ABSTENTION REDUCES ERRORS —DECISION ABSTAINING N-VERSION GENETIC PROGRAMMING

Kosuke Imamura **Robert B. Heckendorn** **Terence Soule** **James A. Foster**

Initiative for Bioinformatics and Evolutionary STudies (IBEST),
Dept. of Computer Science,
University of Idaho, Moscow, ID 83844-1010
{kosuke,heckendo,tsoule,foster}@cs.uidaho.edu

Abstract

Optimal fault masking N-Version Genetic Programming (NVGP) is a technique for building fault tolerant software via ensemble of automatically generated modules in such a way as to maximize their collective fault masking ability. Decision Abstaining N-Version Genetic Programming is NVGP that abstains from decision-making, when there is no decisive vote among the modules to make a decision. A special course of action may be taken for an abstained instance. We found that decision abstention contributed to error reduction in our experimental *Escherichia coli* DNA promoter sequence classification problem. Though decision abstention may reduce errors, high abstention rate makes the system of little use. This paper investigates the trade-off between abstention rate and error reduction.

1 INTRODUCTION

This paper investigates the effect of an abstention threshold on the trade-off between abstention rate and error reduction, using an N-Version Genetic Programming ensemble classifier [1].

An ensemble binary classifier makes a yes/no decision based on votes from the participating ensemble member classifiers. A decision abstention occurs, when there is no decisive vote among the ensemble modules to make decision. An unanimous vote is the most decisive (highest ensemble confidence), while a tie vote is the least decisive (lowest ensemble confidence). The abstention threshold is set somewhere between these two extremes. If the vote count of either "yes" or "no" does not reach to this threshold, the ensemble abstains from decision-making. The ensemble, thus, produces three outputs: *yes*, *no*, and *don't know*. A special course of action may be taken for an abstained instance (such as classification by human experts) [2]. Abstention reduces the number of errors, potentially avoiding overfitting [2]. However, if the ensemble classifier abstains too often, it is of little use. Our experimental test problem is *Escherichia coli* DNA promoter sequence classification. This problem has been explored with artificial neural networks [3][4][5] and genetic programming [6].

1.1 BRIEF INTRODUCTION OF N-VERSION GENETIC PROGRAMMING (NVGP)

N-Version Genetic Programming (NVGP), which provides an optimal fault masking ensemble of automatically generated modules, is a new technique for building fault tolerant software that significantly reduces errors when applied to an *E. coli* promoter sequence classification problem [1]. Genetic programming is used to provide a large pool of candidate modules with sufficient diversity to allow us to select an ensemble whose faults are nearly uncorrelated. We find ensembles with a high degree of fault masking by randomly sampling from this large pool of modules. The ensembles that produce the expected error rate are retained. The expected failure rate f for n independent components, each of which fails with probability p, where the composite system requires m component faults in order to fail (initially derived for n-modular redundant hardware systems [7]) is

$$f = \sum_{k=m}^{n} \binom{n}{k} \left((1-p)^{n-k} p^{k}\right).$$

For an N-version classifier system, such as ours, the i-th individual fault rate p_i is the ratio of misclassified examples to the total number of training instances. In this case, f is the failure rate of an ideal ensemble. If the fault rate is the same for every p_i, f is an area under a binomial probability density function as shown in the above formula. The failure rate of an ensemble is close to the theoretically optimal rate f precisely when component failures are not correlated. This is our criterion for selecting the *qualified* ensembles. Explicit quantification of the module diversity with the theoretical failure probability is a distinct feature of NVGP, guaranteeing phenotypic diversity and the optimal ensemble.

N-version programming (NVP) was an early approach to building fault tolerant software that adapted proven hardware approaches to fault tolerance [8]. A fundamental assumption of the NVP approach was that independent programming efforts would reduce the probability that similar errors will occur in two or more versions [9]. But this assumption was questioned, because experiments showed that modules developed for NVP tended to fail under similar circumstances. For example, Knight and Leveson rejected the hypothesis of the assumed independence of faults by independently developed programs [10]. However, this conclusion does not invalidate NVP in general. Hatton showed that his 3-version NVP system increased the reliability by a factor of 45. Though this is far less than the theoretical improvement of a factor of 833.6, it is still a significant improvement in system reliability [11].

The next section reviews previous work on abstention and ensemble methods.

2 PREVIOUS WORK

Different ensemble construction methods have been studied in an effort to enhance accuracy. This section reviews abstention, averaging, median selection, boosting, and bagging. All methods exploit heterogeneity of ensemble components in one way or another.

2.1 ABSTENTION

Freund, et al., showed the error bound of averaging classifier with abstention [2]. The abstention threshold is based on the log ratio of the weighted sum of positive predictions and negative predictions. If the absolute value of this log ratio is smaller than the threshold, the ensemble classifier abstains from predicting. They identify the region of abstention as the locations of potential overfitting. Their theoretical work shows that the error of such predictor cannot be worse than twice of the best individual.

2.2 AVERAGE AND MEDIAN

A simple averaging method gathers outputs from all component modules and calculates their arithmetic average. Imamura and Foster showed simple averaging reduces error margins in path prediction [12] and function approximation with evolved digital circuits [13]. Another approach is weighted averaging, in which component modules are assigned optimal weights for computing a weighted average of the module outputs. Linearly optimal combination of artificial neural networks takes this approach [14][15]. Zang and Joung proposed Mixing Genetic Programs (MGP). MGP chooses a pool of individuals from a population and the master unit assigns the voting weights to these individuals using an additive weighting scheme [16]. The median value of the outputs is then the ensemble output. Soule approximated the sine function by taking the median of individuals, which were evolved, with subset of the entire training set for specialization [17]. Brameier and Banzhaf evolved teams of predictors. The individuals are coevolved as a team as opposed to post-evolutionary combination [18].

2.3 BOOSTING AND BAGGING

Boosting and bagging are methods that perturb the training data by resampling to induce classifier diversity. The `AdaBoost` algorithm trains a weak learner (slightly better than random guessing) by iterating training while increasing the weights of misclassified samples and decreasing the weights of correctly classified ones [19]. The effect is that the weak learner focuses more and more on the misclassified samples. The trained classifiers in each successive round are weighted according to their performance. The final decision is a weighted majority vote. Bagging (Bootstrap aggregating) replicates training subsets by sampling with replacement [20]. It then trains classifiers separately on these subsets and builds an ensemble by aggregating these individual classifiers. For evolutionary computation, Iba applied Boosting and Bagging to genetic programming and his experiment validated their effectiveness and their potential for controlling bloat [21]. Land used a boosting technique to improve performance of Evolutionary Programming derived neural network architectures in a breast cancer diagnostic application [22]. However, both techniques have limitations. Boosting is susceptible to noise, Bagging is not any better than a simple ensemble in some cases, neither Boosting nor Bagging is appropriate for data poor cases, and bootstrap methods can have a large bias [19, 23, 24, 25, 26, 27].

2.4 CLASSIFICATION OF ENSEMBLES

Table 1 categorizes current ensemble methods in genetic programming in terms of their sampling technique in combination with the evolutionary approach. In cooperative methods [17][28], speciation pressure (such as that caused by crowding penalties [28]) plays a vital role in evolving heterogeneous individuals, while in isolation methods there is no interaction between individuals during evolution. Resampling methods create different classifiers by using different training sets (bagging) or varying weights of training instances (boosting). Non-resampling methods create different classifiers from the same training set with or without explicit speciation pressure. NVGP and Decision abstaining NVGP are non-resampling techniques based on isolated evolution of diverse individuals.

Table 1. Classification of ensemble creation methods.

Evolutionary Approach	Training set selection	
	Resampling	**Non-resampling**
Non-Isolation	Boosting	Crowding
Isolation	Bagging	NVGP

3 EXPERIMENT

This section briefly summarizes the NVGP computational method (for detail description, see [1]) and presents the test results. We first run NVGP then incorporate abstention threshold to it. Our experimental problem is to classify whether a given DNA sequence is an *E. coli* promoter, using a decision abstaining NVGP. The data set is taken from UCI ML repository [29]. It contains 53 *E. coli* DNA promoter sequences and 53 non-promoter sequences of length 68.

3.1 COMPUTING ENVIRONMENT

The cluster supercomputing facilities from the Initiative for Bioinformatics and Evolutionary STudies (IBEST) is used to implement distributed computation. This device uses commodity computing parts to build substantial computing power for considerably less money than traditional supercomputers[1]. (http: //www.cs.uidaho.edu/ thecollective). This machine enabled experiments that would normally run for a month to complete in half a day.

3.2 INPUT AND OUTPUT

We used 2-gram encoding for input [30]. The 2-gram encoding counts the occurrences of two consecutive input characters (nucleotides) in a sliding window. Since there are four characters in DNA sequences (“a”, “c”, “g”, “t”), we have 16 unique two-character strings to count. For example, a sequence “caaag” will be encoded as {ca=1, aa=2, ag=1}. The classifier clusters the positive instances and places the negative instances outside the cluster. The cluster is defined by the mean output value of postitve instances ± 3*(standard deviation). If an output value from a given sequence falls in the cluster, it is classified as a promoter. Otherwise, it is classified as a non-promoter.

3.3 CLASSIFIER

3.3.1 Target Machine Architecture

Our classifier is a linear genome machine [31], which mimics MIPS architecture [32]. There are two instruction formats in this architecture: `(Opcode r1,r2,r3)` and `(Opcode r1,r2,data)`. The instructions are ADDI, ADDR, MUL, DIV, MULI, DIVI, SIN, COS, LOG, EXP, NOP, MOVE, LOAD, CJMP, and CJMPI. The length of an individual program is restricted to a maximum of 80 instructions. Each evolving individual (a potential component for our NVGP ensemble system) used sixteen read-only registers for input data, which contained counts for individual nucleotide 2-grams as described above, and four read/write working registers.

[1] The total cost of the machine is about US$44,000. Micron Technology generously donated all of the memory for the machine.

3.3.2 Genetic Programming

We used 5 crossover methods. Methods (1) and (2) are traditional one and two point crossover, respectively. Method (3) is one point crossover with inversion applied to each crossover segment. Methods (4) and (5) use four random crossover points, with (5) being a single parent recombination operator. Fitness is calculated by the following correlation formula

$$C = \frac{PN - P_f N_f}{\sqrt{(N+N_f)(N+P_f)(P+N_f)(P+P_f)}}$$

where P and N are numbers of correctly identified positives and negatives, and P_f and N_f are the numbers of falsely identified positives and negatives [33]. Steady state is used for population replacement. Evolution continues until an individual of fitness 0.8 or above appears.

3.3.3 Evolution and Ensemble Testing

A common holdout test divides the dataset into 2 exclusive sets, 2/3 for the training set and 1/3 for the test set [27]. Our training sets used a random sample of 35 (53*2/3) positive and 35 negative examples, and used the remaining examples for the test sets. We performed experiments for 10 different holdout sets. The evolution and ensemble procedures are described below:

1. Create a training set and test set.
2. Evolve 40 isolated islands with 100 individuals each in parallel. Add an individual whose fitness is 0.8 from each island to a set B of single best versions.
3. Select N individuals by uniform-random sampling from B for N=15, 31 to form an NVGP ensemble. See 3.4.1 for the sampling frequency.
4. Evaluate the performance of each ensemble. If the ensemble is *qualified*, then retain it for a test set trial. Goto 3. The ensemble is *qualified* if the difference between the number of errors expected when versions have independent faults and the number of errors observed is small (less than one in our case).

3.4 EXPERIMENTAL RESULTS

The evolution and ensemble testing procedure described in section 3.3 is repeated for 10 different holdout tests in an attempt to reduce stochastic errors caused by sampling in performance estimation. We first show the performance of NVGP without abstention, then with abstention. We assume the number of errors have a normal distribution, since each test instance can be viewed as a Bernoulli trial [27].

3.4.1 Performance of NVGP

There are 40×10^9 and 27×10^7 possible ensembles to be formed respectively for 15 and 31 voter systems out of 40 candidate modules. Uniform random search sampled approximately 40×10^3 and 27×10^3 ensembles for 15 and 31 voter ensembles respectively, from which we selected qualified ensembles for statistics. Table 2 shows the numbers of qualified ensembles found for each test. For example, we found 23199 qualified 15-voter ensembles out of 40×10^3 samples for the test 1. Table 3 is the result of t-test on the null hypothesis that average performance of the ensembles and the single best versions is not significantly different. Table 4 is the result of F-test on the null hypothesis that standard error of the ensembles and the single best versions is not significantly different. Table 5 shows error reduction percentage observed in ensembles relative to the error rates of the single best versions in the set B (see 3.3.3). It represents the average error reduction achieved by NVGP over single modules produced by genetic programming.

Figure 1 presents the performance distribution intervals of the single best versions and the corresponding *N*-voter NVGP ensemble at a 90% limit. For each holdout test, we present statistics for the single best versions, and for each of the four NVGP ensembles (N=15, 31). For example, the leftmost bar in holdout test 1 is the performance distribution of the 40 single best versions, showing that the best is estimated to be 20% error and the worst to be 48%, with a mean of 34%. The middle bar is 15-voter and the rightmost bar is 31-voter ensembles.

Table 2. The number of sampled qualified-ensembles

	test1	*test2*	*test3*	*test4*	*test5*	*test6*	*test7*	*test8*	*test9*	*test10*
15-voter	23199	29370	11596	8601	15973	4267	30455	32141	13171	17279
31-voter	19650	27205	9910	3197	13778	814	27340	27340	7672	23113

Table 3. The result of t-test, degree of freedom .≅ 40 for all the test cases.

	test1	*test2*	*test3*	*test4*	*test5*	*test6*	*test7*	*test7*	*test9*	*test10*
15-voter	11.07	9.10	3.27	8.67	9.67	10.21	8.80	6.06	6.64	5.46
31-voter	14.08	10.47	2.14	8.17	10.54	13.53	9.30	5.78	6.62	6.94

Table 4. The result of F test on error rate standard deviations

	test1	*test2*	*test3*	*test4*	*test5*	*test6*	*test7*	*test7*	*test9*	*test10*
15voter	3.15	4.10	4.98	3.53	4.01	2.22	2.71	4.37	7.22	3.76
31-voter	7.17	6.47	12.03	9.36	10.89	4.71	8.13	24.11	47.44	8.07

Table 5. Percentage error reduction of NVGP relative to set of best individual in isolation

	test1	*test2*	*test3*	*test4*	*test5*	*test6*	*test7*	*test7*	*test9*	*test10*
15-voter	44	33	14	31	40	31	37	22	25	24
31-voter	56	38	9	29	43	42	39	21	25	31

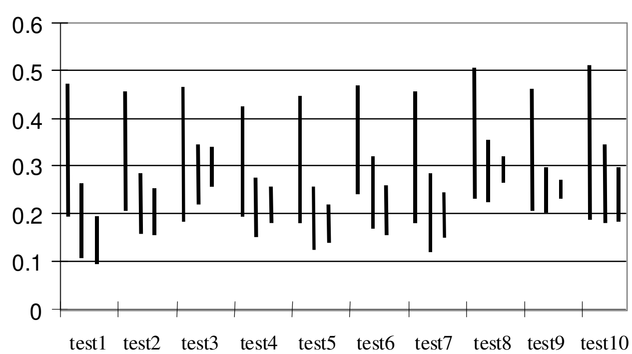

Figure 1. Error rate distribution intervals of the single best versions and the corresponding *N*-voter NVGP ensemble at a 90% limit. Leftmost, middle, and rightmost bars are distribution of single-version, 15-voter, and 31 voter system respectively.

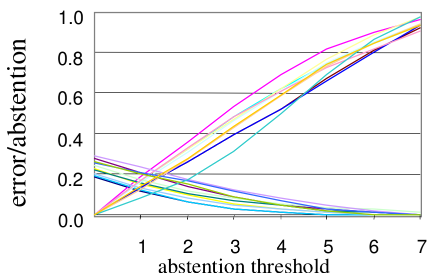

Figure 2. abstention rate and error rate for 15 voter system

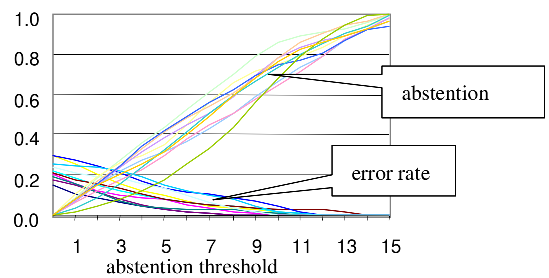

Figure 3. abstention rate and error rate for 31 voter system

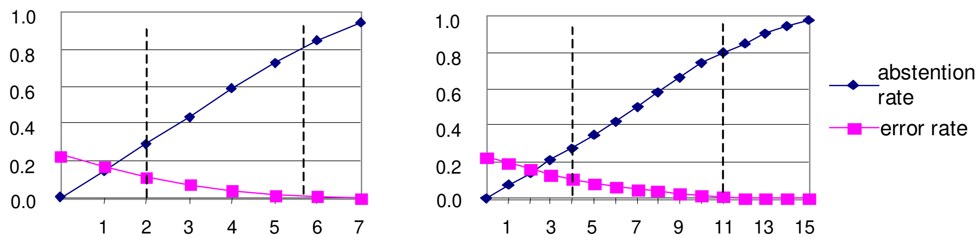

Figure 4. average abstention and error rates from figure 2 and 3.

3.4.2 Behavior of Decision Abstaining NVGP

The abstention thresholds are incorporated into the NVGP outputs of 3.4.1. The decision abstaining ensemble requires abstention threshold, h, and needs $((N+1)/2 + h)$ votes, either positive or negative, to make a decision, where N (odd) is the number of vote participating individuals. The voting scheme is a simple majority rule, if h=0. Figure 2 and 3 are the plots of the abstention rates and the error rates of 15 and 31 voter ensemble for the 10 holdout tests with respect to the abstention threshold, h. Figure 4 represents the average abstention and error rates from figure 2 and 3 for collective analysis. The error rate is a decreasing function and the abstention rate is an increasing function.

4 DISCUSSION

4.1 NVGP

Though a holdout test is commonly used to measure performance of evolutionary algorithms, it is not reliable. Kohavi argues that holdout testing does not provide a good estimate of error rate [27]. Nonetheless, we repeated the holdout test 10 times with different training/test sets for somewhat fair statistics. In figure 1, the hold-out test 3 does not exhibit apparent superiority of NVGP as in the test 1, though we reject the null hypothesis that average performance of single best version and NVGP are not significantly different at α=0.975. For all the other nine test cases, we reject the hypothesis virtually at 100% and conclude that NVGP is superior. NVGP error rates in all ten tests are far below the theoretical bound shown by Freund [2] even without abstention. Table 4 indicates that performance fluctuation of NVGP is statistically significantly smaller than single versions. Apparently, as the ensemble size approaches to the pool size, the performance fluctuation becomes smaller. If we combine all the individuals in the pool, there is no performance fluctuation. Therefore, a larger fluctuation may be expected for NVGP if the component pool size is huge. But, also true is that duplicate phenotypes start populating the pool as the pool size becomes larger. In fact, our experiment witnessed that an exhaustive search for an optimal ensemble of 39 voters from the pool failed in three out of the ten holdout tests. This possibly indicates that the entropy of the pool may have reached a plateau with the given training data and training method. If this is the case, the small performance fluctuation for *optimally sized* NVGP will still hold regardless of the pool size increase. Further study is needed for an optimal size of NVGP.

Notice that a single best individual has a chance to become practically a random classifier (error rate above 0.4) roughly 10%-20% of the time on unseen data. Unfortunately, we have no way of knowing which individual would become a random classifier beforehand, because they all have the same fitness (0.8) on the training set. This is the risk we must bear with a single best classifier. Fluctuation in performance is the very reason why we compared the distributions, and why NVGP has superior performance.

4.2 ABSTENTION

Figure 4 shows (see dashed lines) that the decision abstaining NVGP achieved a near zero error rate, at high

Table 6. Q values for 31-voter ensembles (ρ=0.5)

abstention threshold	*test1*	*test2*	*test3*	*test4*	*test5*	*test6*	*test7*	*test8*	*test9*	*test10*
0	6.7	8.0	10.1	7.7	6.8	8.8	7.3	10.4	9.1	9.5
1	7.2	8.3	10.0	7.9	7.1	9.1	7.6	10.6	9.0	9.7
2	8.5	9.2	9.8	8.4	7.8	9.9	8.6	11.1	9.1	10.2
3	10.5	10.5	10.1	9.5	9.3	11.2	10.1	12.2	9.8	11.0

cost of abstention rate, approximately 80%, for both 15-voter and 31-voter ensembles. Abstention rates 29% of 15-voter and 28% of 31-voter ensembles give 50% error reduction over NVGP alone (no abstention). Whether these abstention rates are acceptable for error reduction depends on how critical it is to have wrong predictions.

The abstention rates and the error rates are monotonic with respect to the abstention thresholds, and the trade-off between abstention and error reduction can be estimated almost linearly. Consequently, there is no analytically measurable peak gain by abstention. Subjective judgment must be used to set the abstention threshold. The following formula may be used to numerically measure the effect of abstention: $Q= E_a + \rho N$, where E_a is the number of errors with abstention, N is the number of *don't know* outputs, and $0 \le \rho \le 1$. If ρ=1, then *don't know* is as bad as wrong prediction and counted as an error. On the other extreme, if ρ=0, it is as good as correct prediction. The larger the ρ value is the more penalties for *don't know* outputs.

Let the number of errors of NVGP alone (no abstention) be E_z. If $Q \le E_z$, then we are unconditionally better off with abstention. For example, setting ρ=0.5 (half way between correct and incorrect prediction), we obtain Q values for 31-voter ensembles as shown in Table 6. The Q values are fairly close to E_z when the threshold is 1, which gives 3.2% error reduction (Figure 4 data). In other words, threshold = 1 is a break-even point for the trade-off between abstention and error reduction for ρ=0.5. For safety critical applications, such as medical diagnostics, a smaller ρ value would be appropriate for the trade-off analysis. That is to say, do not penalize heavily when an ensemble is trying to avoid a random guess. It may well be the case where the training set was inappropriate for particular instances.

4.3 POST-EVOLUTIONARY COMBINATION

Post-evolutionary combination is thought to be computationally inefficient, because many runs are required to obtain a sufficient number of individuals [18]. However, inexpensive cluster computing alleviates this problem (see section 3.1). Not only can the post-evolutionary search for the optimal NVGP ensemble be performed in parallel, but also the search may no longer need to be continued after an optimal ensemble is found.

5 CONCLUSION AND FUTURE RESEARCH

We showed the experimental classification result by NVGP, which significantly improved accuracy and reduced the performance fluctuation. Then, we incorporated decision abstention to it. Abstention in effect avoids random guesses when the ensemble confidence is low, i.e., votes are too close to call. It is a viable method to reduce errors. The trade-off between abstention and error reduction is subjective. The abstention threshold value depends on how critical an application is.

It is important to curve the abstention rate increase. We plan to embed the individual confidence to enhance the ensemble confidence. The individual confidence, in our case, can be measured by the distance of an instance from the cluster center. The further the distance, the lower the confidence. The ensemble confidence in prediction is measured by the level of disagreement among the voters.

Acknowledgments

This work is supported by the Initiative for Bioinformatics and Evolutionary STudies (IBEST) at the University of Idaho; by NIH NCRR grant 1P20RR016454-01; and by NIH NCRR grant NIH NCRR 1P20RR016448-01; and by NSF grant NSF EPS 809935. We are grateful to colleagues in the Initiative for Bioinformatics and Evolutionary STudies (IBEST) for insightful discussions. Finally, we wish to thank the many students who designed and built our Beowulf cluster, and Micron Technology for donating the memory for that machine. We thank John Cavalieri and Janet Holmberg for proofreading.

References

1. Imamura, K., Heckendorn,R B., Soule, T., Foster, J A.: N-version Genetic Programming via Fault Masking Proceedings of the Euro EuroGP2002 5th European Conference on Genetic Programming (To appear)

2. Freund, Y., Mansour, Y., Schapire, R E.: Why Averaging Classifiers Can Protect Against Overfitting. Proceedings of the 8th International Workshop on Artificial Intelligence and Statistics,2001 (www.ai.mit.edu/confereces/aistats2001/papers.html)

3. Pedersen, A.G., Engelbrecht, J.: Investigations of *Escherichia Coli* promoter sequences with artificial neural networks: New signals discovered upstream of the transcriptional startpoint. Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology (1995) 292-299 (http://citeseer.nj.nec.com/25393.html)
4. Towell,G.G., Shavlik, J.W., Noordewier, M.O.: Refinement of approximate domain theories by knowledge-based neural networks. Proceedings of AAAI-90 (1990) 861-866 (http://citeseer.nj.nec.com/towell90refinement.html)
5. Ma, Q., Wang, J.T.L.: Recognizing Promoters in DNA Using Bayesian Neural Networks. Proceedings of the IASTED International Conference, Artificial Intelligence and Soft Computing (1999) 301-305 (http://citeseer.nj.nec.com/174424.html)
6. Handley, S.: Predicting Whether Or Not a Nucleic Acid Sequence is an *E. Coli* Promoter Region Using Genetic Programming. Proceedings of First International Symposium on Intelligence in Neural and Biological Systems, IEEE Computer Society Press, (1995) 122-127
7. Pradhan, D. K., Banerjee, P.: Fault-Tolerance Multiprocessor and Distributed Systems: Principles. In Pradhan, D.K.: Fault-Tolerant Computer System Design. Chapter 3, Prentice Hall PTR, (1996), 142
8. Avizienis, A. and J.P.J. Kelly: Fault Tolerance by Design Diversity: Concepts and Experiments. IEEE Computer, vol. 17 no. 8, (1984), 67-80
9. Victoria Hilford., Lyu, M. R., Cukic B., Jamoussi A., Bastani F. B.: Diversity in the Software Development Process. Proceedings of Third International Workshop on Object-Oriented Real-Time Dependable Systems, IEEE Comput. Soc, (1997), 129-36 (http://www.cse.cuhk.edu.hk/~lyu/papers.html#SFT_Techniques)
10. Knight, J.C., Leveson, N.B.: An Experimental Evaluation of the Assumption of Independence in Multiversion Programming. IEEE Transaction on Software Engineering, vol. SE-12, no. 1 (1986)
11. Hatton, L.: N-version vs. one good program. IEEE Software, vol 14, no. 6 (1997) 71-76
12. Imamura, K., Foster, J.A.: Fault Tolerant Computing with N-Version Genetic Programming. Proceedings of Genetic and Evolvable Computing Conference (GECCO), Morgan Kaufmann, (2001) 178
13. Imamura, K., Foster, J.A.: Fault Tolerant Evolvable Hardware Through N-Version Genetic Programming. Proceedings of World Multiconference on Systemics, Cybernetics, and Informatics (SCI), vol. 3, (2001) 182-186
14. Hashem, S.: Optimal Linear Combinations of Neural Networks. Neural_Networks, vol. 10, no. 4, (1997) 599-614 (http://www.emsl.pnl.gov:2080/proj/neuron/papers/hashem.nn97.abs.html)
15. Hashem, S.: Improving Model Accuracy Using Optimal Linear Combinations of Trained Neural Networks. IEEE Transactions on Neural Networks, vol.6, no.3 (1995) 792-794 (www.emsl.pnl.gov:2080/proj/neuron//papers/hashem.tonn95.abs.html)
16. Zang, B-T., Joung, J-G.: Enhancing Robustness of Genetic Programming at the Species Level. Proceedings of the 2nd Annual Conference Genetic Programming 97, Morgan Kaufmann (1997) 336-342.
17. Terence Soule, “Heterogeneity and Specialization in Evolving Teams”, Proceeding of Genetic and Evolvable Computing Conference (GECCO), Morgan Kaufmann (2000) 778-785
18. Brameier, M., Banzhaf, W.:Evolving Teams of Predictors with Linear Genetic Programming. Genetic Programmin and Evolvable Machines, vol. 2, (2001) 381-407
19. Schapire R.E., Freund, F.: A Short Introduction to Boosting. Journal of Japanese Society for Artificial Intelligence 14, no. 5, (1999) 771-80 (http://citeseer.nj.nec.com/freund99short.html)
20. Breiman, L.: Bagging Predictor. Technical Report No.421, Department of Statistics, University of California Berkley, 1994 (http://www.salford-systems.com/docs/BAGGING_PREDICTORS.PDF)
21. Iba, H.: Bagging, Boosting, and Bloating in Genetic Programming. Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2, Morgan Kaufmann, (1999) 1053-1060
22. Land, W.H. Jr., Masters T., Lo J.Y., McKee, D.W., Anderson, F.R.: New results in breast cancer classification obtained from an evolutionary computation/adaptive boosting hybrid using mammogram and history data. Proceedings of the 2001 IEEE Mountain Workshop on Soft Computing in Industrial Applications. IEEE, (2001) 47-52
23. Basak, S.C., Gute, B.D., Grunwald, G.D., David W. Opitz, D.W., Balasubramanian, K.: Use of statistical and neural net methods in predicting toxicity of chemicals: A hierarchical QSAR approach. Predictive Toxicology of Chemicals: Experiences and Impact of AI Tools - Papers from the 1999 AAAI Symposium, AAAI Press, (1999) 108-111
24. Opitz, D.W., Basak, S.C., Gute, B.D.: Hazard Assessment Modeling: An Evolutionary Ensemble Approach. Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2, Morgan Kaufmann (1999) 1643-1650
25. Maclin, R., Opitz, D.: An empirical evaluation of bagging and boosting. Proceedings of the Fourteenth International Conference on Artificial Intelligence, AAAI Press/MIT Press (1999) 546-551 (http://citeseer.nj.nec.com/maclin97empirical.html)

26. Bauer, E., Kohavi, R.: An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. Machine Learning, vol. 36, 1/2, Kluwer Academic Publishers (1999) 105-139 (http://citeseer.nj.nec.com/bauer98empirical.html)
27. Kohavi, R.: A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI), Morgan Kaufmann (1995) 1137-1145 (http://citeseer.nj.nec.com/kohavi95study.html)
28. Soule, T.: Voting Teams: A Cooperative Approach to Non-Typical Problems. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99), vol. 1,Morgan Kaufmann (1999) 916-922
29. UCI Machine Learning Repository, Molecular Biology Databases (http://www1.ics.uci.edu/~mlearn/MLSummary.html)
30. Wang, J.T.L., Ma, Q., Shash D., Wu, C.: Application of neural networks to biological data mining: a case study in protein sequence classification. Proceedings KDD-2000. Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, (2000) 305-309 (http://citeseer.nj.nec.com/382372.html)
31. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications. Academic Press/Morgan Kaufmann (1998)
32. MIPS32™ Architecture for Programmers Volume I: Introduction to the MIPS32™ Architecture (http://www.mips.com/publications/index.html)
33. Matthwes, B. W.:Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochimica et Biophysica Acta, vol. 405 (1975) 443-451