Cost Complexity Pruning of Ensemble Classifiers *

Andreas L. Prodromidis†
Computer Science Department
Columbia University
New York, NY 10027
andreas@cs.columbia.edu

Salvatore J. Stolfo
Computer Science Department
Columbia University
New York, NY 10027
sal@cs.columbia.edu

ABSTRACT

In this paper we study methods that combine multiple classification models learned over separate data sets in a distributed database setting. Numerous studies posit that such approaches provide the means to efficiently scale learning to large datasets, while also boosting the accuracy of individual classifiers. These gains, however, come at the expense of an increased demand for run-time system resources. The final ensemble meta-classifier may consist of a large collection of base classifiers that require increased memory resources while also slowing down classification throughput. Here, we present a technique for measuring the tradeoff between predictive performance and available run time system resources and we describe an algorithm for pruning (i.e. discarding a subset of the available base classifiers) the ensemble meta-classifier as a means to reduce its size while preserving its accuracy. The algorithm is independent of the method used initially when computing the meta-classifier. It is based on decision tree pruning methods and relies on the mapping of an arbitrary ensemble meta-classifier to a decision tree model. Through an extensive empirical study on meta-classifiers computed over two real data sets, we illustrate our pruning algorithm to be a robust approach to discarding classification models without degrading the overall predictive performance of an ensemble computed over those that remain after pruning.

1. INTRODUCTION

Recently, there has been considerable interest in *meta-learning* techniques that combine or integrate an ensemble of models computed by the same or different learning algorithms over multiple data subsets [7, 10]. An advantage of such an approach is that it can improve both *efficiency* and *scalability* by executing the machine learning processes *in parallel* and on (possible disjoint) subsets of the data (*a data reduction technique*). Moreover, it can produce a "higher quality" final classification model, also called *meta-classifier*, by combining classifiers with different inductive bias [26].

The JAM system (Java Agents for Meta-learning) [38], for example, is a distributed agent-based data mining system that integrates multiple models. JAM takes full advantage of the inherent parallelism and distributed nature of meta-learning by providing a set of learning agents that compute models (classifier agents) over data stored locally at a site. JAM supports the remote dispatch and exchange of learning and classifier agents among the participating data base sites through a special distribution mechanism. It also provides a set of meta-learning agents that combine the computed models that were learned (perhaps) at different sites.

Several methods for integrating ensembles of models have been studied, including techniques that combine the set of models in some linear fashion [1, 2, 3, 12, 20, 27, 29, 37, 39, 21], techniques that employ referee functions to arbitrate among the predictions generated by the classifiers, [16, 17, 18, 19, 28, 36], methods that rely on principal components analysis [23, 24] or methods that apply inductive learning techniques to learn the behavior and properties of the candidate classifiers [6, 40].

Constructing ensembles of classifiers is not cheap and produces a final outcome that is expensive due to the increased complexity of the final meta-classifier. In general, meta-classifiers combine all their constituent base-classifiers. Hence, to classify unlabeled instances, predictions need to be generated from all base-classifiers before the meta-classifier can produce its final classification. This results in significant decrease in classification throughput (the speed with which an unknown datum can be classified) and increased demand for system resources (including memory to store base classifiers).

From experiments conducted on a Personal Computer with a 200MHz Pentium processor running Solaris 2.5.1 where base- and meta-classifiers were trained to detect credit card fraud, we measured a decrease of 50%, 70% and 80% credit card transaction processing throughput for meta-classifiers composed of 13, 20 and 30 base-classifiers, respectively. Meta-classifier throughput is crucial in real-time systems, such as e-commerce or intrusion detection systems. Memory constraints are equally important. For the same problem, a single ID3 decision tree ([33]) may require more than 850KBytes

*This research is supported by the Intrusion Detection Program (BAA9603) from DARPA (F30602-96-1-0311), NSF (IRI-96-32225 and CDA-96-25374) and NYSSTF (423115-445).

†Supported in part by an IBM fellowship.

SIGKDD 2000 Workshop on Distributed and Parallel Knowledge Discovery
2000 Boston, MA USA

of main memory, while a C4.5 decision tree ([34]) may need 100KBytes. Given the phenomenal growth in the number and size of the databases and data warehouses, retaining and deploying an increasing number of base classifiers and meta-classifiers may not be practical nor feasible.

In this paper we describe a technique for studying the tradeoff between predictive performance and available resources and we consider pruning techniques that aim to reduce the complexity and cost of ensemble meta-classifiers. We introduce a *post-training pruning* algorithm for discarding the base classifiers that are redundant or less "relevant" to the meta-classifier. *Post-training pruning* is applied to a pre-existing meta-classifier.[1] The objective of pruning, here, is to build a smaller and faster meta-classifier that achieves comparable performance (accuracy) to the unpruned meta-classifier.

Our post-training pruning algorithm is based on the pruning methods employed by decision-tree learning algorithms. In principle, it can be applied to any meta-classifier, regardless of the method used to integrate the base classifiers. In this study, we test it on meta-classifiers computed by the Naive Bayesian[25] and the Ripper[8] learning algorithms (stacking), on meta-classifers combined by the majority voting and weighted voting schemes and on meta-classifiers integrated by the SCANN [23] algorithm. The remainder of this paper is organized as follows. In Section 2 we provide a brief overview of these classifier-combining techniques and in Section 3 we introduce the post-training pruning algorithm. The effectiveness of this algorithm is empirically evaluated on classifiers and meta-classifiers computed over two credit card transaction data data sets. Section 4 details these experiments and illustrates the decision-tree-based post-training pruning method to be highly robust in reducing the size of the meta-classifiers while maintaining predictive performance. Finally, in Section 5 we conclude the paper.

2. BACKGROUND

We consider three methods for combining the predictions of multiple base classifiers, voting, stacking and SCANN (based on Correspondence Analysis). Before we overview the three combining strategies, we introduce the following notation. Let C_i, $i = 1, 2, ..., K$, denote a base classifier computed over data subsets D_j, $j = 1, 2, ..., L$, of training data D, and M the number of possible classes. Let $\mathbf{x}$ be an instance whose classification we seek, and $C_1(\mathbf{x})$, $C_2(\mathbf{x})$,..., $C_K(\mathbf{x})$, $C_i(\mathbf{x}) \in \{y_1, y_2, ..., y_M\}$, be the predicted classifications of $\mathbf{x}$ from the K base classifiers. Finally, let V be a separate validation set of size n that is used to generate *meta-level* predictions of the K base classifiers.

2.1 Voting

Voting denotes the simplest method of combining predictions from multiple classifiers. In its simplest form, called *plurality* or *majority* voting, each classification model C_i, $i = 1, 2, ..., K$ contributes a single vote (its own classification). The collective prediction is decided by the majority of the votes, i.e. the class $y_j, j \in \{y_1, y_2, ..., y_M\}$ with the most votes is the final prediction. In *weighted* voting, on the other hand, the classifiers have varying degrees of influence on the collective predictions that is relative to their predictive accuracy. Each model is associated with a specific weight determined by its performance (e.g accuracy, cost model) on a validation set. The final prediction is decided by summing over all weighted votes and by choosing the class with the highest aggregate. For a binary classification problem, for example, where each classifier C_i with weight w_i casts a 0 vote for class y_1 and a 1 vote for class y_2, the aggregate is given by:

$$S(\mathbf{x}) = \frac{\sum_{i=1}^{K} w_i C_i(\mathbf{x})}{\sum_{i=1}^{K} w_i} \qquad (1)$$

If we choose 0.5 to be the threshold distinguishing classes y_1 and y_2, the weighted voting method classifies unlabeled instances $\mathbf{x}$ as y_1 if $S(\mathbf{x}) < 0.5$, as y_2 if $S(\mathbf{x}) > 0.5$ and randomly if $S(\mathbf{x}) = 0.5$.

This approach can be extended to non-binary classification problems by mapping the m-class problem into m binary classification problems and by associating each class j with a separate $S_j(\mathbf{x})$, $j \in 1, 2, ...m$. To classify an instance $\mathbf{x}$, each $S_j(\mathbf{x})$ generates a confidence value indicating the prospect of $\mathbf{x}$ being classified as j versus being classified as non-j. The final class selected corresponds to the $S_j(\mathbf{x})$, $j \in 1, 2, ...m$ with highest confidence value.

In this study, the weights w_i's are set according to the performance (with respect to a selected evaluation metric, e.g accuracy) of each classifier C_i on a separate validation set.

2.2 Stacking

The main difference between voting and *Stacking* [40] (or *Class-Combiner Meta-learning* [5]) is that the latter combines base classifiers in a non-linear fashion. The combining task, called a *meta learner*, integrates the independently computed base classifiers into a higher level meta-classifier, by learning over a *meta-level training set*. This meta-level training set is composed by using the base classifiers' predictions $C_1(\mathbf{x})$, $C_2(\mathbf{x})$,..., $C_K(\mathbf{x})$, $C_i(\mathbf{x})$ on the validation set V as attribute values, and the true class as the target. From these predictions, the meta-learner learns the characteristics and performance of the base classifiers C_i, $i = 1, 2, ..., K$ and computes a meta-classifier which is a model of the "global" data set. To classify an unlabeled instance $\mathbf{x}$, the base classifiers present their own predictions to the meta-classifier which then makes the final classification.

2.3 SCANN

The third approach we consider is Merz's SCANN [23] (Stacking, Correspondence Analysis and Nearest Neighbor) algorithm for combining multiple models as a means to improve classification performance. As with stacking, the combining task integrates the independently computed base classifiers in a non-linear fashion. The meta-level training set is composed by using the base classifiers' predictions on the validation set as attribute values, and the true class as the target. In this case, however, the predictions and the correct class are de-multiplexed and represented in a 0/1 form.[2] SCANN employs correspondence analysis [14]

[1] Conversely, pre-training pruning [30] refers to the filtering of base classifiers before they are included in an ensemble meta-classifier.

[2] The meta-level training set is a matrix of n rows (records) and $[(K+1) \cdot M]$ columns (0/1 attributes), i.e. there are M columns assigned to each of the K classifiers and M columns

(similar to Principal Component Analysis) to geometrically explore the relationship between the validation examples, the models' predictions and the true class. In doing so, it maps the true class labels and the predictions of the classifiers onto a new scaled space that clusters similar prediction behaviors. Then, the nearest neighbor learning algorithm is applied over this new space to meta-learn the transformed predictions of the individual classifiers. To classify unlabeled instances, SCANN maps them onto the new space and assigns them the class label corresponding to the closest class point.

SCANN is a sophisticated combining method that seeks to geometrically uncover the position of the classifiers relative to the true class labels. On the other hand, it relies on singular value decomposition techniques to compute the new scaled space and capture these relationships, which can be expensive, both in space and time, as the number of examples and hence as the number of the base classifiers increases (the overall time complexity of SCANN is $(O((M \cdot K)^3))$.

3. POST-TRAINING PRUNING

Post-training pruning refers to the pruning of a meta-classifier after it is constructed. In contrast with pre-training pruning [30] which uses greedy forward-search methods to choose classifiers, post-training pruning is considered a backwards selection method. It starts with all available K classifiers (or with the classifiers pre-training pruning selected) and iteratively removes one at a time. The objective is to perform a search on the (possibly reduced) space of meta-classifiers and prune the meta-classifier without degrading predictive performance.

The deployment and effectiveness of post-training pruning methods depends highly upon the availability of unseen labeled (validation) data. Post-training pruning can be facilitated if there is an abundance of data, since a separate labeled subset can be used to estimate the effects of discarding specific base classifiers and thus guide the backwards elimination process. A hill climbing pruning algorithm, for example, would employ the separate validation data to evaluate and select the best (out of K possible) meta-classifier with $K-1$ base classifiers, then evaluate and select the best meta-classifier with $K-2$ (out of $K-1$ possible) and so on. In the event that additional data is not available, standard cross validation techniques can be used, instead, to estimate the performance of the pruned meta-classifier, at the expense of increased complexity.

A disadvantage of such a method, besides the need for a separate data set, and its vulnerability to the horizon effect, is the overhead of constructing and evaluating all the intermediate meta-classifiers ($O(K^2)$ in the example), which, depending on the combining methods (e.g the learning algorithm in stacking) and the data set size, can be prohibitive.

To our knowledge, Margineantu and Dietterich's "Pruning Adaptive Boosting" [22], constitutes the only other (besides our prior work on pre-training pruning methods) related research on this subject. In that paper, the authors study the problem of pruning the ensemble of classifiers obtained by the boosting algorithm ADABOOST [13]. Their research however, was restricted to computing all classifiers by applying the *same* learning algorithm on many different subsets of the *same* training set. Furthermore, their most effective pruning algorithm, "Reduce-Error Pruning with Backfitting" suffers from the same disadvantage of computing intermediate meta-classifiers as our hill climbing method.

Next, we describe an efficient post-training pruning algorithm that does not require intermediate meta-classifiers or separate validation data. Instead it extracts information from the ensemble meta-classifier and employs the meta-classifier training set seeking to minimize (meta-level) training error. Furthermore, we consider the more general setting where the final classifications may be obtained be any meta-learning technique, not just voting (ADABOOST), and ensembles of classifiers may be generated by applying, possibly, different learning algorithms over (possibly) distinct databases,

3.1 Cost complexity pruning

The algorithm is based on the *minimal cost complexity pruning* method of the CART decision tree learning algorithm [4]. CART computes a large decision tree T_0 to fit the training data, by allowing the splitting process to continue until all terminal nodes are either small, or pure (i.e. all instances belong to the same class) or contain only instances with identical attribute-value vectors. Next, it applies the cost complexity pruning method to compute a set of consecutive nested subtrees $T_i, i \in \{1, 2, ..., R\}$ of decreasing size from the original large tree T_0 by progressively pruning upwards to its root node (T_R corresponds to the subtree that consists of the root node only).

To compute the set of these nested subtrees $T_i, i \in \{1, 2, ..., R\}$, the cost complexity pruning method associates a complexity measure $C(T)$ with the number of terminal nodes of decision tree T. The method prunes decision trees by seeking to minimize a cost complexity metric $R_\alpha(T)$ that combines two factors, the size (complexity) and the accuracy (or equivalently the error rate) of the tree. Specifically, $R_\alpha(T)$ is defined as

$$R_\alpha(T) = R(T) + \alpha \cdot C(T) \qquad (2)$$

where $R(T)$ denotes the misclassification cost (error rate[3]) of the decision tree T and α represents a complexity parameter ($\alpha \geq 0$).

The degree of pruning of the initial tree T_0 can be controlled by adjusting the complexity parameter α, which, according to Equation 2 corresponds to the weight of the complexity factor $C(T)$. If α is small, the penalty for having a large number of terminal nodes is small; as the penalty α per terminal node increases, the pruning algorithm removes an increasing number of terminal nodes in an attempt to compensate and thus generates the nested subtrees $T_i, i \in \{1, 2, ..., R\}$. The *minimal cost complexity pruning* method guarantees to find the "best" (according to the misclassification cost) pruned decision tree T_r, $r \in \{1, 2, ..., R\}$, of the original tree T_0 of a specific size (as dictated by the complexity parameter). An alternative pruning method using Rissanen's minimum description length is described in [35].

for the correct class. If classifier $C_i(\mathbf{x}) = j$, then the j^{th} column of C_i will be assigned a 1 and the rest $(M-1)$ columns a 0.

[3]Estimated over a separate pruning subset of the training set or using cross validation methods.

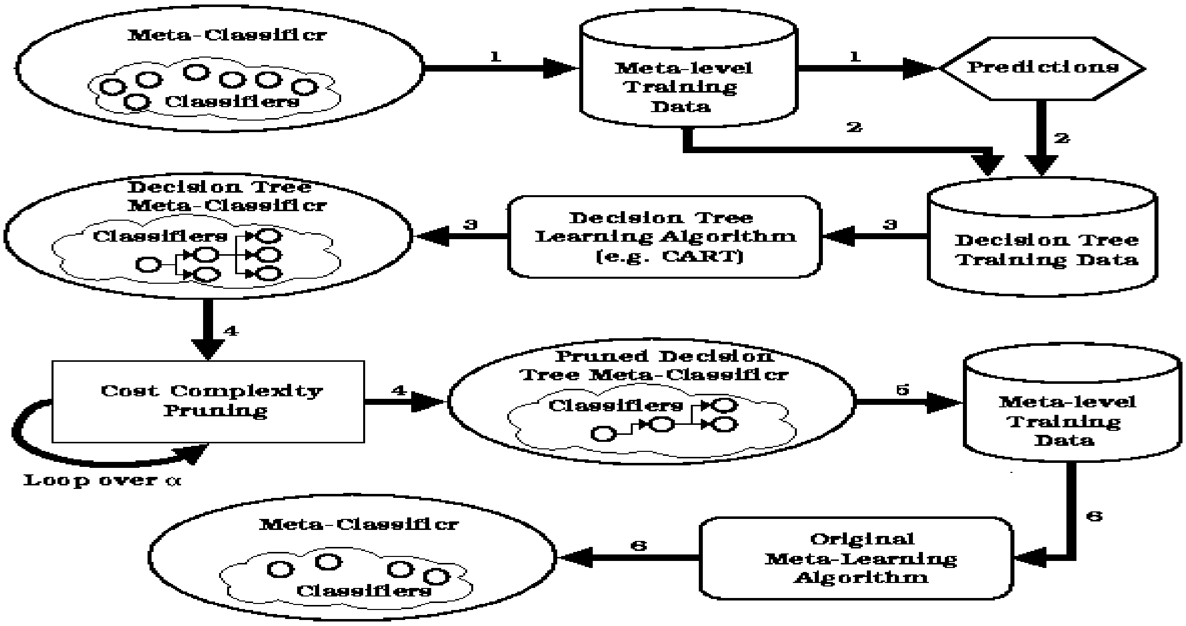

Figure 1: The six steps of the Post-Training Pruning Algorithm

3.2 Pruning meta-classifiers

The post-training pruning algorithm employs the minimal cost complexity method as a means to reduce the size (number of base-classifiers) of the meta-classifiers. In cases where the meta-classifier is built via a decision tree learning algorithm, the use of the cost complexity pruning method is straight forward. The decision tree model constitutes the ensemble meta-classifier and each node corresponds to a single base-classifier. Thus, by determining which nodes to remove, the pruning algorithm discovers and discards the base classifiers that are least important.

To apply this method on meta-classifiers of arbitrary representation The post-training pruning algorithm employs the minimal cost complexity method as a means to reduce the size (number of base-classifiers) of the meta-classifiers. In cases where the meta-classifier is built via a decision tree learning algorithm, the use of the cost complexity pruning method is straight forward. The decision tree model constitutes the ensemble meta-classifier and each node corresponds to a single base-classifier. Thus, by determining which nodes to remove, the pruning algorithm discovers and discards the base classifiers that are least important.

(i.e. non decision-tree), the post-training pruning algorithm maps the meta-classifier to its "decision-tree equivalent". In general, the algorithm can be considered as a three-phase process. Phase one seeks to model the arbitrary meta-classifier as an equivalent decision tree meta-classifier that imitates its behavior (a similar approach appeared in [9] in the context of neural networks). Phase two removes as many base-classifiers as needed using the minimal cost complexity pruning method on the derived decision tree model, and phase three re-combines the remaining base classifiers using the original meta-learning algorithm. Hereinafter, the term "decision tree model" will refer to the decision tree trained to imitate the behavior of the initial arbitrary meta-classifier.

These three phases are graphically illustrated in six steps in Figure 1. For completeness, a detailed description of the post-training pruning algorithm is provided in Table 1.

1. The meta-classifier is applied to its own (meta-learning) training set. (Recall that the attributes of the meta-level training set correspond to the predictions of the base classifiers on the validation set and the true class labels correspond to the correct classes of the validation set.)

2. A new training set, called decision tree training set, is composed by using the meta-level training set (without the true class labels) as attributes, and the predictions of the meta-classifier on the meta-level training set as the true class target.

3. A decision-tree-based algorithm, such as CART, computes the "decision-tree equivalent" of the original meta-classifier by learning its input/output behavior recorded in the decision-tree training set. The resultant decision-tree meta-classifier is trained to imitate the behavior of the original meta-classifier and discover the manner in which it combines its constituent base classifiers. Furthermore, this decision tree reveals the base classifiers that do not participate in the splitting criteria and hence are irrelevant to the meta-classifier. Those base classifiers that are deemed irrelevant are pruned in order to meet the performance objective.

4. The next stage aims to further reduce the number of selected base classifiers, if necessary, according to the restrictions imposed by the available system resources and/or the runtime constraints. The post-training pruning algorithm applies the minimal cost

Input: Set of base classifiers $\mathcal{C} = \{C_1, C_2, ..., C_K\}$, validation set $\mathcal{V} = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}\}$, meta learning algorithm $\mathbf{A_{ML}}$, meta classifier $\mathbf{MC}$, decision tree algorithm $\mathbf{A_{DT}}$, throughput requirement (classifications/sec) T, convergence parameter $\delta(\delta > 0)$, stopping criteria $\epsilon(0 < \epsilon < \delta)$
Output: Pruned meta classifier $\mathbf{MC}^*$
Begin
$\mathcal{MLT} = \{< \vec{a}_1, \vec{a}_2, ..., \vec{a}_K, \vec{tc} >\}$, $\vec{a}_i[j] = C_i(\mathbf{x_j}) \forall i, \vec{tc}[j] =$ TrueClass($\mathbf{x_j}$), j=1, 2, ...,n, $x_j \in \mathcal{V}$;
$\mathcal{DTT} = \{< \vec{a}_1, \vec{a}_2, ...\vec{a}_K, \vec{pc} >\}$, predicted class $\vec{pc}[j] = \mathbf{MC}(\mathbf{x_j})$;
$\mathbf{DTM} = \mathbf{A_{DT}}(\mathcal{DDT})$, the decision tree model trained with $\alpha = 0$;
$K^* = K$; $\mathbf{DTM}^* = \mathbf{DTM}$; /* initialization of running variables */
$T^* = 1/$t, $t = max\{t_i | t_i = time(\mathbf{MC}(\mathbf{x_i}))$, i = 1, 2, ...,n }; /* Throughput estimate of **MC** */
While ($T^* < T$) **do**
$\delta^* = \delta$; $K^* = K^* - 1$; /* remove one classifiers at a time */
L_1: **While** $\mathbf{DTM}^*$ has more than K^* classifiers **do**
$\alpha = \alpha + \delta^*$; /* increase α to remove classifiers */
$\mathbf{DTM}^* = \underset{i}{argmin}\{[R_\alpha(\mathbf{DTM_i}) + \alpha \cdot C(\mathbf{DTM_i})] | \mathbf{DTM_i} =$ subtree of $\mathbf{DTM}\}$;
end while
if ($\mathbf{DTM}^*$ has too many classifiers pruned (i.e. less than K^*) and $\delta^* > \epsilon$)
reset $\alpha = \alpha - \delta^*$; adjust $\delta^* = \delta^*/2$; $\mathbf{DTM}^* = \mathbf{DTM}$; **goto** L_1;
$\mathcal{C} = \mathcal{C} - \hat{C}$, $\hat{C} =$ the classifier that is not included in $\mathbf{DTM}^*$;
$\mathcal{MLT}^* = \{< \vec{a}_{j_1}, \vec{a}_{j_2}, ..., \vec{a}_{j_{K^*}}, \vec{tc} >\}$, $C_{j_1}, C_{j_2}, ..., C_{j_{K^*}} \in \mathcal{C}$ (retained classifiers);
$\mathbf{MC}^* = \mathbf{A_{ML}}(\mathcal{MLT}^*)$;
$T^* = 1/$t, $t = max\{t_i | t_i = time(\mathbf{MC}^*(\mathbf{x_i}))$, i = 1, 2, ...,n };
end while
$\mathcal{MLT}^* = \{< \vec{a}_{j_1}, \vec{a}_{j_2}, ..., \vec{a}_{j_{K^*}}, \vec{tc} >\}$, $C_{j_1}, C_{j_2}, ..., C_{j_{K^*}} \in \mathcal{C}$;
$\mathbf{MC}^* = \mathbf{A_{ML}}(\mathcal{MLT}^*)$;
End

Table 1: Post-Training Pruning Algorithm

complexity pruning method to reduce the size of the decision tree and thus prune away additional base classifiers. The degree of pruning can be controlled by the complexity parameter α, as described in Section 3.1. (The loop in the figure corresponds to the search for the proper value of the α parameter). Since minimal cost complexity pruning eliminates first the branches of the tree with the least contribution to the decision tree's performance, the base classifiers pruned during this phase will also be the least "important" base classifiers.

5. A new meta-level training set is composed by using the predictions of the remaining base classifiers on the original validation set as attributes and the true class labels as target.

6. Finally, the original meta-learning algorithm is trained over this new meta-level training set to re-compute the pruned ensemble meta-classifier.

3.2.1 Remarks

The algorithm is guaranteed to terminate due to the finite structure of the decision tree and the bounded number of times ($\lceil \log_2(\delta/\epsilon) \rceil$) the inner loop will be executed. Moreover, Theorem 3.10 of [4] proves the monotonic relation between the decreasing size of the decision tree as the complexity parameter α increases.

The success of the post-training pruning algorithm depends on the degree the decision tree learning algorithm "overfits" the original meta-classifier. We choose to train the model the meta-classifier on the very same data that was used to compute it in the first place. In some sense this is equivalent to generating test suits of test sets to exercise all program execution paths. By modeling the original meta-classifier based on its responses to its own training set we ensure that the decision tree learning algorithm has access to the most inclusive information regarding the meta-classifier's behavior. Furthermore, we do not require a separate *pruning* data set.

This post-training pruning method is algorithm and representation independent, i.e. it does not examine the internal structure and strategies of the learning algorithms that generate the base classifiers, nor the assumptions and characteristics of the underlying data or their schema definitions. Instead, it treats the base classifiers as black boxes and relies on an independent training set that composes based on the responses of each base classifier.

Another benefit of the post-training pruning algorithm comes from the modeling of the original meta-classifier by a decision tree classifier. In general, an ensemble meta-classifier may have an internal representation that we cannot easily view or parse (except, of course, for its constituent base classifiers). By inducing a "decision tree equivalent" model, the algorithm generates an alternative and more declarative representation that we can inspect. Other related methods for describing and computing comprehensible models of ensemble meta-classifiers have been studied in the contexts of Knowledge Acquisition [11], Knowledge Probing [15] and meta-classifier correlation-based visualization tools [32].

Computing decision tree models as part of the post-training pruning algorithm are not only useful for pruning or for explaining the behavior of the meta-classifier. These intermediate models are also meta-classifiers and hence can too be used to classify unlabeled instances. Recall, that they are also trained over the predictions of the available base classifiers, albeit for a different target class. As a result, their predictive accuracy may be inferior to that of the original meta-classifier. On the other hand, in a distributed data mining system, such as JAM, where classifiers and meta-classifiers can be exchanged and used as black boxes, it may not be possible to prune the imported meta-classifiers to adhere to local constraints (e.g if the original meta-learning algorithm is not known or not accessible). In this case, it may be preferable to trade some of their predictive accuracy for the more efficient pruned decision tree meta-classifiers.

The next section describes a comprehensive set of experiments that evaluates our post-training pruning algorithm and attempts to shed light on these issues. Specifically, we address the following questions:

- How accurately can a decision tree-based algorithm learn the behavior of a meta-classifier of arbitrary representation?
- What is the penalty of using the decision-tree meta-classifiers instead of the original meta-classifiers?
- How robust is post-training pruning? What is the tradeoff between the predictive accuracy and the classification throughput of the meta-classifier as we increase the degree of pruning?

4. EMPIRICAL EVALUATION

We evaluated the post-training pruning algorithm on meta-classifiers trained to predict credit card transactions as legitimate or fraudulent. (Results of the pre-training pruning methods on the same task are reported in [30]. We obtained two large databases from Chase and First Union banks, members of FSTC (Financial Services Technology Consortium) each with 500,000 records (69 MBytes of data) of sampled credit card transaction data spanning one year. The schemas of the databases were developed over years of experience and continuous analysis by bank personnel to capture important information for fraud detection. The records have a fixed length of 137 bytes each and about 30 numeric and categorical attributes including the binary class label (fraud/legitimate transaction). Chase bank data consisted of 20% fraud and 80% legitimate transactions, whereas First Union data consisted of 15% versus 85% of fraud/legitimate distribution.

To compute the set of base classifiers we employed five different inductive learning programs, Bayes, C4.5, ID3, CART and Ripper. Bayes, implements a naive Bayesian learning algorithm described in [25], ID3 [33], its successor C4.5 [34], and CART [4] are decision tree based algorithms, and Ripper [8] is a rule induction algorithm.

We evaluated the various classification models from two perspectives, their predictive performance and their efficiency. To measure and compare predictive performance we used several different metrics including accuracy, TP-FP spread (TP and FP stand for True Positive, and False Positive respectively), and a cost model fit to the credit card fraud detection problem. Overall accuracy expresses the ability to give correct predictions, TP-FP denotes the ability to catch fraudulent transactions while minimizing false alarms and the cost model captures the performance of a classification model with respect to the goal of the target application (stop loss due to fraud). To compare the efficiencies of the various ensemble meta-classifiers we measured classification throughput. Throughput here denotes the rate at which a stream of data items can be piped through and labeled by a meta-classifier.

Due to space constraints, however, we limit this report to (mostly) accuracy and throughput results. The emphasis in this paper is not on presenting an effective fraud detection method, but rather on the evaluation of the post-training pruning algorithm as a general method for reducing the size of an ensemble meta-classifier. Detailed information on effective fraud detectors with extensive results (TP-FP spread and cost model) from the mining of these credit card data sets can be found in [31].

4.1 Computing Base Classifiers

The first step involves the training of the base classifiers. We split each data set in 12 subsets and distribute them across six different data sites (each site storing two subsets). Then we apply the 5 learning algorithms on each subset of data, therefore creating 60 classifiers (10 classifiers per data site). Next, each data site imports all "remote" base classifiers (50 in total) to test them against its "local" data. In essence, each classifier is evaluated on 5 different (and unseen) subsets.

Figure 2 presents the averaged accuracy results for the Chase (left plot) and First Union (right plot) credit card data, respectively. The x-axis represents the base classifiers, and each vertical bar corresponds to a single model (60 in total). The first set of 12 bars denotes Bayesian classifiers, the second set of 12 bars denotes C4.5 classifiers etc. Each bar in an adjacent group of 12 bars corresponds to a specific subset used to train the classifiers. For example, the first bar of the left plot represents the accuracy (83.5%) of the Bayesian classifier that is trained on the first data subset, the second bar represents the accuracy (82.7%) of the Bayesian classifier that is trained on the second data subset and the 13^{th} bar of the same plot (the first of the second group of 12 adjacent bars) represents the accuracy 86.8% of the C4.5 classifier that is trained on the first subset.

With the exception of two ID3 base classifiers on the Chase data set, all base classifiers demonstrate significantly better results comparing to the trivial mean predictor (80% for the Chase and 85% for the First Union data). Moreover, by combining these separately learned classifiers, it is possible to generate new models with improved fraud detection capabilities. Overall, it appears that all learning algorithms performed better on the First Union data set than on the Chase data set. On the other hand, note that there are fewer fraudulent transactions in the First Union data and this causes a higher baseline accuracy. Next, we combine these separately learned classifiers under the voting, stacking and SCANN strategies hoping to generate meta-classifiers with improved fraud detection accuracies.

4.2 Combining Base Classifiers

Although the sites are populated with 60 classification models, in meta-learning they combine only the 50 "imported" base classifiers. Since each site uses its local data to meta-learn (the first subset as the validation set and the second subset as the test set) it avoids using the 10 "local" base classifiers to ensure that no classifiers predict on their own training data. We applied 5 different meta-learning methods at each site: the 2 voting strategies (majority and weighted), the 2 stacking strategies corresponding to the Bayes and Ripper learning algorithms and the SCANN al-

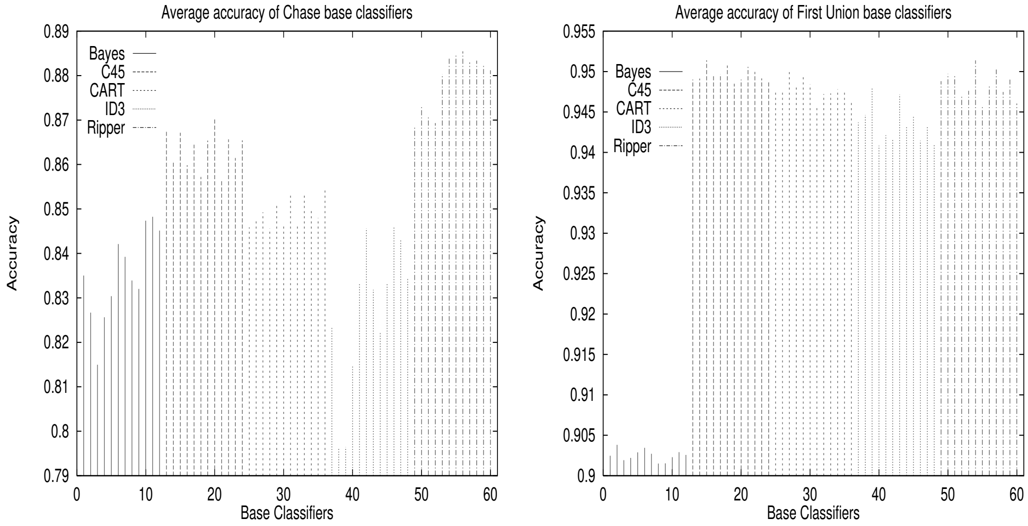

Figure 2: Accuracy of base classifiers on Chase(left) and First Union (right) data.

Table 2: Performance of the meta-classifiers

Bank data	Majority Voting	Weighted Voting	Bayes Stacking	Ripper Stacking	SCANN
Chase	89.58%	89.59%	88.65%	89.66%	89.74%
First Union	96.16%	96.19%	96.21%	96.53%	96.44%

gorithm. Since all sites meta-learn the base classifiers independently, the setting of this experiment corresponds to a 6-fold cross validation with each fold executed in parallel.

The performance results of these meta-classifiers averaged over the 6 sites are reported in Table 2. A comparison to Figure 2 indicates that meta-classifiers outperform all base classifiers, and in most cases by a significant margin. Due to the size of the data sets, the empirical results exhibit a very low variance. In general, a 0.1% accuracy difference between two methods is a statistically significant result with 99% confidence according to the 2-tailed paired t test. On the other hand, the meta-classifiers are composed of 50 base classifiers suggesting a significant increase in resource requirements (e.g memory, CPU time) and substantial reduction in classification throughput. Post-training pruning seeks to address this issue. The objective is to remove a subset of the constituent base classifiers while preserving the accuracy gains.

4.3 Post-training pruning experiments

The first two phases of the post-training pruning seek to learn and prune a decision tree model of the original meta-classifier. Initially, we applied the CART learning algorithm on the decision tree training set of each meta-classifier. The initial decision-tree models were generated with the complexity parameter α set to zero to allow the growth of trees that are as accurate and as "close" to the original meta-classifiers as possible. Then, by increasing the α parameter (i.e. the degree of pruning), we gradually computed trees with fewer nodes (thus pruning base classifiers). One can quantify the effectiveness of the derived models by comparing their predictions against the predictions of their corresponding meta-classifiers on the separate test sets.

The accuracy of the decision tree models and the impact of pruning on their performance is shown in Figure 3. Each plot (left for Chase, right for First Union) displays the average accuracy (y-axis) of the decision tree algorithm in imitating the behavior of the 5 different methods for constructing ensembles. The x-axis represents decision-tree models of progressively smaller sizes. The initial (unpruned) model corresponds to the left-most points of each curve (i.e. degree of pruning is 0%). Naturally, an accuracy result of 100% would signify a decision tree model that has perfectly learned to imitate its meta-classifier. According to this figure, learning the behavior of these meta-classifiers has been fairly successful, with decision tree models achieving a 98%-99% accuracy rate, even when pruning is as heavy as 80%.

The last phase of the post-training pruning algorithm aims to re-combine the remaining base classifiers (those included in the decision tree model) using the original meta-learning algorithm. To evaluate the effectiveness of the pruning algorithm, we measured the accuracy of all the intermediate meta-classifiers obtained as we increased the degree of pruning from 0% (original unpruned meta-classifier) to 100% (all base classifiers are pruned, default prediction is the most frequent class, i.e. legitimate transaction).

In Figure 4 we present the accuracy results for the Chase (left side plots) and the First Union (right side plots) meta-classifiers. The plots at top row correspond to the stacking meta-learning methods (Bayes, Ripper), the plots at the center row represent the voting (majority, weighted) meta-learning methods and the plots at the bottom row belong to the SCANN meta-learning method. These graphs demonstrate that post-training pruning was quite successful in all cases examined. The algorithm determined and pruned the

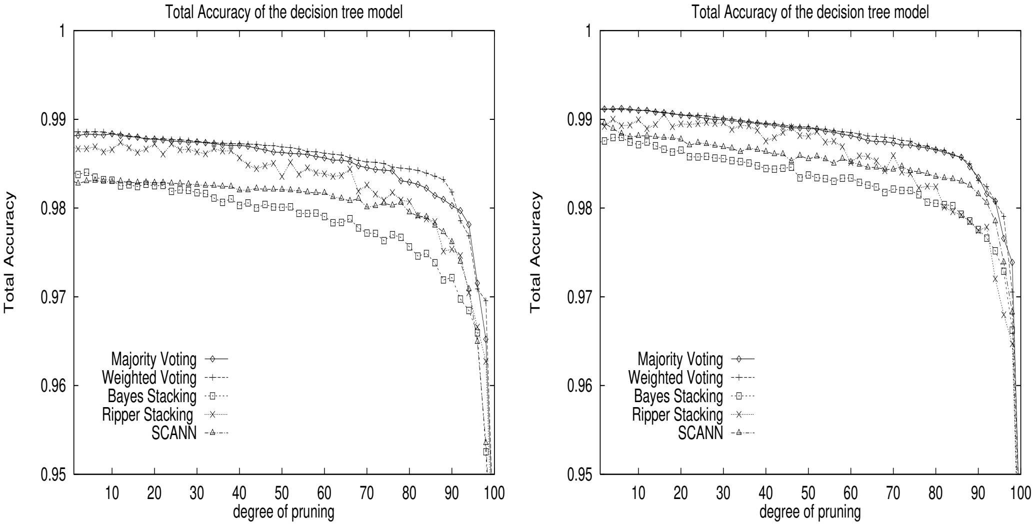

Figure 3: Accuracy of decision tree models in emulating the behavior of Chase (left) and First Union (right) meta-classifiers

redundant and/or the less "contributing" base classifiers, and generated substantially smaller meta-classifiers without significant performance penalties.

The graphs also depict the overall accuracy of the decision tree models of the meta-classifiers. Recall, that these are models trained to "imitate" the behavior of an ensemble, not to directly detect fraudulent transactions. As expected, their predictive performance is inferior to that of their corresponding (pruned) meta-classifier. On the other hand, it is interesting to note that it is possible to compute decision tree models that outperform other original meta-classifiers. In these experiments, for example, the decision tree model of Ripper (and those of the voting meta-classifiers) outperforms the original Bayesian meta-classifier.

To further explore the last observation, we also measured the accuracy of the original stacking CART meta-classifier as a function of the degree of pruning. In other words, we studied the direct performance of a decision tree learning algorithm as a method of combining base classifiers. The results are depicted by the curve denoted as "CART stacking". Surprisingly, CART appears to be more effective when learning to combine base classifiers indirectly (e.g. by observing the ripper meta-classifier) than through direct learning of this target concept. This suggests that searching the hypothesis space as modeled by a previously computed meta-classifier may, in some cases, be easier than searching the original hypothesis space. It is not entirely clear why this may be so, and is a subject of further study. As a result, in cases where meta-classifiers are considered as black boxes, or meta-learning algorithms are not available, it may be beneficial to compute, prune and use their "decision tree equivalents" instead.

4.4 Predictive performance vs. throughput

The degree of pruning of a meta-classifier within a data site is dictated by the classification throughput requirements of the particular problem. In general, higher throughput requirements necessitate heavier pruning. The last set of experiments investigates the trade-off between throughput and predictive performance.

To normalize the different evaluation metrics and better quantify the effects of pruning, we measured the ratio of the performance improvement of the pruned meta-classifier over the performance improvement of the complete (original) meta-classifiers. In other words, we measured the performance *gain* ratio $G = \frac{P_{PRUNED} - P_{BASE}}{P_{COMPLETE} - P_{BASE}}$, where P_{PRUNED}, $P_{COMPLETE}$ and P_{BASE} denote the performance (e.g. accuracy, TP-FP spread, cost savings) of the pruned meta-classifier, the complete meta-classifier and the best base classifier, respectively. Values of $G \simeq 1$ indicate pruned meta-classifiers that sustain the performance levels to that of the complete meta-classifier while values of $G < 1$ indicate performance losses. When only the best base classifier is used, there is no performance improvement and $G = 0$.

Figure 5 demonstrates the algorithm's effectiveness on the Chase (left) and First Union (right) classifiers by displaying the predictive performance and throughput of the pruned Ripper stacking meta-classifiers as a function of the degree of pruning. In this figure, we have also included the performance results of the meta-classifier with respect to two more evaluation metrics, the TP-FP spread, and the savings due to timely fraud detection. Similar results have been obtained for the other meta-classifiers, but are not shown here due to space considerations. The black colored bars represent the accuracy gain ratios, the dark gray colored bars represent the TP-FP gain ratios and the light gray bars represent the savings gain ratios of the pruned meta-classifier. The very light gray bars correspond to the relative throughput of the pruned meta-classifier T_P to the throughput of the complete meta-classifier T_C. To estimate the throughput of the meta-classifiers, we measured the time needed for a meta-classifier to generate a prediction. This time includes the time required to obtain the predictions of the constituents base classifiers sequentially on an unseen credit card transaction, the time required to assemble these predictions into a single meta-level "prediction" vector and the time required for the meta-classifier to input the vector and

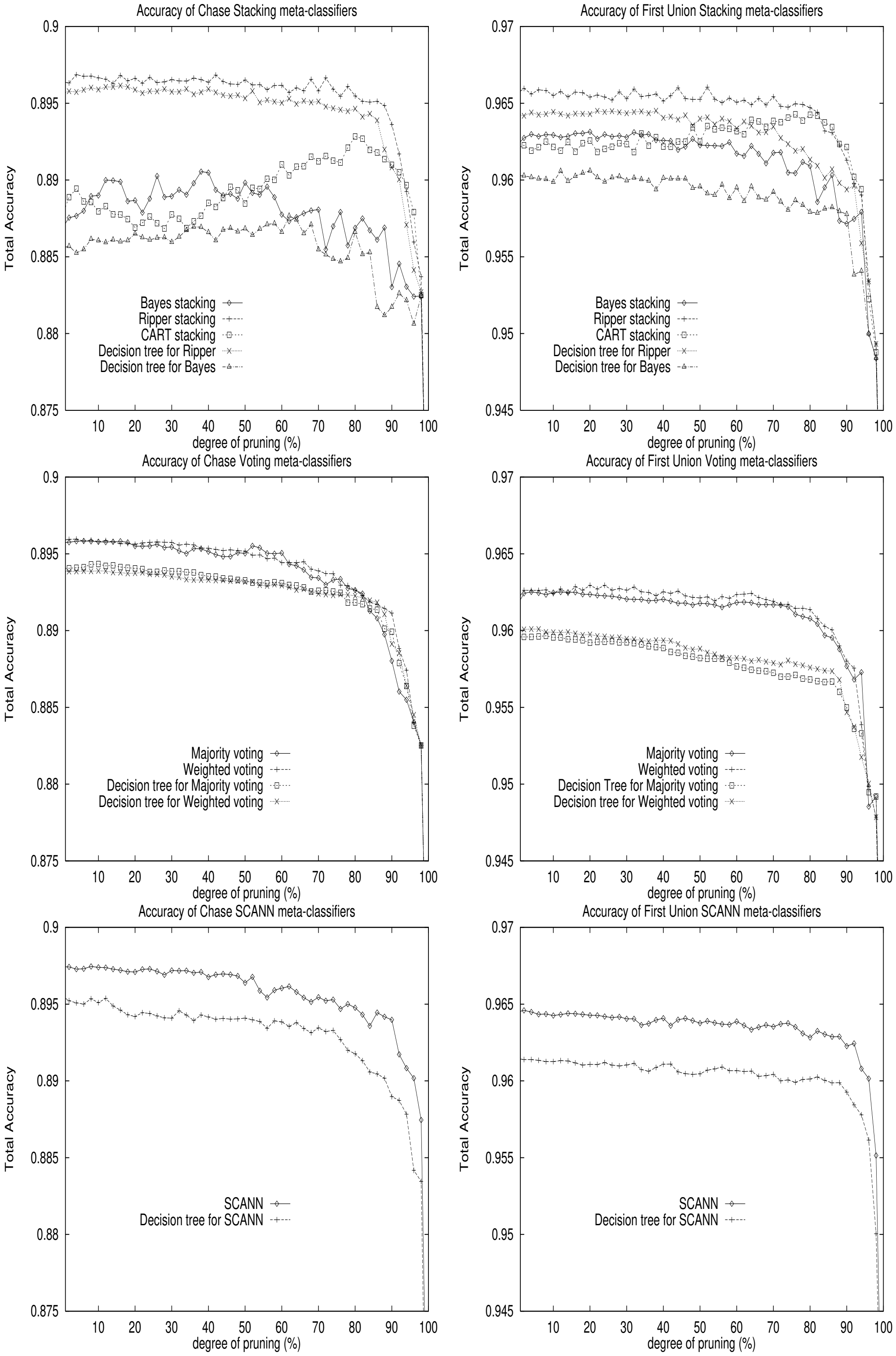

Figure 4: Accuracy of the Stacking (top plots), Voting (middle plots) and SCANN (bottom plots) meta-classifiers and their decision tree models for Chase (left) and First Union (right) data

generate the final prediction. The measurements were performed on a Personal Computer with a 200MHz Pentium processor running Solaris 2.5.1. These measurements show that cost-complexity pruning is successful in finding Chase

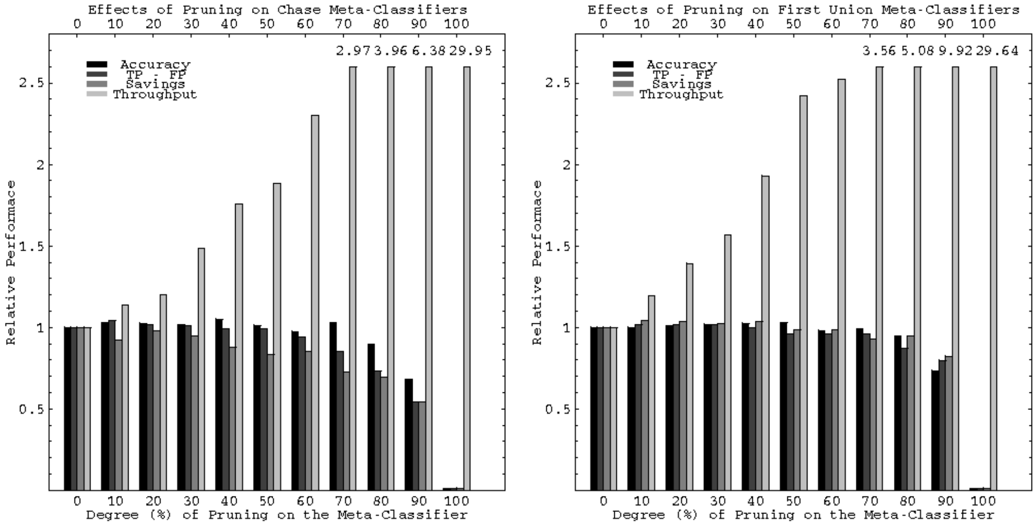

Figure 5: Bar charts of the accuracy (black), TP-FP (dark gray), savings (light gray) and throughput (very light gray) of the Chase (right) and First Union (left) meta-classifiers as a function of the degree of pruning.

meta-classifiers that retain their performance levels to 100% of the original even with as much as 60% of the base classifiers pruned or within 60% of the original with 90% pruning. At the same time, the pruned classifiers exhibit 230% and 638% higher throughput. For the First Union base classifiers, the results are even better. With 80% pruning, the pruned meta-classifiers have gain ratios $G \simeq 1$ and with 90% pruning they are within 80% of the original performance. The throughput improvement in this case is 5.08 and 9.92 times better, respectively.

In absolute numbers, the best meta-classifier achieves on average 89.66% accuracy, 0.632 TP-FP spread and saves $903K per data subset under a realistic cost model [31] (maximum savings: $1,470) and the best First Union meta-classifiers achieves 96.53% accuracy, 0.848 TP-FP spread and $950K per data subset (maximum savings: $1,085). In contrast, the best base classifier in Chase is 88.5% accurate, with 0.551 TP-FP spread and $812K in savings while the best First Union base classifier is 95.2% accurate with 0.749 TP-FP spread and $806K in savings.

5. CONCLUDING REMARKS

Constructing ensembles of classifiers computed over separate data sets is a scalable and effective solution to learning over large and distributed databases. In this paper, we addressed a shortcoming of this approach that has largely been ignored, the increased demand for run-time system resources. The final ensemble meta-classifier may consist of a large collection of base classifiers that require significantly more memory resources, while substantially slowing down classification throughput. We described a pruning algorithm, called post-training pruning, that seeks to determine the base classifiers that are redundant or "less-important" to the classification process of the ensemble meta-classifier. Thus, given a set of system constraints and requirements, the algorithm computes a smaller and faster ensemble meta-classifier with predictive performance that is comparable to the original meta-classifier.

We evaluated the post-training pruning algorithm on real credit card transaction data sets provided by two separate institutions. Our results demonstrated that our algorithm is successful and highly robust in computing pruned meta-classifiers that preserve the predictive performance of the original unpruned meta-classifiers even when system constraints call for heavy pruning. This line of work, however, is far from being complete; future research plans include studying of alternative algorithms and methods for exploring (analytically, empirically and graphically) the relationships among base classifiers and meta-classifiers. To our knowledge, there have been little attempts to incorporate resource requirements in the process of forming ensemble classifiers, even though such constraints are common place for operational systems.

6. ACKNOWLEDGMENTS

We are in debt to Chris Merz for providing us with his implementation of the SCANN algorithm.

7. REFERENCES

[1] K. Ali and M. Pazzani. Error reduction through learning multiple descriptions. *Machine Learning*, 24:173–202, 1996.

[2] L. Breiman. Heuristics of instability in model selection. Technical report, Department of Statistics, University of California at Berkeley, 1994.

[3] L. Breiman. Stacked regressions. *Machine Learning*, 24:41–48, 1996.

[4] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, Belmont, CA, 1984.

[5] P. Chan and S. Stolfo. Meta-learning for multistrategy and parallel learning. In *Proc. Second Intl. Work. Multistrategy Learning*, pages 150–165, 1993.

[6] P. Chan and S. Stolfo. Toward parallel and distributed learning by meta-learning. In *Working Notes AAAI*

Work. Knowledge Discovery in Databases, pages 227–240, 1993.

[7] P. Chan, S. Stolfo, and D. Wolpert, editors. *Working Notes for the AAAI-96 Workshop on Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms*, Portland, OR, 1996.

[8] W. Cohen. Fast effective rule induction. In *Proc. 12th Intl. Conf. Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[9] M.W. Craven and J. J. W. Shavlik. Extracting tree-structured representations of trained networks. *Advances in Neural Information Processing Systems*, 8, 1996.

[10] T.G. Dietterich. Machine learning research: Four current directions. *AI Magazine*, 18(4):97–136, 1997.

[11] P. Domingos. Knowledge acquisition from examples via multiple models. In *Proc. Fourteenth Intl. Conf. Machine Learning*, pages 98–106, 1997.

[12] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37. Springer-Verlag, 1995.

[13] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proc. Thirteenth Conf. Machine Learning*, pages 148–156, 1996.

[14] Michael J. Greenacre. *Theory and Application of Correspondence Analysis*. Academic Press, London, 1984.

[15] Y. Guo and J. Sutiwaraphun. Knowledge probing in distributed data mining. In P. Chan H. Kargupta, editor, *Work. Notes KDD-98 Workshop on Distributed Data Mining*, pages 61–69. AAAI Press, 1998.

[16] R.A. Jacobs, M.I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixture of local experts. *Neural Computation*, 3(1):79–87, 1991.

[17] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

[18] M. I. Jordan and L. Xu. Convergence results for the em approach to mixtures of experts architectures. In *AI memo 1458*, 1993.

[19] E. B. Kong and T. Dietterich. Error-correcting output coding corrects bias and variance. In *Proc. Twelfth Intl. Conf. Machine Learning*, pages 313–321, 1995.

[20] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Info. Proc. Sys. 7*, pages 231–238. MIT Press, 1995.

[21] M. LeBlanc and R. Tibshirani. Combining estimates in regression and classification. Technical Report 9318, Department of Statistics, University of Toronto, Toronto, ON, 1993.

[22] D. Margineantu and T. Dietterich. Pruning adaptive boosting. In *Proc. Fourteenth Intl. Conf. Machine Learning*, pages 211–218, 1997.

[23] C. Merz. Using correspondence analysis to combine classifiers. *Machine Learning*, 1999. In press.

[24] C. Merz and M. Pazzani. A principal components approach to combining regression estimates. *Machine Learning*, 1999. In press.

[25] M. Minksy and S. Papert. *Perceptrons: An Introduction to Computation Geometry*. MIT Press, Cambridge, MA, 1969. (Expanded edition, 1988).

[26] T. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[27] D. W. Opitz and J. J. W. Shavlik. Generating accurate and diverse members of a neural-network ensemble. *Advances in Neural Information Processing Systems*, 8:535–541, 1996.

[28] J. Ortega, M. Koppel, and S. Argamon-Engelson. Arbitrating among competing classifiers using learned referees. *Machine Learning*, 1999. in press.

[29] M. P. Perrone and L. N. Cooper. When networks disagree: Ensemble methods for hybrid neural networks. *Artificial Neural Networks for Speech and Vision*, pages 126–142, 1993.

[30] A. L. Prodromidis and S. J. Stolfo. Pruning meta-classifiers in a distributed data mining system. In *Proc of the First National Conference on New Information Technologies*, pages 151–160, Athens, Greece, October 1998. Extended version.

[31] A. L. Prodromidis and S. J. Stolfo. Agent-based distributed learning applied to fraud detection. CUCS-014-99, 1999.

[32] A. L. Prodromidis, S. J. Stolfo, and P. K. Chan. Effective and efficient pruning of meta-classifiers in a distributed data mining system. Technical report, Columbia Univ., 1999. CUCS-017-99.

[33] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[34] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[35] R.J Quinlan and R.L. Rivest. Inferring decision trees using the minimum description length princliple. *Information and Computation*, 80:227–248, 1989.

[36] Waterhouse S. R. and Robinson A. J. Classification using hierarchical mixtures of experts. In *IEEE Workshop on Neural Networks for Signal Processing IV*, pages 177–186, 1994.

[37] R. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–226, 1990.

[38] S. Stolfo, A. Prodromidis, S. Tselepis, W. Lee, W. Fan, and P. Chan. JAM: Java agents for meta-learning over distributed databases. In *Proc. 3rd Intl. Conf. Knowledge Discovery and Data Mining*, pages 74–81, 1997.

[39] Volker Tresp and Michiaki Taniguchi. Combining estimators using non-constant weighting functions. *Advances in Neural Information Processing Systems*, 7:419–426, 1995.

[40] D. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.