Heuristic Search-Based Stacking of Classifiers

Agapito Ledezma, Ricardo Aler and Daniel Borrajo
Universidad Carlos III de Madrid

Currently, the combination of several classifiers is one of the most active fields within inductive learning. Examples of such techniques are boosting, bagging and stacking. From these three techniques, stacking is perhaps the least used one. One of the main reasons for this relates to the difficulty to define and parameterize its components: selecting which combination of base classifiers to use, and which classifiers to use as the meta-classifier. The approach we present in this chapter poses this problem as an optimization task, and then uses optimization techniques based on heuristic search to solve it. In particular, we apply genetic algorithms to automatically obtain the ideal combination of learning methods for the stacking system.

INTRODUCTION

One of the most active and promising fields in inductive Machine Learning is the ensemble of classifiers approach. An ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way to classify new examples (Dietterich, 1997). The purpose of combining classifiers consists of improving the accuracy of a single classifier. Experimental results show that this is usually achieved.

There are several ways to construct such ensembles, but currently the most frequently used ones are bagging (Breiman, 1996), boosting (Freund & Schapire, 1995) and less widely used, stacking (Wolpert, 1992). Bagging constructs a set of classifiers by subsampling the training examples to generate different hypotheses.

After the different hypotheses are generated, they are combined by a voting mechanism. Boosting also uses the voting system to combine the classifiers. But, instead of subsampling the training examples, it generates the hypotheses sequentially. In each repetition, a new classifier is generated whose focus are those instances that were handled incorrectly by the previous classifier. This is achieved by giving a weight to each instance in the training examples and adjusting these weights according to its importance after every iteration. Both, bagging and boosting use classifiers generated by the same base-learning algorithm and obtained from the same data. Finally, stacking can combine classifiers obtained from different learning algorithms using a high level classifier –the metaclassifier- to combine the lower level models. This is based on the fact that different classifiers are obtained from the same data and different learning algorithms use different biases to search the hypothesis space. This approach expects that the metaclassifier will be able to learn how to decide between the predictions provided by the base classifiers, to improve their accuracy, much in the same way as a committee of experts.

One problem associated with stacked generalization is identifying which learning algorithm should be used to obtain the metaclassifier, and which ones should be the base classifiers. The approach we present in this chapter poses this problem as an optimization task, and then uses optimization techniques based on heuristic search to solve it. In particular, we apply genetic algorithms (Holland, 1975) to automatically obtain the ideal combination of learning methods for the stacking system.

BACKGROUND

The purpose of this section is to give enough background to understand the rest of the paper. Here, we will explain concepts related to ensembles of classifiers, bagging, boosting, stacking, and genetic algorithms.

Ensemble of Classifiers

The combination of multiple classifiers to improve the accuracy of a single classifier has had good results over several datasets that appear in recent papers about ensembles of classifiers (Bauer & Kohavi, 1999; Breiman, 1996; Freund & Schapire, 1996; Quinlan, 1996). According to Dietterich (1997), an ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way to classify new examples. There are many ways to construct an ensemble of classifiers. Bauer and Kohavi (1999) have made a comparison of algorithms based on voting systems. Dietterich (2000) carried out a survey of the main methods to construct an ensemble of classifiers. One way to construct an ensemble of classifiers is based on subsampling the training set to generate a different set of hypotheses and then combine them. This is called bagging (Breiman, 1996). The second way is to create classifiers sequentially, giving more importance to examples that were

misclassified in the previous classifier. The latter is called boosting (Schapire, 1990). Both bagging and boosting use a single learning algorithm to generate all the classifiers, whereas stacking combines classifiers from different learning algorithms. There are other methods to combine a set of classifiers (Dietterich, 2000), but for experimental purposes in this study we consider here the most well known methods, bagging, boosting and stacking.

Bagging

Breiman (1996) introduced the Bootstrap *aggregating* Algorithm (Bagging) that generates different classifiers from different bootstrap samples and combines decisions from the different classifiers into a single prediction by voting (the class that gets more votes from the classifiers wins). Figure 1 shows the bagging algorithm.

Each bootstrap sample is generated by sampling uniformly (with replacement) the m-sized training set, until a new set with m instances is obtained. Obviously, some of the instances of the training set will be cloned in the bootstrap sample, and some will be missing. Then, a classifier C_i is built from each of the bootstrap samples B_1, B_2, … B_T. Each classifier corresponding to each bootstrap sample will focus on some instances of the training set, and therefore the resulting classifiers will usually differ from each other. The final classifier C* is the ensemble of the C_i classifiers combined by means of the uniform voting system (all classifiers have equal weight).

Since an instance has probability $1-(1-1/m)^m$ of being in the bootstrap sample, each bootstrap replicate has, on average, 63.2% of unique training instances ($1 - 1/e = 63.2\%$). For this reason the generated classifiers will be different if the base learning algorithm is unstable (e.g. decision tree or neural networks). If the generated classifiers are good and do not correlate, the performance will be improved. This will occur if they make few mistakes and different classifiers do not misclassify the same instances frequently.

Figure 1. The Bagging Algorithm

The Bagging Algorithm
Input:
Training set S
Base Learning Algorithm B
Number of bootstrap samples T

Procedure:
For i = 1 to T {
S' = bootstrap sample from S (S' is a sample with replacement from S)
C = B (S') (create a new classifier from S')
}
$C^*(x) = \arg\max_{y \in Y} \sum_{i:C_i(x)=y} 1$ (the most often predicted label y)

Output
Classifier C*

Boosting

Another method to construct an ensemble of classifiers is known as boosting (Schapire, 1990), which is used to boost the performance of a weak learner. A weak learner is a simple classifier whose error is less than 50% on training instances. There are many variants of the boosting idea, but in this study we will describe the widely used AdaBoost Algorithm (Adaptive *Boost*ing) that was introduced by Freund & Schapire (1995). Sometimes the AdaBoost algorithm is also known as AdaBoostM1.

Figure 2 AdaBoostM1 Algorithm.

The AdaBoostM1 Algorithm

Input:
Training set S, of labelled instances: $S = \{(x_i, y_i), i = 1,2, \ldots, m\}$,
classes $y_i \in Y = \{1, \ldots, K\}$
Base Learning Algorithm (weak learner) B
Number of iteration T

Procedure:
Initialise for all i: $w_1(i) = 1/m$, $t = 1$ — Assign equal weights to all training instances

For $t = 1$ to T {

for i: $p_t(i) = w_t(i)/(\Sigma_i\ w_t(i))$ — Normalise the weight of the instances

$C_t = B(p_t)$ — Apply the base learning algorithm with the normalised weights

weighted error on training set

If $\varepsilon_i > 1/2$ then
T= t -1
exit
$\beta_t = \varepsilon_i/(1- \varepsilon_i)$

for all i: $w_{t+1}(i) = w_t(i)$ — Compute the new weights

}

Classifier

Output
Classifier C^*

Similarly to bagging, boosting manipulates the training set to generate a set of classifiers of the same type (e.g. decision trees). But, while in bagging classifiers are generated independently from each other, in boosting they are generated sequentially, in such a way that each one complements the previous one. First, a classifier is generated using the original training set. Then, those instances misclassified by this classifier are given a larger weight (therefore, misclassifying this instance makes the error larger). Next, a new classifier is obtained by using the previously weighted training set. The training error is calculated and a weight is assigned to the classifier in accordance with its performance on the training set. Therefore, the new classifier should do better on the misclassified instances than the previous one, hence complementing it. This process is repeated several times. Finally, to combine the set of classifiers in the final classifier, AdaBoost (like bagging) uses the voting method. But unlike bagging that use equal weights for all classifiers, in boosting the classifiers with lower training error have a higher weight in the final decision.

More formally, let S be the training set and T the number of trials, the classifiers C_1, C_2,…. C_T are built in sequence from the weighted training samples (S_1, S_2,…,S_T) The final classifier is the combination of the set of classifiers in which the weight of each classifier is calculated by its accuracy on its training sample. The AdaBoost algorithm is displayed in Figure 2.

An important characteristic of the AdaBoost algorithm is that it can combine some weak learners to produce a strong learner, so long as the error on the weighted training set is smaller than 0.5. If the error is greater than 0.5, the algorithm finishes the execution. Were it not so, the final prediction of the ensemble would be random.

Stacking

Stacking is the abbreviation used to refer to Stacked Generalization (Wolpert, 1992). Unlike bagging and boosting, it uses different learning algorithms to generate the ensemble of classifiers. The main idea of stacking is to combine classifiers from different learners such as decision trees, instance-based learners, etc. Since each one uses different knowledge representation and different learning biases, the hypothesis space will be explored differently, and different classifiers will be obtained. Thus, it is expected that they will not be correlated.

Once the classifiers have been generated, they must be combined. Unlike bagging and boosting, stacking does not use a voting system because, for example, if the majority of the classifiers make bad predictions, this will lead to a final bad classification. To solve this problem, stacking uses the concept of a *meta learner*. The meta learner (or level-1 model), tries to learn, using a learning algorithm, how the decisions of the base classifiers (or level-0 models) should be combined.

Given a data set S, stacking first generates a subset of training sets S_1, …, S_T and then follows something similar to a cross-validation process: it leaves one of the subsets out (e.g. S_j) to use later The remaining instances $S^{(-j)} = S - S_j$ are used to generate the level-0 classifiers by applying *K* different learning algorithms, $k = 1$, …, *K*, to obtain *K* classifiers. After the level-0 models have been generated, the S_j

set is used to train the meta learner (level-1 classifier). Level-1 training data is built from the predictions of the level-0 models over the instances in S_j that were left out for this purpose. Level-1 data has K attributes, whose values are the predictions of each one of the K level-0 classifiers for every instance in S_j. Therefore, a level-1 training example is made of K attributes (the K predictions) and the target class, which is the right class for every particular instance in S_j. Once the level-1 data has been built from all instances in S_j, any learning algorithm can be used to generate the level-1 model. To complete the process, the level-0 models are re-generated from the whole data set S (this way, it is expected that classifiers will be slightly more accurate). To classify a new instance, the level-0 models produce a vector of predictions that is the input to the level-1 model, which in turn predicts the class.

There are many ways to apply the general idea of stacked generalization. Ting and Witten (1997) use probability outputs from level-0 models instead of a simple class prediction as inputs to the level-1 model. LeBlanc and Tibshirani (1993) analyze the stacked generalization with some regularization (non-negative constraint) to improve the prediction performance on one artificial dataset. Other works on stacked generalization have developed a different focus (Chan & Stolfo, 1995; Fan, Chan & Stolfo, 1996).

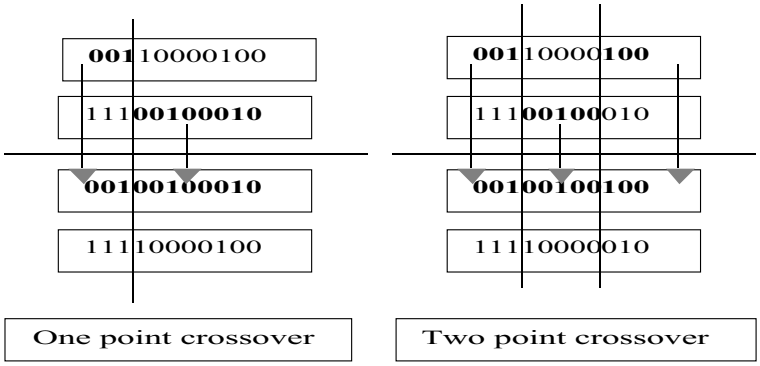
Genetic Algorithms for Optimization Problems

Genetic algorithms (GA's) are search procedures loosely connected to the theory of evolution by means of artificial selection (Holland, 1975). In classical search terms, GA's can be viewed as a kind of beam search procedure. Its main three components are:

- The beam (or population). It contains the set of points (candidate solutions or individuals) in the search space that the algorithm is currently exploring. All points are usually represented by means of bit strings. This domain independent representation of candidate solutions makes GA's very flexible.
- The search operators. They transform current candidate solutions into new candidate solutions. Their main characteristic is that, as they operate on bit strings, they are also domain independent. That is, they can be used to search in any domain. GA's operators are also based in biological analogies. The three most frequently used operators are:
  - Reproduction: it just copies the candidate solution without modification.
  - Crossover: it takes two candidate solutions, mixes them and generates two new candidate solutions. There are many variations of this operator (mainly, single point, two point and uniform). See Figure 3.
  - Mutation: it flips a single bit of a candidate solution (it mutates from 0 to 1, or from 1 to 0). The bit is selected randomly from the bits in the individual.
- The heuristic function (or fitness function). This function measures the worth of a candidate solution. The goal of a GA is to find candidate solutions that maximize this function.

GA's start from a population made up of randomly generated bit strings. Then,

Figure 3. One and Two Point Crossover Operations



genetic operators are applied to worthy candidate solutions (according to the heuristic function) until a new population is built (the new generation). A GA continues producing new generations until a candidate solution is found that is considered to be good enough, or when the algorithm seems to be unable to improve candidate solutions (or until the number of generations reaches a predefined limit). More exactly, a GA follows the algorithm below:

1. Randomly generate initial population G(0)
2. Evaluate candidate solutions in G(0) with the heuristic function
3. Repeat until a solution is found or population converges
   3.1. Apply selection-reproduction: G(i) -> $G_a(0)$
   3.2. Apply crossover: $G_a(i)$ -> $G_b(i)$
   3.3. Apply mutation: $G_b(i)$ -> $G_c(i)$
   3.4. Obtain a new generation G(i+1) = $G_c(i)$
   3.5. Evaluate the new generation G(i+1)
   3.6. Let i=i+1

The production of a new generation G(i+1) from G(i) (steps 3.1, 3.2, and 3.3) is as follows. First, a new population $G_a(i)$ is generated by means of selection. In order to fill up a population of n individuals for $G_a(0)$, candidate solutions are stochastically selected with replacement from G(i) n times. The probability for selecting a candidate solution is the ratio between its fitness and the total fitness of the population. This means that there will be several copies of very good candidate solutions in $G_a(0)$, whereas there might be none of those whose heuristic evaluation is poor. However, due to stochasticity, even bad candidate solutions have a chance of being present in $G_a(0)$. This method is called 'selection proportional to fitness', but there are several more, like tournament and ranking (Goldberg,1989).

$G_b(i)$ is produced by applying crossover to a fixed percentage p_c of randomly selected candidate solutions in $G_a(i)$. As each crossover event takes two parents and produces two offspring, crossover happens $p_c/2$ times. In a similar manner, $G_c(i)$ is

obtained from $G_b(0)$ by applying mutation to a percentage p_m of candidate solutions.

GA-STACKING APPROACH

In this section, the approach we have taken to build stacking systems will be explained. Given that stacked generalization is composed by a set of classifiers from a different set of learning algorithms, the question is: what algorithm should be used to generate the level-0 and the level-1 models? In principle, any algorithm can be used to generate them. For instance, Ting and Witten (1997) showed that a linear model is useful to generate the level-1 model using probabilistic outputs from level-0 models. Also, our classifiers at level-0 are heterogeneous, and any algorithm can be used to build the metaclassifier. In the present study, to determine the optimum distribution of learning algorithms for the level-0 and level-1 models, we have used a GA.

GA-Stacking

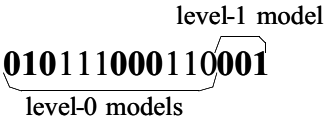
Two sets of experiments were carried out to determine whether stacked generalization combinations of classifiers can be successfully found by genetic algorithms. We also wanted to know whether GA stacking can obtain improvements over the most popular ensemble techniques (bagging and boosting) as well as any of the standalone learning algorithms.

As indicated previously, the application of genetic algorithms to optimization problems requires to define:

- the representation of the individuals (candidate solutions)
- the fitness function that is used to evaluate the individuals
- the selection-scheme (e.g., selection proportional to fitness)
- the genetic operators that will be used to evolve the individuals
- the parameters (e.g., size of population, generations, probability of crossover, etc.)

The representations of the possible solutions (individuals) used in the two sets of performed experiments, are chromosomes with five genes (see Figure 4). Each gene represents the presence of a learning algorithm. Since we want to use seven possible learning algorithms, each gene has three binary digits to represent them. The firsts four genes of the chromosome represent the four learning algorithms to build the level-0 classifiers and the last gene represents the algorithm to build the metaclassifier.

Figure 4. Individual description.



To evaluate every individual, we used the classification accuracy of the stacking system as the fitness function.

EXPERIMENTAL RESULTS

In this section, we present the empirical results for our GA-Stacking system. GA-Stacking has been evaluated in several domains of the UCI database (Blake & Merz, 1998). We have performed two sets of experiments. In the first one, we show some preliminary results, which are quite good but suffer some overfitting. In the second set of experiments, steps were taken to avoid the said overfitting. Both sets of experiments differ only in the way GA individuals are tested (i.e. the fitness function). In this work, we have used the algorithms implemented in Weka (Witten & Frank, 2000). This includes all the learning algorithms used and a implementation of bagging, boosting and stacking.

Basically, the implementation of GA-Stacking combines two parts: a part coming from Weka that includes all the base learning algorithms and another part, which was integrated into Weka, that implements a GA. The parameters used for the GA in the experiments are shown in Table 1. The elite rate is the proportion of the population carried forward unchanged from each generation. Cull rate refers to the proportion of the population that is deemed unfit for reproduction. Finally, the mutation rate is the proportion of the population that can suffer change in the configuration at random.

For the experimental test of our approach we have used seven data sets from the repository of machine learning databases at UCI. Table 2 shows the data sets characteristics.

The candidate learning algorithms for GA stacking we used were:

- C4.5 (Quinlan, 1993). It generates decision trees - (C4.5)
- PART (Frank & Witten, 1998).

Table 1. Genetic algorithms parameters.

Parameters	Values
Population size	10
Generations	10
Elite rate	0.1
Cull rate	0.4
Mutation rate	0.067

Table 2. Datasets description.

Dataset	Attributes	Attributes Type	Instances	Classes
Heart disease	13	Numeric-nominal	303	2
Sonar classification	60	Numeric	208	2
Musk	166	Numeric	476	2
Ionosphere	34	Numeric	351	2
Dermatology	34	Numeric-nominal	366	6
DNA splice	60	Nominal	3190	3
Satellite images	36	Numeric	6435	6

It forms a decision list from pruned partial decision trees generated using the C4.5 heuristic - (PART)
- A probabilistic Naive Bayesian classifier (John & Langley, 1995) - (NB)
- IB1. This is Aha's instance based learning algorithm (Aha & Kibler, 1991) - (IB1)
- Decision Table. It is a simple decision table majority classifier (Kohavi, 1995) - (DT)
- Decision Stump. It generates single-level decision trees - (DS)

Preliminary Experiments

Experiments in this subsection are a first evaluation of GA-Stacking. They use the GA-Stacking scheme described in previous sections. Here, we will describe in more detail how individuals are tested (the fitness function), and how the whole system is tested. Each data set was split randomly in two sets. One of them was used as the training set for the GA fitness function. It contained about 85% of all the instances. Individual fitness was measured as the accuracy of the stacking system codified in it. The rest was used as a testing set to validate the stacked hypothesis obtained. In these experiments we used only two of the data sets shown in Table 2 (dermatology and ionosphere).

Table 3 shows the results for this preliminary approach. Training and test accuracy columns display how many instances in the training and testing sets were correctly predicted for each of the systems. The first part of the table contains the standalone algorithms. The second part shows results corresponding to traditional bagging and boosting approaches (with C4.5 as the base classifier). Finally, results for GA-Stacking are shown. The best GA-Stacking individual in the last generation

Table 3. Preliminary Results.

Algorithm	Ionosphere		Dermatology	
	Training Accuracy	Test Accuracy	Training Accuracy	Test Accuracy
C4.5	98.42	82.86	96.67	92.42
Naive Bayes	85.13	82.86	98.67	96.97
PART	98.73	88.57	96.67	93.94
IBk (one neighbor)	100.00	82.86	100.00	92.42
IB1	100.00	82.86	100.00	92.42
Decision Stump	84.18	80.00	51.33	45.45
Decision Table	94.94	88.57	96.67	87.88
Ensembles				
Bagging with C4.5	97.78	85.71	97.00	93.94
Boosting with C4.5	100.00	91.43	100.00	96.97
GA-Stacking(last generation solutions)	100.00	85.71	100.00	95.45
GA-Stacking (solutions of previous generations)	97.78	94.29	98.67	98.48

has a 100% in the training set in both domains, but drops to 85.71% and 95.45% in testing. On the other hand, some individuals from previous generations manage to get a 94.29% and 98.48% testing accuracies. This means that GA-Stacking overfits the data as generations pass on. Despite this overfitting, GA-Stacking found individuals which are better than most of the base classifiers. Only in the Ionosphere domain, the Decision Table is better in testing than the overfitted individual (88.57 vs. 85.71). Also, performance of GA-Stacking is very close to bagging and boosting (both using C4.5 to generated base classifiers). Non-overfitted individuals (those obtained in previous generations) show an accuracy better than any of the other ensemble systems. In addition, stacking systems only use four level-0 classifiers whereas bagging and boosting use at least 10 base classifiers. This shows that the approach is solid. The following section shows how we overcame the problem of overfitting.

Main Experiments (Overfitting avoidance)

In the preliminary experiments, individuals overfit because they were evaluated with the same training instances that were used to build the stacking associated with the individual. Obviously, the individual will do very well with the instances it has been trained with. Therefore, in order to avoid the overfitting, the fitness value was calculated with a separate validation set (different from the testing set used to evaluate GA-Stacking itself). To do so, the set of training instances was partitioned randomly into two parts. 80% of training instances were used to build the stacking system associated with each individual, and the remaining 20% –the validation set- was used to give a non-biased estimation of the individual accuracy. The latter was used as the fitness of the individual. It is important to highlight that now fewer instances are used to build the stacking systems, as some of them are used to prevent overfitting.

For testing all the systems we used the testing data set, just as we did in the preliminary experiments. However, now a five-fold cross-validation was used. Therefore, results shown are the average of the five cross-validation iterations. In the satellite images domain no cross-validation was carried out because the sets for training and testing are available separately and the data set donor indicated that it is better not to use cross-validation.

Results are divided in two tables. Table 4 shows the testing accuracy of all the base algorithms over the data sets.

Table **5** displays testing results for the three ensemble methods, including GA-Stacking. The best results for each domain are highlighted. Only in the heart domain, one of the base classifiers obtained greater accuracy than any ensemble method. The configuration of the stacked generalization found by the genetic algorithms obtained a greater accuracy in four of the seven domains used in the experiment in contrast to the other ensemble methods. In the domains where stacking has lesser accuracy than bagging or boosting the difference is very small excepting in the musk domain where the difference is +5% in favor of boosting.

*Table 4. Average accuracy rate of independent classifiers in the second experiment. *without cross-validation.*

Dataset	C4.5	PART	NB	IB1	DT	DS
Heart	74.00	82.00	83.00	78.00	76.33	70.67
Sonar	72.38	73.81	67.62	79.52	71.90	73.93
Musk	83.33	85.21	74.38	86.46	82.08	71.46
Ionosphere	90.14	89.30	82.82	87.61	89.30	82.82
Dermatology	94.33	94.59	97.03	94.59	87.83	50.40
DNA splice	94.12	92.11	95.63	75.62	92.49	62.42
Satellite images*	85.20	84.10	79.60	89.00	80.70	41.75

Table 5. Average accuracy rate of ensemble methods in the second experiment.

Dataset	Bagging	Boosting	GA-Stacking
Heart	76.33	79.67	80.67
Sonar	80.00	79.05	80.47
Musk	87.29	88.96	83.96
Ionosphere	92.11	91.83	90.42
Dermatology	94.59	97.03	98.11
DNA splice	94.56	94.43	95.72
Satellite images	88.80	89.10	88.60

CONCLUSIONS

In this chapter, we have presented a new way of building hetereogeneous stacking systems. To do so, a search on the space of stacking systems is carried out by means of a genetic algorithm. Empirical results show that GA-Stacking is competitive with more sophisticated ensemble systems such as Bagging and Boosting. Also, our approach is able to obtain high accuracy classifiers which are very small (they contain only 5 classifiers) in comparison with other ensemble approaches.

Even though the GA-Stacking approach has already produced very good results, we believe that future research will show its worth even more clearly:

- Currently, GA-Stacking searches the space of stacked configurations. However, the behavior of some learning algorithms is controlled by parameters. For instance, the rules produced by C4.5 can be very different depending on the degree of pruning carried out. Our GA-Stacking approach would only have to include a binary coding of the desired parameters in the chromosome.
- Search depends on the representation of candidate hypothesis. Therefore, it would be interesting to use other representations for our chromosomes. For instance, the chromosome could be divided into two parts, one for the level 0 learners and another for the level 1 metaclassifier. A bit in the first part would

indicate whether a particular classifier is included in level 0 or not. The metaclassifier would be coded in the same way we have done in this paper (i.e. if there are 8 possible metaclassifiers, 3 bits would be used).

- Different classifiers could be better suited for certain regions of the instance space. Therefore, instead of having a metaclassifer that determines what output is appropriate according to the level 0 outputs (as currently), it would be interesting to have a metaclassifier that knows what level 0 classifier to trust according to some features of the instance to be classified. In that case, the inputs to the metaclassifier would be the attributes of the instance and the output the right level 0 classifier to use.

REFERENCES

Aha, D., & Kibler, D. (1991). Instance-based learning algorithms. *Machine Learning, 6,* 37-66.

Bauer, E., & Kohavi, R.(1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning, 36 (1/2)*, 105-139.

Blake, C.L., & Merz, C.J. (1998). *UCI Repository of machine learning databases*. Retrieved November 1, 2000, from http://www.ics.uci.edu/~mlearn/MLRepository.html

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

Chan, P., & Stolfo, S. (1995). A comparative evaluation of voting and meta-learning on partitioned data. In *Proceedings of Twelfth International Conference on Machine Learning* (90-98). Morgan Kaufmann.

Dietterich, T. (1997). Machine learning research: four current directions. *AI Magazine 18(4)*, 97-136.

Dietterich, T. (2000). Ensemble methods in machine learning. In Proceedings of *First International Workshop on Multiple Classifier Systems* (pp. 1-15). Springer-Verlag.

Fan, D., Chan, P., & Stolfo, S. (1996). A comparative evaluation of combiner and stacked generalization. In *Proceedings of AAAI-96 workshop on Integrating Multiple Learning Models* , 40-46.

Frank, E., & Witten, I. (1998). Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning* (144-151). Morgan Kaufmann.

Freund, Y., & Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, Springer-Verlag, 23-37.

Freund, Y., & Schapire, R. E. (1996). Experiment with a new boosting algorithm. In L. Saitta, de., *Proceedings Thirteenth International Conference on Machine Learning,* Morgan Kaufmann, 148-156.

Goldberg, D.E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley.

Holland, J.H. (1975). *Adaptation in natural and artificial systems*. The University of Michigan Press.

John, G.H., & Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, 338-345.

Kohavi R. (1995). The power of decision tables. In *Proceedings of the Eighth European*

Conference on Machine Learning.
LeBlanc, M., & Tibshirani, R. (1993). Combining estimates in regression and classification. *Technical Report 9318*. Department of Statistic, University of Toronto.
Quinlan, J. R. (1996). Bagging, boosting, and C4.5. In *Proceedings of Thirteenth National Conference on Artificial Intelligence*, AAAI Press and the MIT Press, 725-730.
Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning, 5(2)*, 197-227.
Ting, K., & Witten, I. (1997). Stacked generalization: when does it work. In *Proceedings International Joint Conference on Artificial Intelligence*.
Whitley, L.D.(1991). Fundamental principles of deception in genetic search. In Gregor y J.E (Ed.) *Foundations of genetic algorithms*, San Mateo, CA: Morgan Kaufmann, 221-241.
Witten, I., & Frank, E. (2000). *Data mining: practical machine learning tools and techniques with Java implementation*. San Francisco, CA: Morgan Kaufmann.
Wolpert, D. (1992). Stacked generalization. *Neural Networks 5*, 241-259.