Sharing Classifiers among Ensembles from Related Problem Domains

Yi Zhang
yi-zhang-2@uiowa.edu

W. Nick Street
nick-street@uiowa.edu

Samuel Burer
samuel-burer@uiowa.edu

Department of Management Sciences, the University of Iowa
Iowa City, IA 52242-1000

Abstract

A classification ensemble is a group of classifiers that all solve the same prediction problem in different ways. It is well-known that combining the predictions of classifiers within the same problem domain using techniques like bagging or boosting often improves the performance. This research shows that sharing classifiers among different but closely related problem domains can also be helpful. In addition, a semi-definite programming based ensemble pruning method is implemented in order to optimize the selection of a subset of classifiers for each problem domain. Computational results on a catalog dataset indicate that the ensembles resulting from sharing classifiers among different product categories generally have larger AUCs than those ensembles trained only on their own categories. The pruning algorithm not only prevents the occasional decrease of effectiveness caused by conflicting concepts among the problem domains, but also provides a better understanding of the problem domains and their relationships.

1. Introduction

A classification ensemble is a multi-classifier system in which each member classifier tries to solve the same classification problem. Those classifiers can be trained on the same data set using different base algorithms [8, 7], or trained on different subsets of data using the same base algorithm, e.g. bagging [2], boosting [11], arcing [3], or even trained on the same data set using the same base algorithm but with different parameters [20, 4]. The classification of the ensemble is obtained by combining the classifications of its members. The simplest method of aggregation is voting, in which the class label receiving most votes is assigned to the data point. Both empirical and theoretical studies have shown that an ensemble is likely to have smaller generalization errors than a single classifier [14, 1].

Ensemble methods not only improve classification accuracy, but also provide a new way to share knowledge. If data are distributed at different locations, classifiers can be trained locally then combined together to create an ensemble. This centralized ensemble gathers information from each site and can potentially be more powerful for further predictions. An empirical study of this method was conducted for credit card fraud detection [8]. The banks participating in the research shared their individual fraud detection models with each other. Computational results showed that pooling knowledge in the form of an ensemble did increase the chance of hitting the target. There has been other research on sharing classifiers within an institution or among different institutions [9, 16]. However, they all required that the data be from the same problem domain, or in other words, the inducted classifiers are all solving the same classification problem. Now the question is, if we are working with several different but closely related problem domains, is sharing classifiers among those domains still a good idea?

The essence of sharing classifiers is sharing common knowledge among different but closely related problem domains. A famous example of the research in this direction is the multi-task neural network [6]. Each problem domain is assigned one or more output nodes, while sharing some of the input and mid-layer nodes with other problem domains. Although this method is sometimes successful, it has several limitations. First, it requires that the data be at the central location for training. Second, the training speed becomes a big issue if the size of the data is large. Our proposed classifier-sharing strategy may avoid both of the drawbacks. Since the classifiers can be trained locally, the data need not be centralized. Moreover, one can use efficient algorithms like decision trees to train classifiers, so computation time becomes less of a problem.

The prerequisite for sharing classifiers among different problem domains is that the data schema for each domain should be identical, or at least very similar. This ensures that classifiers trained on one problem can also be applied to others, although the accuracy can possibly be low. Sometimes, it transformation of variables is required to satisfy this condition. It is hard to tell a priori whether sharing classifiers among different problem domains will improve the overall performance. The success of this strategy depends on the connections among the problem domains and the self-completeness of information within each problem domain. However, with a screening process that will be described later, the chance that sharing classifiers will eventually downgrade the overall performance is minimal.

We tested this classifier-sharing strategy on a publicly available marketing data set. To address the concern that sharing classifiers blindly may sometimes do harm to some of problem domains if they have conflicting concepts, we use a mathematical programming model to select a good subset of classifiers from the entire ensemble for each domain. The pruned ensembles work almost as well as the sharing-all ensemble. In addition, the classifier-sharing structure derived by the pruned ensembles reveals some of the special properties of the problem domains.

The rest of the paper is organized as follows. Section 2 describes the marketing data set used for the research and the data mining task. Section 3 presents the classical bagging approach and explains its partial failure on this particular marketing problem. Section 4 introduces the idea of sharing classifiers and compares its performance with that of the non-sharing strategy. Section 5 briefly describes the subset selection algorithm and demonstrates its effectiveness by computational results. Section 6 discusses some of the properties of the problem domains revealed by the selection process. Section 7 concludes the paper.

2. Data Set and Data Mining Task

The data set used to evaluate the idea of sharing classifiers among different problem domains is a public marketing data set from the Direct Marketing Association (DMEF Academic Data Set Three, Specialty Catalog Company, Code 03DMEF). It was contributed by a specialty catalog company that sells household goods such as clothes and kitchenwares to the customers receiving its catalogs by mail. The company has collected twelve years of purchase and promotion data (such as availability of coupon or discount) from its customer base which is composed of $103,715$ individuals. It is also possible to derive some demographic features (such as income level, home ownership, etc.) from the zip code that the original data set provides. The dependent variables are the customers' responses to a particular promotion held by the company. There are 19 different categories of products involved in the promotion, which correspond to 19 response variables. Unfortunately, the identification of the product categories are not available. The data mining problem we try to solve is to predict which categories of products a customer is going to buy based on the available historical and demographic data. We decomposed this task into 19 separate binary classification problems, building one model for each category and predicting whether the customer is going to buy from that category. Note that this whole task cannot be treated as a typical classification problem with 19 classes because one customer can buy products from any number of categories, including zero, so the class assignment for each individual is not exclusive. This kind of problem is sometimes referred to as "multi-label" classification problem and is more often seen in the text-mining domain [18].

Table 1. Response rate of 19 categories.

Category	%Pos	Category	%Pos
1	0.14	11	0.13
2	0.27	12	0.05
3	0.04	13	0.44
4	0.92	14	2.65
5	0.12	15	1.70
6	0.83	16	2.09
7	0.44	17	0.65
8	0.37	18	0.31
9	0.64	19	1.17
10	0.02		

Table 1 shows the percentage of positive points, or response rate, of each category. A positive point for a category represents a customer that buys one or more products from that category. For most categories, the response rate is lower than one percent, which makes the data set highly skewed toward non-buyers. For a data set with highly biased priors, it is often not appropriate to build classifiers directly on the original data, since one will very likely end up with a classifier that always predicts the majority class. For our marketing problem, most classifiers will always predict that the customer is not going to buy. Although such a classifier will be 99% accurate, it is of no value since the ultimate goal of this data mining effort is to identify potential customers. Previous research also indicate that such a classifier is not a good ranker, either [10]. A common way to deal with this problem is to create training data sets with balanced class distribution through sampling, which has a similar effect to introducing a cost structure that places higher misclassification cost on false negatives. As the cost structure of this problem is unknown, we modify the sampling priors as detailed in the next section.

3. Baseline: Bagging Individual Categories

Since the priors of the data set are highly biased, it is necessary to construct training sets with balanced class distributions, which we achieve by bootstrap sampling. Considering the size of the data and the processing speed of the learning algorithm, we fix the size of the bootstrapped training set at 800. Empirical evidence has shown that a balanced prior is optimal for the kind of marketing problems under consideration [22]. Therefore, 400 positive points and 400 negative points are sampled with replacement from the original data. Since the number of positive points is far less than 400 for most categories, the training sets contain many repeated points. On the other hand, the number of negative points is much greater than 400. To get enough information from the original data, especially the negative class, the bootstrapping process is repeated twenty-five times, resulting in twenty-five different training sets for each category. A C4.5 decision tree [19] is then built for each training set. In total, twenty-five decision trees are obtained for each category. These twenty-five trees are grouped together to create an ensemble for future predictions. This whole approach is very close to bagging [2] so we refer to it as the original bagging approach.

The bagging approach is a standard way to solve such marketing problems and often very powerful. However, it does not work well on the marketing data set used in this study. For instance, the AUC (Area Under the ROC Curve) for Category 10 is only 0.51 (see Table 2), which implies that the ensemble learned hardly anything useful from the training sets. (AUC is a commonly used measurement to evaluate how good a classifier ranks the data points. It can be interpreted as the probablity that a positive point will be ranked higher than a negative point.) Our explanation is that the number of distinctive positive points in the training sets for Category 10 is too small. As shown in Table 1, the original response rate for Category 10 is only 0.02%, so there is simply not enough information to build a decent classifier.

4. Sharing Classifiers among Categories

To improve the performance of the ensembles built by the original bagging approach, extra useful information is necessary. Sharing classifiers among different categories is our proposed solution. There are two reasons that can support our proposition. First, we are dealing with customers from the same catalog company. It is reasonable to expect that those customers who placed orders should share some common properties and those properties should be reflected in the classifiers belonging to those categories with relatively higher response rate. If these classifiers can be included into the ensembles of those categories without enough positive points, they may help improve the overall performance. Second, the purchase patterns of some different categories may be similar. For example, if people are more likely to buy clothes when there are discount coupons, the same may also be true for shoes. Therefore, a predictive model for clothes that stresses the importance of discount coupons may also work for shoes although they belong to different product categories.

The original data schema for each category is identical: same independent variables (purchase history, promotions and demographic information) and a binary dependent variable indicating whether the customer buys products from this category. However, some of the independent variables are category-specific. For example, there are 19 variables each representing presence of a discount for each category. Intuitively, the discount variable for Category i is more informative to the prediction problem for Category i than for other categories, and is likely used in the decision trees built for Category i. Since this discount variable is probably (not absolutely) not relevant to other categories, the decisions trees induced on Category i are less likely to be useful for other categories. To make the decision trees more interchangeable, we did some transformations on those category-specific variables. In the data set for Category i, a copy of each category-specific variable related to Category i is appended to the end of the data schema and labeled as “xxxx-for-this-category”. The values of the original category-specific variables for this Category i (which already have copies) are then set to be uniform for each record so that there will be no splits on these variables in the decision trees. The splits, if necessary, will be made on those newly appended variables. After transformation, each category has the same number of appended category-specific variables related only to itself so the data schema of each category is still identical and the tree splits on the category-specific variables are more meaningful across categories. For instance, if there is a rule induced on a category like “if there is a discount on this category, the customer is going to buy products from this category”, it is reasonably applicable to other categories. Therefore, the interchangeability of trees among categories is increased.

A naive way of sharing classifiers is to pool all the classifiers from the 19 categories into one big ensemble and use it for every category. The performance of the including-all ensemble generated by the sharing-all strategy and the individual ensembles from the non-sharing bagging approach in terms of AUC can be looked up in Table 2. The figures shown in the table are the average and the standard deviation (in parentheses) over five runs with different training-testing splits. Win-loss-tie comparison statistic of the sharing-all strategy against non-sharing is 12-7-0 by mean value and 6-4-9 by paired t test (95% significance level). It is evident that for most of the categories, the including-all ensemble performs better than the ensemble composed of classifiers trained only on its own data. Especially for those categories with very low response rates, such as Category 10, the rise in AUC is substantial. Figure 1 plots AUC percentage difference of the two strategies on all the 19 categories.

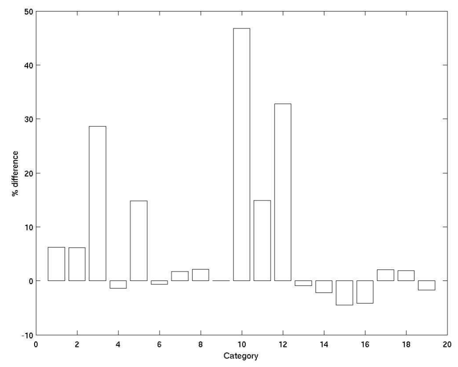

Figure 1. % AUC difference of including-all ensemble and original bagging ensembles

5. Selecting a Good Subset of Classifiers

We have shown that sharing classifiers among related problem domains can improve predicative performance. However, there are risks behind the sharing-all strategy. Specifically, when there are strikingly conflicting concepts among those problem domains, mixing classifiers blindly will degrade the effectiveness of the ensemble method. In fact, there is already some evidence in the results obtained so far. Table 2 shows that the including-all ensemble performs significantly worse on Categories 15 and 16 than the original ensembles. This is because the original ensembles on these categories are already good enough. Adding more classifiers that are either non-informative or even counter-conceptual eventually downgrades the ensemble. Therefore, it is safer to bring in a screening process that is able to select a subset of classifiers for each problem domain.

There has been some research work on subset selection for ensemble optimization, sometimes called ensemble pruning [17, 8, 21, 7]. However, most of the algorithms invented so far are some kind of greedy search, which picks classifiers sequentially based on some quality measurements. Here, we use a newly invented method to select a fixed number of classifiers by semi-definite programming (SDP), which is able to produce good solutions based on theoretical and empirical studies.

As literature has shown, a good ensemble should be composed of classifiers that are not only accurate by themselves, should also make different errors [15, 17, 4]. These two properties are often referred to as strength and divergence. Essentially, the semi-definite programming process is looking for a fixed-size subset of classifiers with the best strength and divergence trade-off.

A set-aside tuning set reflecting the original class distribution is used for subset selection. Since there are 19 categories in the marketing data, there will be 19 tuning sets and the subset selection process will be repeated 19 times, once for each category.

Now we explain how the subset selection is done by semi-definite programming. First, record the predictions of each classifier on the tuning set in matrix P as follows:

$$\begin{aligned} P_{ij} =& 0, \quad \text{if } j\text{th classifier is correct on data point } i, \\ P_{ij} =& 1, \quad \text{otherwise.} \end{aligned}$$

Let $G = P^T P$. Thus, the diagonal term G_{ii} is the total number of errors made by classifier i and the off-diagonal term G_{ij} is the number of common errors of classifier pair i and j. To put all the elements of the G matrix on the same scale, we normalize them by

$$\begin{aligned} \tilde{G}_{ii} =& \frac{G_{ii}}{n}, \\ \tilde{G}_{ij} =& \frac{G_{ij}}{\min\left(G_{ii}, G_{jj}\right)}, \end{aligned}$$

where n is the number of tuning points. After normalization, all elements of the $\tilde{G}$ matrix are between 0 and 1.

The constructed $\tilde{G}$ matrix captures both the strength and the divergence of an ensemble with its diagonal and off-diagonal elements, respectively. Clearly, for a good ensemble, all elements of the $\tilde{G}$ matrix should be small.

The $\tilde{G}$ matrix requires one further modification. Since the tuning data have the original class distribution, which is highly biased towards non-buyers, the resulting $\tilde{G}$ matrix will reflect the performance of the ensemble on the negative points while almost ignoring the positive points. It is necessary to balance the weights of the two classes in $\tilde{G}$. To achieve this goal, we define a third $\hat{G}$ matrix as a convex combination of the $\tilde{G}$ matrix on the positive points and the $\tilde{G}$ matrix on the negative points in the tuning set,

$$\hat{G} = \lambda \tilde{G}_{pos} + (1 - \lambda)\tilde{G}_{neg}.$$

In the following computational experiments, λ is set as 0.5.

Now we can formulate the subset selection problem as a quadratic integer programming problem. Essentially, we are looking for a fixed size subset of classifiers, the sum of whose corresponding elements in the $\hat{G}$ matrix is minimized. The mathematical programming formulation is

$$\begin{aligned} \min \quad & x^T \hat{G} x \\ s.t. \quad & \sum_i x_i = k, \\ & x_i \in \{0,\ 1\}. \end{aligned}$$

The binary variable x_i represents whether the ith classifier will be picked. If $x_i = 1$, the ith classifier is included in the selected set, and its corresponding diagonal and off-diagonal elements will be counted in the objective function.

The cardinality constraint $\sum_i x_i = k$ is mathematically important because without it, there is only one trivial solution to the problem with none of the classifiers picked. In addition, it gives us control over the size of the selected subset.

This quadratic integer programming problem is a standard 0-1 optimization problem, which is NP-hard in general. However, after some transformation, it can be relaxed into a semi-definite programming (SDP) problem. The key point of the relaxation is to relax the binary variable x_i into a unit vector. This relaxation requires that $x_i \in \{-1, 1\}$. So we need to make a transformation of variables

$$y_i = \frac{x_i + 1}{2},$$

and $y_i \in \{-1, 1\}$. Now the objective function becomes

$$\frac{1}{4}(y + \vec{e})^T \hat{G}(y + \vec{e}),$$

where $\vec{e}$ is a column vector of all 1s. The cardinality constraint $\sum_i x_i = k$ can be rewritten into quadratic form

$$x^T I x = k,$$

where I is the identity matrix. After variable transformation, this constraint becomes

$$(y + \vec{e})^T I(y + \vec{e}) = 4k.$$

A variable expansion trick can be applied to put both the transformed objective function and the cardinality constraint back to a nice quadratic form. We expand the variable vector $y = (y_1, y_2, ..., y_n)$ into $y = (y_0, y_1, y_2, ..., y_n)$, and let $y_0 = 1$. We then construct a new matrix

$$H_{(n+1)\times(n+1)} = \begin{pmatrix} \vec{e}^T \hat{G} \vec{e} & \vec{e}^T \hat{G} \\ \hat{G}\vec{e} & \hat{G} \end{pmatrix}.$$

Thus the objective function is equivalent to $y^T H y$. We then use the same trick to construct a matrix D

$$D_{(n+1)\times(n+1)} = \begin{pmatrix} n & \vec{e}^T \\ \vec{e} & I \end{pmatrix},$$

so that the cardinality constraint becomes $y^T D y = 4k$.

After this transformation, the problem formulation becomes

$$\begin{aligned} \min \quad & y^T H y \\ s.t. \quad & y^T D y = 4k, \\ & y_0 = 1, \\ & y_i \in \{-1,\ 1\},\ \forall i \neq 0, \end{aligned}$$

which is equivalent to

$$\begin{aligned} \min \quad & H \bullet yy^T \\ s.t. \quad & D \bullet yy^T = 4k, \\ & y_0 = 1, \\ & y_i \in \{-1,\ 1\},\ \forall i \neq 0, \end{aligned}$$

where $A \bullet B = \sum_{i,j} A_{ij} B_{ij}$.

Now we can invoke the SDP relaxation and the relaxed problem is

$$\begin{aligned} \min \quad & H \bullet Y \\ s.t. \quad & D \bullet Y = 4k, \\ & Y_{ii} = 1,\ \forall i \\ & Y \succeq 0, \end{aligned}$$

where $Y \succeq 0$ implies that Y is positive semi-definite.

There are efficient algorithms to solve such an SDP problem [5] and through a randomization process, the solution can be transformed back to a binary vector [12, 13]. In our computational experiment, selecting 25 out of 475 classifiers usually took 2-4 minutes. More details of the SDP-based ensemble pruning algorithm and how it compares to other well-established pruning algorithms can be found in [25].

Since the original bagging approach builds ensembles of 25 classifiers, we set the size of the selected ensemble to be 25 as well for a fair comparison. Another reason for this choice is that for each category, there are at least 25 relevant classifiers that are trained on its own data.

Table 2 lists the AUCs of the three methods on each category. The approach involving subset selection is referred to as selective-sharing. Win-loss-tie summarization based on mean value and paired t test (95% significance level) is attached at the bottom of the table. Although the overall performance of the ensembles produced by selective-sharing is a little worse than that of the including-all ensemble, most parts of the improvement gained by sharing classifiers are still kept (Figure 2). Note that the selected ensembles use only 25 classifiers each as compared to 475 of the including-all ensemble. There are two implications of this result. First, the ensemble pruning process is quite effective. Second, the reason that the including-all ensemble is better than those individual bagging ensembles is not simply because it's larger. There is truly useful additional information in the including-all ensemble that can be singled out by the pruning process.

Moreover, the selective-sharing is a conservative version of sharing classifiers, hence it should be more robust. If there were wildly conflicting concepts among the categories, the selective-sharing would have performed better. In fact, the selective-sharing ensembles of Categories 15 and 16 do outperform the including-all ensemble by throwing away bad classifiers, as expected.

6. Knowledge Discovery from the Classifier Sharing Structure

It is also interesting to look at the classifier sharing structure among different categories. Table 3 provides a list of

Table 2. AUCs of non-sharing, sharing-all and selective-sharing ensembles

Cat.	Non-sharing	Sharing-all	Selective-sharing
1	77.19(3.06)	82.00(1.49)	80.77(1.11)
2	76.40(3.42)	81.12(1.08)	76.38(2.91)
3	65.55(5.59)	84.33(1.86)	82.47(3.05)
4	82.08(0.38)	80.98(0.86)	78.56(3.69)
5	72.29(1.71)	82.97(0.58)	78.73(4.50)
6	80.91(0.48)	80.37(0.79)	77.60(3.01)
7	79.78(1.57)	81.17(0.55)	77.63(3.13)
8	76.62(1.28)	78.26(0.40)	76.01(2.18)
9	81.50(0.78)	81.54(0.36)	78.17(2.51)
10	51.38(8.90)	75.41(4.42)	73.80(5.84)
11	69.77(4.24)	80.16(1.18)	78.84(3.09)
12	58.52(7.35)	77.72(1.66)	75.04(3.43)
13	79.21(1.29)	78.52(0.87)	77.32(1.68)
14	83.13(0.41)	81.32(0.68)	80.56(1.55)
15	97.42(0.20)	93.05(1.00)	96.89(0.28)
16	97.56(0.20)	93.52(1.00)	97.75(0.19)
17	82.94(0.78)	84.66(0.28)	82.70(0.89)
18	85.65(1.59)	87.27(0.32)	87.37(1.43)
19	91.22(0.33)	89.66(0.73)	91.27(0.28)
		Comparison	**Selective vs the rest**
	9-10-0	4-15-0	ABS.W-L-T
	3-1-15	3-1-15	SIG.W-L-T

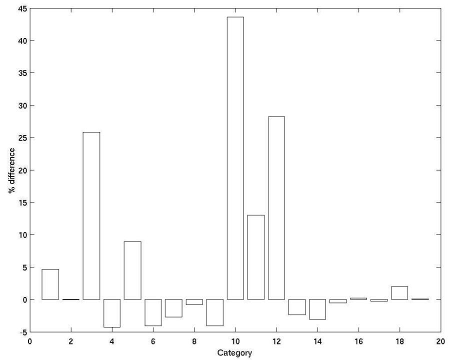

Figure 2. % AUC difference of selective-sharing ensembles and original bagging ensembles

statistics that summarizes the sharing structure, averaged over the same five runs in Section 4. The "Prior" column shows the response rate for each category. The "Used" column is the total number of classifiers trained on this category used for all categories. The "Used Own" column is the total number of classifiers trained on this category used for this category. The "Most Used" column shows the most common training category for the classifiers for each ensemble. Finally, HI index is the Herfindahl index [23], which is computed by $HI_i = \sum_{j=1}^{C} (\frac{n_{ij}}{N_e})^2$, where n_{ij} is the number classifiers trained on jth category used for ith category, and N_e is the size of the pruned ensemble which is 25 in this case. The smaller the HI index, the more diverse the ensemble, in terms of original training categories.

The most surprising fact shown by the table is that for most categories, the classifiers trained on the category are seldom chosen for its own ensemble. Only Categories 14, 17 and 19 used a reasonable number of their own classifiers. However, the performance of the pruned ensembles are close to that of the original bagging ensembles as shown in Table 2. This may imply that some of the categories are closely related therefore their classifiers are highly interchangeable through combination.

The higher the original response rate for each category, the more times its classifiers are used by other categories. Figure 3 plots the number of times its classifiers are used versus the original response rate for each category. The slope of the linear regression is 24.00 with p value 0.053, which verifies the conjecture that the classifiers trained on categories with more positive information may well capture the general properties of the potential customers and are thus widely used by other categories. The fact that the p value is slightly below the 95% significance level implies that the response rate is not the sole factor that decides the popularity of the classifiers.

Categories 15, 16 and 17 are the major outliers from the above rule. Categories 15 and 16 have relatively high response rate, but their classifiers are seldom used, even in their own ensembles. The including-all ensemble performs badly on these two categories. On the other hand, the pruned ensembles perform almost as well as the original bagging ensembles. Our explanation is that these two categories are a little different from the main trend. However, these two categories are somehow closely related to a subset of categories so that a combination of classifiers from those categories may still predict them well. It is unclear why their own classifiers are so rarely used for these categories. One guess is that the original bagging ensemble members for these two categories are not good individually (though they perform well as a group) so the pruning algorithm throws them away. Category 17 is an outlier from the other side. It has relatively low response rate, but its classifiers are the second most used. This indicates that despite the low response rate, the classifiers trained on Category 17 are still good or often provide information that is complementary to that provided from other categories. In addition, there is a subset of categories in which category 17 may be a good representative, including Categories 4 and 5, as shown

Table 3. Classifier Sharing Statistics

Category	Prior (%)	Used	Used Own	Most Used	HI index
1	0.14	2	1	13	0.16
2	0.27	0	0	14	0.27
3	0.04	0	0	6	0.16
4	0.92	12	1	17	0.25
5	0.12	1	0	17	0.27
6	0.83	17	0	14	0.35
7	0.44	9	0	14	0.35
8	0.37	14	1	14	0.67
9	0.64	22	3	14	0.29
10	0.02	1	0	19	0.33
11	0.13	3	0	19	0.57
12	0.05	2	0	19	0.47
13	0.44	18	0	19	0.67
14	2.65	80	7	14	0.30
15	1.70	11	0	19	0.33
16	2.09	14	1	19	0.35
17	0.65	94	9	19	0.34
18	0.31	24	2	19	0.49
19	1.17	150	18	19	0.63

in the “Most Used” column of Table 3.

Classifiers from categories with response rate less than 0.3% are rarely used. Usually, the including-all ensemble performs pretty well on those categories. It shows that for those problem domains without enough information, using aggregated information from all related problem domains may be a good solution.

Classifiers from Categories 14 and 19 occupies almost half of the classifiers that are selected. Classifiers from Category 14 dominate pruned ensembles in six categories while classifiers from Category 19 dominate in nine categories. This might be because the customers of these two categories well-represent the potential buyers of the whole catalog. There exists a subtle difference between these two categories. For Category 14, although its classifiers are widely used, there are only 7 selected for its own problem, while for Category 19, almost all of its own classifiers are selected. There are two explanations of this phenomenon. First, classifiers of Category 14 capture only some of the characteristics of its customers, therefore, it still needs extra information for its own problem. Second, those classifiers are good but very close to each other. So the divergence criteria of the subset selection process prevents including too many of its own classifiers.

The Herfindahl index shows the homogeneity of the sources of the classifiers in the pruned ensembles. If a category prefers universal knowledge for its prediction, the HI index will be low, which means that the pruned ensemble picks classifiers from many categories. On the other hand,

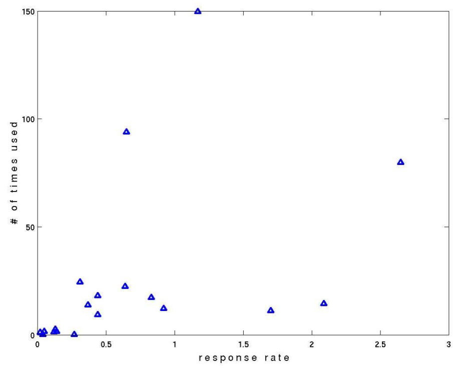

Figure 3. Response rate of each category vs. total number of times its classifiers are used in the selected ensembles

if a category needs specialized knowledge for its prediction, the HI index will be higher, which implies that the pruned ensemble picks classifiers only from a small number of categories that share some particular properties with the category in question. Usually for those categories with small response rate, the HI index is low, for example Category 1 and Category 3. Categories 7 and 13 have the highest HI index. From the “Most Used” column, we know that Category 3 used classifiers mostly from Category 14. So it can be inferred that Category 3 and category 14 are closely related. For the same reason, the customers of Category 13 may share some special properties with that of Category 19.

These pieces of information might make more sense if we knew what these categories were. Unfortunately, it is currently unavailable. The information gained by studying the classifier sharing structure may help improve the mailing and promotion strategy of the catalog company. For instance, customers that buy products from Categories 14, 17 and 19 seem to capture the main trend of the customer base, so they are likely the core customers of the company and may need extra attention. Customers of Category 14 and 3 seem to be similar from some perspective. Therefore, a promotion strategy that proves to be successful for Category 14 may also work well for Category 3.

7. Conclusion and Future Work

The computational results in the previous sections have proved that sharing classifiers among related problem domains is a promising strategy, particularly when there is lack of information in some of the problem domains. A selective-sharing algorithm that prunes the collected ensemble results in more category-specific predictions and may be more robust than the naive sharing-all strategy. Potentially valuable knowledge about the relationships among the problem domains may be obtained by carefully studying the sharing structure of the pruned ensembles.

This new methodology can be applied to quite a few practical problems. For instance, when a company is trying to promote a new product, usually there will be only limited information about its potential customers. However, if the company has good marketing models existing, closey-related products, the old models can be combined (selectively) and applied on the new product. Another potential application is peer-to-peer email filtering. The definitions of a spam email are often different among email users. However, there will be a lot of overlap, particularly for those users with similar interests. By applying the selective sharing algorithm, one can gather useful email filters from other people (without sharing the content of the emails!). A good thing about this combined email filter is that it is possibly able to screen out a spam that has been sent to your peers but never been sent to you before.

This work is the first step towards the research on sharing useful knowledge among related, but different problem domains. The strategy of directly sharing classifiers among domains might be too simple. There are other more flexible ways of sharing knowledge by way of ensemble methods. For example, the output of the models from other problem domains may be treated as input variables for each problem domain instead of bluntly counted as votes. This strategy adopts the idea from the ensemble algorithm called "stacking" [24]. In the future, we will compare these different strategies for knowledge sharing.

Acknowledgement

Samuel Burer would like to acknowledge support from NSF Grant CCR-0203426.

References

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

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

[3] L. Breiman. Arcing classifiers. *Annals of Statistics*, 26:801–849, 1998.

[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] S. Burer and R. Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming (Series B)*, 95:329–357, 2003.

[6] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[7] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Proc. of the 21st International Conference on Machine Learning*, 2004.

[8] P. K. Chan, W. Fan, A. Prodromidis, and S. J. Stolfo. Distributed data mining in credit card fraud detection. *IEEE Intelligent Systems*, November/December:67–74, 1999.

[9] W. Fan, S. J. Stolfo, and J. Zhang. The application of adaboost for distributed, scalable and on-line learning. In *Proc. of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 362–366. ACM Press, 1999.

[10] W. Fan, H. Wang, and P. Yu. Mining extremely skewed trading anomalies. In *Proc. of the 9th International Conference on Extending Database Technology*, pages 801–810, 2004.

[11] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[12] M. Geomans and D. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of ACM*, 42:1115–1145, 1995.

[13] Q. Han, Y. Ye, and J. Zhang. An improved rounding method and semidefinite programming relaxation for graph partition. *Mathematical Programming*, pages 509–535, 2002.

[14] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(10):993–1001, 1990.

[15] 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 Information Processing Systems*, volume 7, pages 231–238. MIT Press, 1995.

[16] A. Lazarevic and Z. Obradovic. The distributed boosting algorithm. In *KDD '01: Proc. of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 311–316. ACM Press, 2001.

[17] D. Margineantu and T. Dietterich. Pruning adaptive boosting. In *14th International Conference on Machine Learning*, pages 211–218, 1997.

[18] A. McCallum. Multi-label text classification with a mixture model trained by EM, 1999. AAAI Workshop on Text Learning.

[19] R. J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, San Manteo, CA, 1993.

[20] A. Sharkey. On combining artificial neural nets. *Connection Science*, 8:299–313, 1996.

[21] W. N. Street and Y. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-01)*, pages 377–382, 2001.

[22] G. Weiss and F. Provost. Learning when training data are costly: The effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, 19:315–354, 2003.

[23] Wikipedia. Herfindahl index, 2004.

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

[25] Y. Zhang, W. N. Street, and S. Burer. Ensemble pruning via semi-definite programming, 2005. Under review.