combining both elements. That means that we check for each α_i with probability $1/n$ if the value should be mutated at all. If the current value α_i is greater than 0, α_i is set to 0. If α_i is equal to 0, α_i is set to a random value with $0 \leq \alpha_i \leq C$. Figure 4 gives an overview over this hybrid mutation. The function $random(a, b)$ returns an uniformly distributed random number between a and b. The other parameters are the same as described for the EvoSVM-G. We call this version *EvoSVM-H* (H for $h$ybrid).

As was mentioned before, the optimization problem usually is concave and the risk for local extrema is small. Therefore, we also applied a PSO technique. It should be investigated if PSO, which is similar to the usual quadratic programming approaches for SVMs in a sense that the gradient information is exploited, is able to find a global optimum in shorter time. We call this last version *PsoSVM* and use a standard PSO with inertia weight 0.1, local best weight 1.0, and global best weight 1.0. The inertia weight is dynamically adapted during optimization [14].

5. EXPERIMENTS AND RESULTS

In this section we try to evaluate the proposed evolutionary optimization SVMs. We compare our implementation to the quadratic programming approaches usually applied to large margin problems. The experiments demonstrate the competitiveness in terms of classification error minimization, runtime, and robustness.

We apply the discussed EvoSVM variants as well as the PsoSVM on six real-world benchmark datasets. We selected these datasets from the UCI machine learning repository [16] and the StatLib dataset library [24], because they already define a binary classification task, consist of real-valued numbers only and do not contain missing values. Therefore, we did not need to perform additional preprocessing steps which might introduce some bias. The properties of all datasets are summarized in Table 1. The default error corresponds to the error a lazy default classifier would make by always predicting the major class. Classifiers must produce lower error rates in order to learn at all instead of just guessing.

In order to compare the evolutionary SVMs described in this paper with standard implementations we also applied two other SVMs on all datasets. Both SVMs use a slightly different optimization technique based on quadratic programming. The used implementations were *mySVM* [20] and *LibSVM* [4]. The latter is an adaptation of the widely used SVM^{light} [12].

Dataset	n	m	**Source**	σ	**Default**
Liver	346	6	UCI	0.010	42.03
Ionosphere	351	34	UCI	1.000	35.90
Sonar	208	60	UCI	1.000	46.62
Lawsuit	264	4	StatLib	0.010	7.17
Lupus	87	3	StatLib	0.001	40.00
Crabs	200	7	StatLib	0.100	50.00

Table 1: The evaluation datasets. n is the number of data points, m is the dimension of the input space. The kernel parameter σ was optimized for the comparison SVM learner *mySVM*. The last column contains the default error, i. e. the error for always predicting the major class.

We use a RBF kernel for all SVMs and determine the best parameter value for σ with a grid search parameter optimization for mySVM. This ensures a fair comparison since the parameter is not optimized for one of the evolutionary SVMs. Possible parameters were 0.001, 0.01, 0.1, 1 and 10. The optimal value for each dataset is also given in Table 1.

In order to determine the performance of all methods we perform a k-fold cross validation. That means that the dataset T is divided into k disjoint subsets T_i. For each $i \in \{1, \ldots, k\}$ we use $T \backslash T_i$ as training set and the remaining subset T_i as test set. If F_i is the number of wrong predictions on test set T_i we calculate the average classification error

$$E = \frac{1}{k} \sum_{i=1}^{k} \frac{F_i}{|T_i|} \tag{34}$$

over all test sets in order to measure the classification performance. In our experiments we choose $k = 20$, i. e. for each evolutionary method the average and standard deviation of 20 runs is reported. All experiments were performed with the machine learning environment YALE [5].

Table 2 summarizes the results for different values of C. It can be seen that the EvoSVM variants frequently yield smaller classification errors than the quadratic programming counterparts (mySVM and LibSVM). For $C = 1$, a statistical significant better result was achieved by using LibSVM only for the Liver data set. For all other datasets the evolutionary optimization outperforms the quadratic programming approaches. The same applies for $C = 0.1$. For rather small values of C most learning schemes were not able to produce better predictions than the default classifier. For $C = 0.01$, however, PsoSVM at least provides a similar accuracy to LibSVM. The reason for higher errors of the quadratic programming approaches is probably a too aggressive termination criterion. Although this termination behavior further reduces runtime for mySVM and LibSVM, the classification error is often increased.

It turns out that the standard ES approach EvoSVM-G using a mutation adding a Gaussian distributed random variable often outperforms the other SVMs. However, the

$C = 1$												
	Liver		**Ionosphere**		**Sonar**		**Lawsuit**		**Lupus**		**Crabs**	
	Error	T	Error	T	Error	T	Error	T	Error	T	Error	T
EvoSVM-G	34.71±8.60	68	10.81±5.71	80	**14.03±4.52**	26	2.05±1.87	52	25.20±11.77	8	**2.25±3.72**	25
EvoSVM-S	35.37±6.39	4	8.49±3.80	9	17.45±6.64	6	2.40±1.91	10	30.92±12.42	<1	4.05±4.63	2
EvoSVM-H	34.97±7.32	7	**6.83±3.87**	22	15.41±6.39	10	2.01±1.87	14	**24.03±13.68**	1	3.95±4.31	7
PsoSVM	34.78±4.95	8	9.90±4.38	9	16.94±5.61	7	3.02±2.83	3	25.22± 7.67	<1	3.40±3.70	2
mySVM	33.62±4.31	2	8.56±4.25	4	15.81±5.59	2	**1.89±2.51**	1	25.28± 8.58	1	3.00±3.32	1
LibSVM	**32.72±5.41**	2	7.70±3.63	3	14.60±4.96	3	2.41±2.64	1	24.14±12.33	1	3.00±4.58	1
F Test	**3.20 (0.01)**		**9.78 (0.00)**		**6.19 (0.00)**		1.51 (0.19)		**11.94 (0.00)**		**2.25 (0.05)**	

$C = 0.1$												
	Liver		**Ionosphere**		**Sonar**		**Lawsuit**		**Lupus**		**Crabs**	
	Error	T	Error	T	Error	T	Error	T	Error	T	Error	T
EvoSVM-G	33.90±4.19	74	9.40±6.14	89	**21.72±6.63**	35	**2.35±1.92**	50	**24.90±10.51**	7	7.20±4.36	27
EvoSVM-S	35.57±3.55	4	7.12±3.54	18	24.90±6.62	4	4.47±2.31	13	25.98±12.56	<1	7.95±5.68	2
EvoSVM-H	34.76±4.70	5	**6.55±4.61**	23	24.40±6.09	11	4.16±3.14	19	26.51±13.03	1	6.50±5.02	2
PsoSVM	36.81±5.04	3	13.96±7.56	10	24.18±6.11	3	3.03±2.83	3	29.86±12.84	1	8.15±6.02	1
mySVM	42.03±1.46	2	35.90±1.35	2	46.62±1.62	2	7.17±2.55	1	41.25±6.92	1	**6.50±4.50**	1
LibSVM	**33.08±10.63**	2	11.40±6.52	3	22.40±6.45	3	4.55±3.25	1	25.29±16.95	1	21.00±12.41	1
F Test	**34.46 (0.00)**		**492.88 (0.00)**		**323.83 (0.00)**		**20.64 (0.00)**		**64.83 (0.00)**		**100.92 (0.00)**	

$C = 0.01$												
	Liver		**Ionosphere**		**Sonar**		**Lawsuit**		**Lupus**		**Crabs**	
	Error	T	Error	T	Error	T	Error	T	Error	T	Error	T
EvoSVM-G	42.03±1.46	75	35.90±1.35	86	45.33±2.20	39	7.17±2.55	55	40.00±6.33	7	26.20±12.66	27
EvoSVM-S	42.03±1.46	3	35.90±1.35	9	46.62±1.62	4	7.17±2.55	3	40.00±6.33	<1	8.58±4.35	1
EvoSVM-H	42.03±1.46	3	35.90±1.35	20	46.27±1.42	12	7.17±2.55	3	40.00±6.33	1	7.00±4.00	2
PsoSVM	**41.39±8.59**	3	35.90±1.35	4	**27.90±6.28**	3	7.17±2.55	2	31.94±12.70	1	10.05±7.26	1
mySVM	42.03±1.46	2	35.90±1.35	2	46.62±1.62	2	7.17±2.55	1	40.00±6.33	1	**6.50±4.50**	1
LibSVM	42.03±1.46	2	35.90±1.35	3	28.46±10.44	2	7.17±2.55	1	**26.11±16.44**	1	50.00±0.00	1
F Test	0.52 (0.77)		0.00 (1.00)		**442.46 (0.00)**		0.00 (1.00)		**78.27 (0.00)**		**1095.94 (0.00)**	

Table 2: Classification error, standard deviation, and runtime of all SVMs on the evaluation datasets for parameters $C = 1$, $C = 0.1$, and $C = 0.01$. The runtime T is given in seconds. The last line for each table depicts the F test value and the probability that the results are not statistical significant.

runtime for this approach is far to big to be feasible in practical situations. The mere GA based selection mutation switching between 0 and C converges much faster but is often less accurate. The remaining runtime differences between EvoSVM-S and the quadratic programming counterparts can surely be reduced by code optimization. The used SVM implementations are matured and have been optimized over the years whereas the implementations of the evolutionary approaches follow standard recipes without any code optimization.

The hybrid version EvoSVM-H combines the best elements of both worlds. It converges nearly as fast as the EvoSVM-S and is often nearly as accurate as the EvoSVM-G. In some cases (Ionosphere, Lupus) it even outperforms all other SVMs.

PsoSVM on the other hand does not provide the best performance in terms of classification error. Compared to the other evolutionary approaches, however, it converged much earlier than the other competitors.

Please note that the standard deviations of the errors achieved with the evolutionary SVMs are similar to the standard deviations achieved with mySVM or LibSVM. We can therefore conclude that the evolutionary optimization is as robust as the quadratic programming approaches and differences mainly derives from different subsets for training and testing due to cross validation instead of the used randomized heuristics.

Therefore, evolutionary SVMs provide an interesting alternative to more traditional SVM implementations. Beside the similar results EvoSVM is also able to cope with non-positive definite kernel functions and multivariate optimization.

6. CONCLUSION

In this paper we connected evolutionary computation with statistical learning theory. The idea of large margin methods was very successful in many applications from machine learning and data mining. We used the most prominent representative of this paradigm, namely Support Vector Machines, and employed evolution strategies and particle swarm optimization in order to solve the constrained optimization problem at hand. We developed a hybrid mutation which decreases convergence time while the classification accuracy is preserved.

An interesting property of large margin methods is that the runtime for fitness evaluation is reduced by transforming the problem into the dual problem. In our case, the algorithm is both faster and provides space for other improvements like incorporating a kernel function for non-linear classification tasks. This is a nice example how a transformation into the dual optimization problem can be exploited by evolutionary algorithms.

We have seen that evolutionary SVMs are at least as accurate as their quadratic programming counterparts. For practical values of C the evolutionary SVM variants frequently outperformed their competitors. We can conclude that evolutionary algorithms proved as reliable as other optimization schemes for this type of problems. In addition, beside the inherent advantages of evolutionary algorithms (e. g. parallelization, multi-objective optimization of train-

ing error and capacity) it is now also possible to employ non positive semidefinite kernel functions which would lead to unsolvable problems for other optimization techniques. In our future work we plan to make experiments with such non positive semidefinite kernel functions. This also applies for multi-objective optimization of both the margin and the training error.

It turns out that the hybrid mutation delivers results nearly as accurate as the Gaussian mutation and has a similar convergence behavior compared to the switching mutation known from GAs. Future improvements could start with a switching mutation and can post-optimize with a Gaussian mutation after a first convergence. Values always remaining 0 or C during the first run could be omitted in the post-optimization step. It is possible that this mutation is even faster and more accurate then EvoSVM-H.

7. ACKNOWLEDGMENTS

This work was supported by the *Deutsche Forschungsgemeinschaft (DFG)* within the *Collaborative Research Center "Reduction of Complexity for Multivariate Data Structures"*.

8. REFERENCES

[1] H.-G. Beyer and H.-P. Schwefel. Evolution strategies: A comprehensive introduction. *Journal Natural Computing*, 1(1):2–52, 2002.

[2] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[3] G. Camps-Valls, J. Martin-Guerrero, J. Rojo-Alvarez, and E. Soria-Olivas. Fuzzy sigmoid kernel for support vector classifiers. *Neurocomputing*, 62:501–506, 2004.

[4] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001.

[5] S. Fischer, R. Klinkenberg, I. Mierswa, and O. Ritthoff. Yale: Yet Another Learning Environment – Tutorial. Technical Report CI-136/02, Collaborative Research Center 531, University of Dortmund, Dortmund, Germany, 2002.

[6] F. Friedrichs and C. Igel. Evolutionary tuning of multiple svm parameters. In *Proc. of the 12th European Symposium on Artificial Neural Networks (ESANN 2004)*, pages 519–524, 2004.

[7] H. Frþhlich, O. Chapelle, and B. Schölkopf. Feature selection for support vector machines using genetic algorithms. *International Journal on Artificial Intelligence Tools*, 13(4):791–800, 2004.

[8] B. Haasdonk. Feature space interpretation of svms with indefinite kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(4):482–492, 2005.

[9] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, 2001.

[10] T. Howley and M. Madden. The genetic kernel support vector machine: Description and evaluation. *Artificial Intelligence Review*, 2005.

[11] W. James and C. Stein. Estimation with quadratic loss. In *Proceedings of the Fourth Berkeley Symposium on Mathematics, Statistics and Probability*, pages 361–380, 1960.

[12] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11. MIT Press, Cambridge, MA, 1999.

[13] T. Joachims. A support vector method for multivariate performance measures. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 377–384, 2005.

[14] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proc. of the International Conference on Neural Networks*, pages 1942–1948, 1995.

[15] H.-T. Lin and C.-J. Lin. A study on sigmoid kernels for svm and the training of non-psd kernels by smo-type methods, March 2003.

[16] D. Newman, S. Hettich, C. Blake, and C. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[17] C. Ong, X. Mary, S. Canu, and A. J. Smola. Learning with non-positive kernels. In *Proc. of the 21st International Conference on Machine Learning (ICML)*, pages 639–646, 2004.

[18] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.

[19] T. Runarsson and S. Sigurdsson. Asynchronous parallel evolutionary model selection for support vector machines. *Neural Information Processing*, 3(3):59–67, 2004.

[20] S. Rüping. *mySVM Manual*. Universität Dortmund, Lehrstuhl Informatik VIII, 2000. http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/.

[21] B. Schölkopf and A. J. Smola. *Learning with Kernels – Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[22] A. Smola, B. Schölkopf, and K.-R. Müller. General cost functions for support vector regression. In *Proceedings of the 8th International Conference on Artificial Neural Networks*, pages 79–83, 1998.

[23] A. J. Smola, Z. L. Ovari, and R. C. Williamson. Regularization with dot-product kernels. In *Proc. of the Neural Information Processing Systems (NIPS)*, pages 308–314, 2000.

[24] Statlib – datasets archive. http://lib.stat.cmu.edu/datasets/.

[25] T. Storch. On the impact of objective function transformations on evolutionary and black-box algorithms. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 833–840, 2005.

[26] B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin. Learning structured prediction models: A large margin approach. In *Proc. of the International Conference on Machine Learning (ICML)*, 2005.

[27] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[28] V. Vapnik and A. Chervonenkis. The necessary and sufficient conditions for consistency in the empirical risk minimization method. *Pattern Recognition and Image Analysis*, 1(3):283–305, 1991.