Feature Selection for Ensembles

David W. Opitz
Computer Science Department
University of Montana
Missoula, MT 59812
opitz@cs.umt.edu

Abstract

The traditional motivation behind feature selection algorithms is to find the best subset of features for a task using one particular learning algorithm. Given the recent success of ensembles, however, we investigate the notion of *ensemble feature selection* in this paper. This task is harder than traditional feature selection in that one not only needs to find features germane to the learning task and learning algorithm, but one also needs to find a set of feature subsets that will promote disagreement among the ensemble's classifiers. In this paper, we present an ensemble feature selection approach that is based on genetic algorithms. Our algorithm shows improved performance over the popular and powerful ensemble approaches of AdaBoost and Bagging and demonstrates the utility of ensemble feature selection.

Introduction

Feature selection algorithms attempt to find and remove the features which are unhelpful or destructive to learning (Almuallim & Dietterich 1994; Cherkauer & Shavlik 1996; Kohavi & John 1997). Previous work on feature selection has focused on finding the appropriate subset of relevant features to be used in constructing *one* inference model; however, recent "ensemble" work has shown that combining the output of a *set* of models that are generated from separately trained inductive learning algorithms can greatly improve generalization accuracy (Breiman 1996; Maclin & Opitz 1997; Shapire *et al.* 1997). This paper argues for the importance of and presents an approach to the task of *feature selection for ensembles*.

Research has shown that an effective ensemble should consist of a set of models that are not only highly correct, but ones that make their errors on different parts of the input space as well (Hansen & Salamon 1990; Krogh & Vedelsby 1995; Opitz & Shavlik 1996). Varying the feature subsets used by each member of the ensemble should help promote this necessary diversity. Thus, while traditional feature-selection algorithms have the goal of finding the best feature subset that is germane to both the learning task and the selected inductive-learning algorithm, the task of ensemble feature selection has the additional goal of finding a *set* of features subsets that will promote disagreement among the component members of the ensemble. This search space is enormous for any non-trivial problem.

In this paper, we present a genetic algorithm (GA) approach for searching for an appropriate set of feature subsets for ensembles. GAs are a logical choice since they have been shown to be effective global optimization techniques (Holland 1975; Mitchell 1996). Our approach works by first creating an initial population of classifiers where each classifier is generated by randomly selecting a different subset of features. We then continually produce new candidate classifiers by using the genetic operators of crossover and mutation on the feature subsets. Our algorithm defines the overall fitness of an individual to be a combination of accuracy and diversity. The most fit individuals make up the population which in turn comprise the ensemble. Using neural networks as our classifier, results on 21 datasets show that our simple and straight-forward algorithm for creating the initial population produces better ensembles on average than the popular and powerful ensemble approaches of Bagging and Boosting. Results also show that further running the algorithm with the genetic operators improves performance.

Review of Ensembles

Figure 1 illustrates the basic framework of a predictor ensemble. Each predictor in the ensemble (predictor 1 through predictor N in this case) is first trained using the training instances. Then, for each example, the predicted output of each of these predictors (o_i in Figure 1) is combined to produce the output of the ensem-

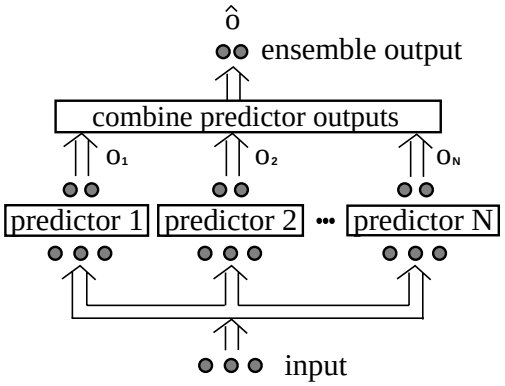

Figure 1: A predictor ensemble.

ble ($\hat{o}$ in Figure 1). Many researchers (Breiman 1996; Hansen & Salamon 1990; Opitz & Shavlik 1997) have demonstrated the effectiveness of combining schemes that are simply the weighted average of the predictors (i.e., $\hat{o} = \sum_{i \in N} w_i \cdot o_i$ and $\sum_{i \in N} w_i = 1$); this is the type of ensemble on which we focus in this paper.

Combining the output of several predictors is useful only if there is disagreement on some inputs. Obviously, combining several identical predictors produces no gain. Hansen and Salamon 1990 proved that for an ensemble, if the average error rate for an example is less than 50% and the predictors in the ensemble are independent in the production of their errors, the expected error for that example can be reduced to zero as the number of predictors combined goes to infinity; however, such assumptions rarely hold in practice. Krogh and Vedelsby 1995 later proved that the ensemble error can be divided into a term measuring the average generalization error of each individual classifier and a term measuring the disagreement among the classifiers. What they formally showed was that an ideal ensemble consists of highly correct classifiers that disagree as much as possible. Numerous authors have empirically verified that such ensembles generalize well (e.g., Opitz & Shavlik 1996; Breiman 1996a; Freund 1996).

As a result, methods for creating ensembles center around producing predictors that disagree on their predictions. Generally, these methods focus on altering the training process in the hope that the resulting predictors will produce different predictions. For example, neural network techniques that have been employed include methods for training with different topologies, different initial weights, different parameters, and training only on a portion of the training set (Breiman 1996; Freund & Schapire 1996; Hansen & Salamon 1990). *Varying the feature subsets to create a diverse set of accurate predictors is the focus of this article.*

Numerous techniques try to generate disagreement among the classifiers by altering the training set each classifier sees. The two most popular techniques are Bagging (Breiman 1996) and Boosting (particularly AdaBoost; Freund & Schapire 1996). Bagging is a bootstrap ensemble method that trains each predictor in the ensemble with a different partition of the training set. It generates each partition by randomly drawing, with replacement, N examples from the training set, where N is the size of the training set. Breiman 1996 showed that Bagging is effective on "unstable" learning algorithms (such as neural network) where small changes in the training set result in large changes in predictions. As with Bagging, AdaBoost also chooses a training set of size N and initially sets the probability of picking each example to be $1/N$. After the first predictor, however, these probabilities change as follows. Let ϵ_k be the sum of the misclassified instance probabilities of the currently trained classifier C_k. AdaBoost generates probabilities for the next trial by multiplying the probabilities of C_k's incorrectly classified instances by the factor $\beta_k = (1 - \epsilon_k)/\epsilon_k$ and then re-normalizing these probabilities so that their sum equals 1. AdaBoost then combines the classifiers $C_1, \ldots, C_k$ using weighted voting where C_k has weight $\log(\beta_k)$. Numerous empirical studies have shown that Bagging and Boosting are highly successful methods that usually generalize better than their base predictors (Bauer & Kohavi 1998; Maclin & Opitz 1997; Quinlan 1996); thus, we include these two methods as baselines for our study.

Ensemble Feature Selection

Kohavi & John 1997 showed that the efficacy of a set of features to learning depends on the learning algorithm itself; that is, the appropriate feature subset for one learning algorithm may not be the appropriate feature subset for another learner. Kohavi & John's 1997 wrapper approach works by conducting a search through the space of possible feature subsets, explicitly testing the accuracy of the learning algorithm on each search node's feature subset. The search space of feature subsets is enormous and quickly becomes impractical to do hill-climbing searches (the traditional wrapper search technique) with slow-training learners such as neural networks; this search space is even larger when considering an appropriate set of feature subsets for ensembles. This is why we consider a global search technique (GAs) in this paper.

The notion of feature selection for ensembles is new. Other researchers have investigated using GAs for feature selection (e.g., Cherkauer & Shavlik 1996; Guo & Uhrig 1992); however, they have only looked at it from the aspect of traditional feature selection – find-

ing one appropriate set for learning – rather than from the ensemble perspective.

The GEFS Algorithm

Table 1 summarizes our algorithm (called GEFS for Genetic Ensemble Feature Selection) that uses GAs to generate a set of classifiers that are accurate and diverse in their predictions. (We focus on neural networks in this paper; however, GEFS can be easily extended to other learning algorithms as well.) GEFS starts by creating and training its initial population of networks. The representation of each individual of our population is simply a dynamic length string of integers, where each integer indexes a particular feature. We create networks from these strings by first having the input nodes match the string of integers, then creating a standard single-hidden-layer, fully connected neural network. Our algorithm then creates new networks by using the genetic operators of crossover and mutation.

GEFS trains these new individuals using backpropogation. It adds new networks to the population and then scores each population member with respect to its prediction accuracy and diversity. GEFS normalizes these scores, then defines the fitness of each population member (i) to be:

$$Fitness_i = Accuracy_i + \lambda \; Diversity_i \qquad (1)$$

where λ defines the tradeoff between accuracy and diversity. Finally, GEFS prunes the population to the N most-fit members, then repeats this process. At every point in time, the current ensemble consists of simply averaging (with equal weight) the predictions of the output of each member of the current population. Thus as the population evolves, so does the ensemble.

We define accuracy to be network i's training-set accuracy. (One may use a validation-set if there are enough training instances). We define diversity to be the average difference between the prediction of our component classifier and the ensemble. We then separately normalize both terms so that the values range from 0 to 1. Normalizing both terms allows λ to have the same meaning across domains.

It is not always clear at what value one should set λ; therefore, we automatically adjust λ based on the discrete derivatives of the ensemble error $\hat{E}$, the average population error $\bar{E}$, and the average diversity $\bar{D}$ within the ensemble. First, we never change λ if $\hat{E}$ is decreasing; otherwise we (a) increase λ if $\bar{E}$ is not increasing and the population diversity $\bar{D}$ is decreasing; or (b) decrease λ if $\bar{E}$ is increasing and $\bar{D}$ is not decreasing. We started λ at 1.0 for the experiments in this article. The amount λ changes is 10% of its current value.

Table 1: The GEFS algorithm.

GOAL: Find a set of input subsets to create an accurate and diverse classifier ensemble.

1. Using varying inputs, create and train the initial population of classifiers.
2. Until a stopping criterion is reached:
   (a) Use genetic operators to create new networks.
   (b) Measure the diversity of each network with respect to the current population.
   (c) Normalize the accuracy scores and the diversity scores of the individual networks.
   (d) Calculate the fitness of each population member.
   (e) Prune the population to the N fittest networks.
   (f) Adjust λ.
   (g) The current population composes the ensemble.

We create the initial population by first randomly choosing the number of features to include in each feature subset. For classifier i, the size of each feature subset (N_i) is independently chosen from a uniform distribution between 1 and twice the number of original features in the dataset. We then randomly pick, with replacement, N_i features to include in classifier i's training set. Note that some features may be picked multiple times while other may not be picked at all; replicating inputs for a neural network may give the network a better chance to utilize that feature during training. Also, replicating a feature in a genome encoding allows that feature to better survive to future generations.

Our crossover operator uses dynamic-length, uniform crossover. In this case, we chose the feature subsets of two individuals in the current population proportional to fitness. Each feature in both parent's subset is independently considered and randomly placed in the feature set of one of the two children. Thus it is possible to have a feature set that is larger (or smaller) than the largest (or smallest) of either parent's feature subset. Our mutation operator works much like traditional genetic algorithms; we randomly replace a small percentage of a parent's feature subset with new features. With both operators, the network is trained from scratch using the new feature subset; thus no internal structure of the parents are saved during the crossover.

Since GEFS continually considers new networks to include in its ensemble, it can be viewed as an "anytime" learning algorithm. Such a learning algorithm

Table 2: Summary of the data sets used in this paper. Shown are the number of examples in the data set; the number of output classes; the number of continuous and discrete input features; the number of input, output, and hidden units used in the neural networks tested; and how many epochs each neural network was trained.

			Features		Neural Network			
Dataset	Cases	Classes	Continuous	Discrete	Inputs	Outputs	Hiddens	Epochs
credit-a	690	2	6	9	47	1	10	35
credit-g	1000	2	7	13	63	1	10	30
diabetes	768	2	9	-	8	1	5	30
glass	214	6	9	-	9	6	10	80
heart-cleveland	303	2	8	5	13	1	5	40
hepatitis	155	2	6	13	32	1	10	60
house-votes-84	435	2	-	16	16	1	5	40
hypo	3772	5	7	22	55	5	15	40
ionosphere	351	2	34	-	34	1	10	40
iris	159	3	4	-	4	3	5	80
kr-vs-kp	3196	2	-	36	74	1	15	20
labor	57	2	8	8	29	1	10	80
letter	20000	26	16	-	16	26	40	30
promoters-936	936	2	-	57	228	1	20	30
ribosome-bind	1877	2	-	49	196	1	20	35
satellite	6435	6	36	-	36	6	15	30
segmentation	2310	7	19	-	19	7	15	20
sick	3772	2	7	22	55	1	10	40
sonar	208	2	60	-	60	1	10	60
soybean	683	19	-	35	134	19	25	40
vehicle	846	4	18	-	18	4	10	40

should produce a good concept quickly, then continue to search concept space, reporting the new "best" concept whenever one is found (Opitz & Shavlik 1997). This is important since, for most domains, an expert is willing to wait for weeks, or even months, if a learning system can produce an improved concept.

GEFS is inspired by our previous approach of applying GAs to ensembles called ADDEMUP (Opitz & Shavlik 1996); however, the algorithms are quite different. ADDEMUP is far more complex, does not vary its inputs, and its genetic operators were designed explicitly for hidden nodes in *knowledge-based* neural networks; in fact ADDEMUP does not work well with problems lacking prior knowledge.

Results

To evaluate the performance of GEFS, we obtained a number of data sets from the University of Wisconsin Machine Learning repository as well as the UCI data set repository (Murphy & Aha 1994). These data sets were hand selected such that they (a) came from real-world problems, (b) varied in characteristics, and (c) were deemed useful by previous researchers. Table 2 gives the characteristics of our data sets. The data sets chosen vary across a number of dimensions including: the type of the features in the data set (i.e., continuous, discrete, or a mix of the two); the number of output classes; and the number of examples in the data set. Table 2 also shows the architecture and training parameters used with our neural networks.

All results are averaged over five standard 10-fold cross validation experiments. For each 10-fold cross validation the data set is first partitioned into 10 equal-sized sets, then each set is in turn used as the test set while the classifier trains on the other nine sets. For each fold an ensemble of 20 networks is created (for a total of 200 networks for each 10-fold cross validation). We trained the neural networks using standard backpropagation learning. Parameter settings for the neural networks include a learning rate of 0.15, a momentum term of 0.9, and weights are initialized randomly to be between 0.5 and -0.5. Table 2 also shows the architecture and training parameters used in our neural networks experiments. We chose the number of hidden units based on the number of input and output units. This choice was based on the criteria of having at least one hidden unit per output, at least one hidden unit for every ten inputs, and five hidden units being a minimum. Parameter settings for the GA portion of GEFS includes a mutation rate of 50%, a population

Table 3: Test set error rates for the data sets using (1) a single neural network classifier; (2) the Bagging ensemble method, (3) the AdaBoost ensemble method; (4) the ensemble of GEFS's initial population, and (5) GEFS run to consider 250 networks. The bottom of the table contains a win-loss-tie comparison between the learning algorithms on the datasets.

	Traditional			GEFS	
Dataset	Single Net	Bagging	AdaBoost	Initial Pop	100 networks
credit-a	14.8	13.8	15.7	13.6	13.1
credit-g	27.9	24.2	25.3	23.9	24.8
diabetes	23.9	22.8	23.3	24.5	23.0
glass	38.6	33.1	31.1	30.8	30.4
heart-cleveland	18.6	17.0	21.1	16.8	16.1
hepatitis	20.1	17.8	19.7	15.5	16.7
house-votes-84	4.9	4.1	5.3	3.9	4.4
hypo	6.4	6.2	6.2	7.5	5.9
ionosphere	9.7	9.2	8.3	6.3	5.4
iris	4.3	4.0	3.9	4.0	3.3
kr-vs-kp	2.3	0.8	0.3	3.0	0.7
labor	6.1	4.2	3.2	3.5	3.5
letter	18.0	10.5	4.6	10.3	9.5
promoters-936	5.3	4.0	4.6	4.9	4.3
ribosome-bind	9.3	8.4	8.2	7.9	7.8
satellite	13.0	10.6	10.0	14.2	11.2
segmentation	6.6	5.4	3.3	5.2	3.6
sick	5.9	5.7	4.5	6.1	3.5
sonar	16.6	16.8	13.0	17.3	17.8
soybean	9.2	6.9	6.3	6.0	5.9
vehicle	24.9	20.7	19.7	22.2	19.0
Single Net		**20-1-0**	**18-3-0**	**15-6-0**	**20-1-0**
Bagging			13-7-1	13-7-1	**15-6-0**
AdaBoost				9-12-0	14-7-0
Initial Pop					**16-4-1**

size of 20, a $\lambda = 1.0$, and a search length of 250 networks (note this is not 250 generations). While the mutation rate may seem high as compared with traditional GAs, certain aspects of our approach call for a higher mutation rate (such as the goal of generating a population that cooperates as well as our emphasis on diversity); other mutation values were tried during our pilot studies.

Table 3 shows the error rates for the algorithms on all the datasets. As points of comparison, we include the results of running the *Bagging* and *AdaBoost* algorithms. (We described these algorithms in the second section.) Two results are presented for the GEFS algorithm: (1) accuracy after the initial population was created (the population size was 20), and (2) accuracy after 250 trained networks are considered during the search (20 for the initial population plus 230 with our genetic operators). For the convenience of the reader, the bottom of the Table 3 contains a win-loss-tie comparison between the learning algorithms on the datasets. A comparison in bold means the difference in performance between the algorithms is statistically significant at the 95% confidence level when using the one-tailed sign test.

Discussion and Future Work

First, our results confirm earlier findings (Maclin & Opitz 1997; Quinlan 1996) that Bagging almost always produces a better classifier than a single neural network and that the AdaBoost method is a powerful technique that can usually produce better ensembles than Bagging; however, it is more susceptible to noise and can quickly overfit a data set.

We draw two main conclusions with our new algorithm's performance. First, GEFS can produce a good initial population and the algorithm for producing this population is both simple and fast. The fact that the initial population is competitive with both Bagging and AdaBoost is somewhat surprising. This shows that in many cases more diversity is created among the pre-

dictors by varying our feature set in this manner than is lost in individual predictor accuracy by not using the whole feature set.

The second conclusion we draw is that running GEFS longer usually increases performance. This is desirable since it allows the user to fully utilize available computer cycles to generate an improved model. Running AdaBoost and Bagging longer does not appreciably increase performance since previous results have shown their performance nearly fully asymptotes at around 20 networks; thus they do not appear to have the same ability to get better over time.

While GEFS's results are already impressive, we view this as just the first step toward ensemble feature selection. An important contribution of this paper is simply the demonstration of the utility for creating ensemble feature selection algorithms. Many improvements are possible and need to be explored. One area of future research is combining GEFS with AdaBoost's approach of emphasizing examples not correctly classified by the current ensemble. We also plan a further investigation of tuning parameters, such as the maximum size of the feature subsets in the initial population (results of such experiments are not presented in this paper due to limited space). Finally, we plan to investigate applying GEFS to other inductive learning algorithms such as decision trees and Bayesian Learning.

Conclusions

In this paper we have argued for the importance of feature detection for ensembles and presented such an algorithm, GEFS, that is based on genetic algorithms. Our ensemble feature selection approach is straightforward, simple, generates good results quickly, and has the ability to further increase its performance if allowed to run longer. Our results show that GEFS compared favorably with the two powerful ensemble techniques of AdaBoost and Bagging. Thus, this paper shows the utility of feature selection for ensembles and provides an important and effective first step in this direction.

Acknowledgments

This work was partially supported by National Science Foundation grant IRI-9734419 and a University of Montana MONTS grant.

References

Almuallim, H., and Dietterich, T. 1994. Learning Boolean concepts in the presence of many irrelevant features. *Artificial Intelligence* 69(1):279–305.

Bauer, E., and Kohavi, R. 1998. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*.

Breiman, L. 1996. Bagging predictors. *Machine Learning* 24(2):123–140.

Cherkauer, K., and Shavlik, J. 1996. Growing simpler decision trees to facilitate knowledge discovery. In *The Second International Conference on Knowledge Discovery and Data Mining*, 315–318. AAAI/MIT Press.

Freund, Y., and Schapire, R. 1996. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, 148–156. Morgan Kaufmann.

Guo, Z., and Uhrig, R. 1992. Using genetic algorithms to select inputs for neural networks. In *Int. Conf. on Genetic Algorithms and Neural Networks*, 223–234.

Hansen, L., and Salamon, P. 1990. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12:993–1001.

Holland, J. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: Univ of Michigan Press.

Kohavi, F., and John, G. 1997. Wrappers for feature subset selection. *Artificial Intelligence* 97(1):273–324.

Krogh, A., and Vedelsby, J. 1995. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems*, volume 7, 231–238. Cambridge, MA: MIT Press.

Maclin, R., and Opitz, D. 1997. An empirical evaluation of bagging and boosting. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 546–551. Providence, RI: AAAI/MIT Press.

Mitchell, M. 1996. *An Introduction to Genetic Algorithms*. MIT Press.

Murphy, P. M., and Aha, D. W. 1994. UCI repository of machine learning databases (machine-readable data repository). University of California-Irvine, Department of Information and Computer Science.

Opitz, D., and Shavlik, J. 1996. Actively searching for an effective neural-network ensemble. *Connection Science* 8(3/4):337–353.

Opitz, D., and Shavlik, J. 1997. Connectionist theory refinement: Searching for good network topologies. *Journal of Artificial Intelligence Research* 6:177–209.

Quinlan, J. R. 1996. Bagging, boosting, and c4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 725–730. AAAI/MIT Press.

Shapire, R.; Freund, Y.; Bartlett, P.; and Lee, W. 1997. Boosting the margin: A new explanation for the effectiveness of voting methods. In *Proceedings of the Fourteenth International Conference on Machine Learning*, 322–330. Morgan Kaufmann.