Ensemble Data Mining Methods

Nikunj C. Oza, Ph.D., NASA Ames Research Center, USA

INTRODUCTION

Ensemble Data Mining Methods, also known as Committee Methods or Model Combiners, are machine learning methods that leverage the power of multiple models to achieve better prediction accuracy than any of the individual models could on their own. The basic goal when designing an ensemble is the same as when establishing a committee of people: each member of the committee should be as competent as possible, but the members should be complementary to one another. If the members are not complementary, i.e., if they always agree, then the committee is unnecessary---any one member is sufficient. If the members are complementary, then when one or a few members make an error, the probability is high that the remaining members can correct this error. Research in ensemble methods has largely revolved around designing ensembles consisting of competent yet complementary models.

BACKGROUND

A supervised machine learner constructs a mapping from input data (normally described by several features) to the appropriate outputs. It does this by learning from a training set---N inputs $x_1, x_2, \ldots, x_N$ for which the corresponding true outputs $y_1, y_2, \ldots, y_N$ are known. The model that results is used to map new inputs to the appropriate outputs. In a classification learning task, each output is one or more classes to which the input belongs. The goal of classification learning is to develop a model that separates the data

into the different classes, with the aim of classifying new examples in the future. For example, a credit card company may develop a model that separates people who defaulted on their credit cards from those who did not based on other known information such as annual income. A model would be generated based on data from past credit card holders. The model would be used to predict whether a new credit card applicant is likely to default on his credit card and thereby decide whether to approve or deny this applicant a new card. In a regression learning task, each output is a continuous value to be predicted (e.g., the average balance that a credit card holder carries over to the next month).

Many traditional machine learning algorithms generate a single model (e.g., a decision tree or neural network). Ensemble learning methods instead generate multiple models. Given a new example, the ensemble passes it to each of its multiple *base* models, obtains their predictions, and then combines them in some appropriate manner (e.g., averaging or voting). As mentioned earlier, it is important to have base models that are competent but also complementary (Tumer and Ghosh, 1996). To further motivate this point, consider Figure 1. This figure depicts a classification problem in which the goal is to separate the points marked with plus signs from points marked with minus signs. None of the three individual linear classifiers (marked A, B, and C) is able to separate the two classes of points. However, a majority vote over all three linear classifiers yields the piecewise-linear classifier shown as a thick line. This classifier is able to separate the two classes perfectly. For example, the plusses at the top of the figure are correctly classified by A and B, but are misclassified by C. The majority vote over these correctly classifies these points as plusses. This happens because A and B are very different from C. If our

ensemble instead consisted of three copies of C, then all three classifiers would misclassify the plusses at the top of the figure, and so would a majority vote over these classifiers.

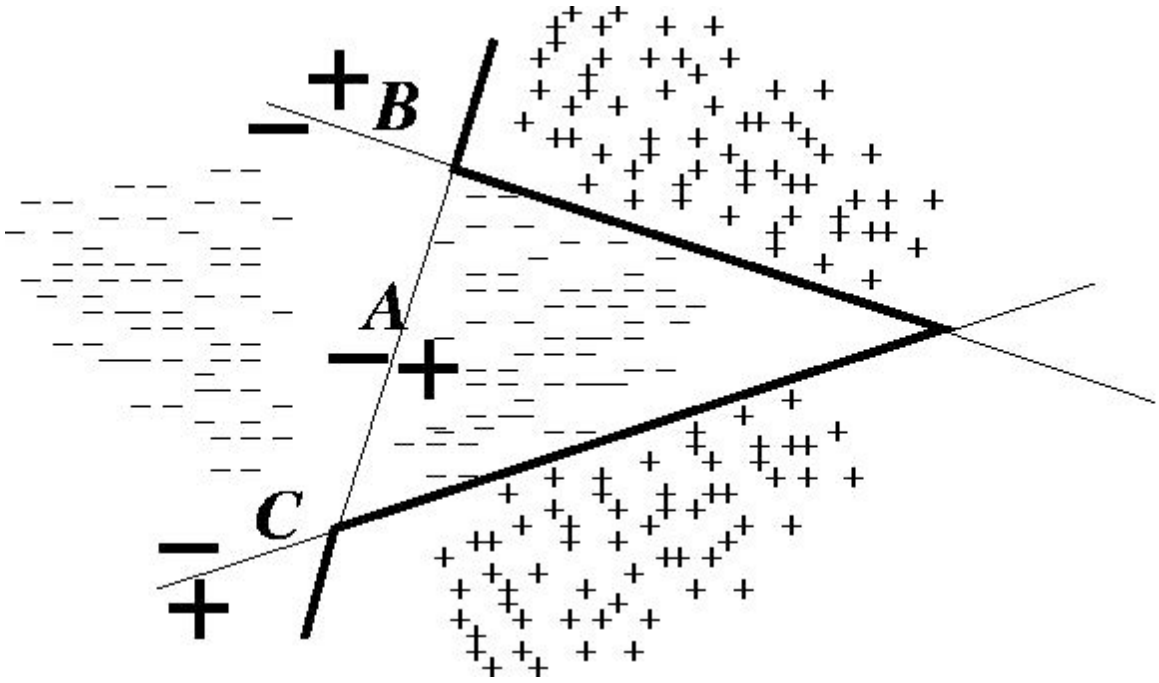

Figure 1: An ensemble of linear classifiers. Each line---A, B, and C---is a linear classifier. The boldface line is the ensemble that classifies new examples by returning the majority vote of A, B, and C.

MAIN THRUST OF THE CHAPTER

We now discuss the key elements of an ensemble learning method and ensemble model and, in the process, discuss several ensemble methods that have been developed.

Ensemble Methods

The example shown in figure 1 is an artificial example. We cannot normally expect to obtain base models that misclassify examples in completely separate parts of the input space and ensembles that classify all the examples correctly. However, there are many algorithms that attempt to generate a set of base models that make errors that are as different from one another as possible. Methods such as Bagging (Breiman, 1994) and Boosting (Freund and Schapire, 1996) promote diversity by presenting each base model with a different subset of training examples or different weight distributions over the examples. For example, in figure 1, if the plusses in the top part of the figure were temporarily removed from the training set, then a linear classifier learning algorithm trained on the remaining examples would probably yield a classifier similar to C. On the other hand, removing the plusses in the bottom part of the figure would probably yield classifier B or something similar. In this way, running the same learning algorithm on different subsets of training examples can yield very different classifiers which can be combined to yield an effective ensemble. Input Decimation Ensembles (IDE) (Tumer and Oza, 2003) and Stochastic Attribute Selection Committees (SASC) (Zheng and Webb, 1998) instead promote diversity by training each base model with the same training examples but different subsets of the input features. SASC trains each base model with a random subset of input features. IDE selects, for each class, a subset of features that has the highest correlation with the presence or absence of that class. Each feature subset is used to train one base model. However, in both SASC and IDE, all the training patterns are used with equal weight to train all the base models.

So far we have distinguished ensemble methods by the way they train their base models. We can also distinguish methods by the way they combine their base models' predictions. Majority or plurality voting is frequently used for classification problems and is used in Bagging. If the classifiers provide probability values, simple averaging is commonly used and is very effective (Tumer and Ghosh, 1996). Weighted averaging has also been used and different methods for weighting the base models have been examined. Two particularly interesting methods for weighted averaging include Mixtures of Experts (Jordan and Jacobs, 1994) and Merz's use of Principal Components Analysis (PCA) to combine models (Merz, 1999). In Mixtures of Experts, the weights in the weighted average combining are determined by a gating network, which is a model that takes the same inputs that the base models take, and returns a weight for each of the base models. The higher the weight for a base model, the more that base model is trusted to provide the correct answer. These weights are determined during training by how well the base models perform on the training examples. The gating network essentially keeps track of how well each base model performs in each part of the input space. The hope is that each model learns to specialize in different input regimes and is weighted highly when the input falls into its specialty. Intuitively, we regularly use this notion of giving higher weights to opinions of experts with the most appropriate specialty. Merz's method uses PCA to lower the weights of base models that perform well overall but are redundant and therefore effectively give too much weight to one model. For example, in Figure 1, if there were instead two copies of A and one copy of B in an ensemble of three models, we may prefer to lower the weights of the two copies of A since, essentially, A is being given too much weight. Here, the two copies of A would always outvote B, thereby

rendering B useless. Merz's method also increases the weight on base models that do not perform as well overall but perform well in parts of the input space where the other models perform poorly. In this way, a base model's unique contributions are rewarded. Many of the methods described above have been shown to be specific cases of one method: Importance Sampled Learning Ensembles (Friedman and Popescu, 2003).

When designing an ensemble learning method, in addition to choosing the method by which to bring about diversity in the base models and choosing the combining method, one has to choose the type of base model and base model learning algorithm to use. The combining method may restrict the types of base models that can be used. For example, to use average combining in a classification problem, one must have base models that can yield probability estimates. This precludes the use of linear discriminant analysis which is not set up to return probabilities. The vast majority of ensemble methods use only one base model learning algorithm but use the methods described earlier to bring about diversity in the base models. There has been surprisingly little work (e.g., (Merz 1999)) on creating ensembles with many different types of base models.

Two of the most popular ensemble learning algorithms are Bagging and Boosting, which we briefly explain next.

Bagging

*B*ootstrap *Agg*regat*ing* (Bagging) generates multiple bootstrap training sets from the original training set (using sampling with replacement) and uses each of them to generate a classifier for inclusion in the ensemble. The algorithms for bagging and sampling with replacement are given in figure 2 below. In these algorithms, $\{(x_1,y_1)$,

(x_2,y_2), …, (x_N,y_N)}} is the training set., M is the number of base models to be learned, L_b is the base model learning algorithm, the h_i's are the base models, *random_integer(a,b)* is a function that returns each of the integers from a to b with equal probability, and *I(X)* is the indicator function that returns 1 if X is true and 0 otherwise.

Bagging($\{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\},M$)
For each $m = 1,2,\ldots,M$
  T_m = Sample_With_Replacement($\{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\},N$)
  $h_m = L_b(T_m)$
Return $h_{fin}(x) = \arg\max_{y\in Y} \sum_{m=1}^{M} I(h_m(x) = y)$.

Sample_With_Replacement(T,N)
$S = \{\}$
For $i = 1,2,\ldots,N$
  r = random_integer(1,N)
  Add $T[r]$ to S.
Return S.

Figure 2: Batch Bagging Algorithm and Sampling with Replacement

To create a bootstrap training set from an original training set of size N, we perform N Multinomial trials, where in each trial, we draw one of the N examples. Each example has probability $1/N$ of being drawn in each trial. The second algorithm shown in figure 2 does exactly this---N times, the algorithm chooses a number r from 1 to N and adds the rth training example to the bootstrap training set S. Clearly, some of the original training examples will not be selected for inclusion in the bootstrap training set and

others will be chosen one time or more. In bagging, we create M such bootstrap training sets and then generate classifiers using each of them. Bagging returns a function $h(x)$ that classifies new examples by returning the class y that gets the maximum number of votes from the base models $h_1,h_2,\ldots,h_M$. In bagging, the M bootstrap training sets that are created are likely to have some differences. (Breiman, 1994) demonstrates that bagged ensembles tend to improve upon their base models more if the base model learning algorithms are *unstable*---differences in their training sets tend to induce significant differences in the models. He notes that decision trees are unstable, which explains why bagged decision trees often outperform individual decision trees; however, decision stumps (decision trees with only one variable test) are stable, which explains why bagging with decision stumps tends not to improve upon individual decision stumps.

AdaBoost$(\{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\},L_b,M)$

Initialize $D_1(n) = 1/N$ for all $n \in \{1,2,\ldots,N\}$.

For each $m = 1,2,\ldots,M$:

$h_m = L_b(\{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\},D_m)$.

$\varepsilon_m = \sum_{n:h_m(x_n)\neq y_n} D_m(n)$.

If $\varepsilon_m \geq 1/2$ then,

set $M = m-1$ and abort this loop.

Update distribution D_m:

$$D_{m+1}(n) = D_m(n) \times \begin{cases} \dfrac{1}{2(1-\varepsilon_m)} & \text{if } h_m(x_n) = y_n \\ \dfrac{1}{2\varepsilon_m} & \text{otherwise.} \end{cases}$$

Return $h_{fin}(x) = \arg\max_{y\in Y} \sum_{m=1}^{M} I(h_m(x) = y) \log\left(\dfrac{1-\varepsilon_m}{\varepsilon_m}\right)$.

Figure 3: AdaBoost Algorithm

Boosting

Boosting algorithms are a class of algorithms that have been mathematically proven to improve upon the performance of their base models in certain situations. We now explain the AdaBoost algorithm because it is the most frequently used among all boosting algorithms. AdaBoost generates a sequence of base models with different weight distributions over the training set. The AdaBoost algorithm is shown in Figure 3. Its inputs are a set of N training examples, a base model learning algorithm L_b, and the number M of base models that we wish to combine. AdaBoost was originally designed for two-class classification problems; therefore, for this explanation we will assume that there are two possible classes. However, AdaBoost is regularly used with a larger number of classes. The first step in AdaBoost is to construct an initial distribution of weights D_1 over the training set. This distribution assigns equal weight to all N training examples. We now enter the loop in the algorithm. To construct the first base model, we call L_b with distribution D_1 over the training set. After getting back a model h_1, we calculate its error ε_1 on the training set itself, which is just the sum of the weights of the training examples that h_1 misclassifies. We require that $\varepsilon_1 < 1/2$ (this is the *weak learning* assumption---the error should be less than what we would achieve through randomly guessing the class[i]). If this condition is not satisfied, then we stop and return the ensemble consisting of the previously-generated base models. If this condition is satisfied, then we calculate a new distribution D_2 over the training examples as follows. Examples that were correctly classified by h_1 have their weights multiplied by $1/(2(1-\varepsilon_1))$. Examples that were misclassified by h_1 have their weights multiplied by $1/(2\varepsilon_1)$. Note that, because of our condition $\varepsilon_1 < 1/2$, correctly classified examples have their weights reduced and

misclassified examples have their weights increased. Specifically, examples that h_1 misclassified have their total weight increased to *1/2* under D_2 and examples that h_1 correctly classified have their total weight reduced to *1/2* under D_2. We then go into the next iteration of the loop to construct base model h_2 using the training set and the new distribution D_2. The point is that the next base model will be generated by a weak learner (i.e., the base model will have error less than *1/2*); therefore, at least some of the examples misclassified by the previous base model will have to be correctly classified by the current base model. In this way, boosting forces subsequent base models to correct the mistakes made by earlier models. We construct M base models in this fashion. The ensemble returned by AdaBoost is a function that takes a new example as input and returns the class that gets the maximum weighted vote over the M base models, where each base model's weight is $log((1-\varepsilon_m)/\varepsilon_m)$, which is proportional to the base model's accuracy on the weighted training set presented to it.

AdaBoost has performed very well in practice and is one of the few theoretically-motivated algorithms that has turned into a practical algorithm. However, AdaBoost can perform poorly when the training data is noisy (Dietterich, 2000), i.e., the inputs or outputs have been randomly contaminated. Noisy examples are normally difficult to learn. Because of this, the weights assigned to noisy examples often become much higher than for the other examples, often causing boosting to focus too much on those noisy examples at the expense of the remaining data. Some work has been done to mitigate the effect of noisy examples on boosting (Oza 2004, Ratsch, et. al., 2001).

FUTURE TRENDS

The fields of machine learning and data mining are increasingly moving away from working on small datasets in the form of flat files that are presumed to describe a single process. The fields are changing their focus toward the types of data increasingly being encountered today: very large datasets, possibly distributed among different locations, describing operations with multiple modes, time-series data, online applications (the data is not a time series but nevertheless arrives continually and must be processed as it arrives), partially-labeled data, and documents. Research in ensemble methods is beginning to explore these new types of data. For example, ensemble learning traditionally has required access to the entire dataset at once, i.e., it performs batch learning. However, this is clearly impractical for very large datasets that cannot be loaded into memory all at once. Some recent work (Oza and Russell, 2001; Oza, 2001) applies ensemble learning to such large datasets. In particular, this work develops online versions of bagging and boosting. That is, whereas standard bagging and boosting require at least one scan of the dataset for every base model created, online bagging and online boosting require only one scan of the dataset regardless of the number of base models. Additionally, as new data arrives, the ensembles can be updated without reviewing any past data. However, because of their limited access to the data, these online algorithms do not perform as well as their standard counterparts. Other work has also been done to apply ensemble methods to other types of problems such as remote sensing (Rajan and Ghosh, 2005), person recognition (Chawla and Bowyer, 2005), one vs. all recognition (Cabrera, et. al., 2005), and medicine (Pranckeviciene, et. al., 2005)---a recent survey of such applications is (Oza and Tumer, 2008). However, most of this work is experimental.

Theoretical frameworks that can guide us in the development of new ensemble learning algorithms specifically for modern datasets have yet to be developed.

CONCLUSIONS

Ensemble Methods began about fifteen years ago as a separate research area within machine learning and were motivated by the idea of wanting to leverage the power of multiple models and not just trust one model built on a small training set. Significant theoretical and experimental developments have occurred over the past fifteen years and have led to several methods, especially bagging and boosting, being used to solve many real problems. However, ensemble methods also appear to be applicable to current and upcoming problems of distributed data mining, online applications, and others. Therefore, practitioners in data mining should stay tuned for further developments in the vibrant area of ensemble methods. An excellent way to do this is to read a recent textbook on ensembles (Kuncheva, 2004) and follow the series of workshops called the International Workshop on Multiple Classifier Systems (proceedings published by Springer). This series' balance between theory, algorithms, and applications of ensemble methods gives a comprehensive idea of the work being done in the field.

REFERENCES

Breiman, L. (1994). Bagging Predictors, Technical Report 421, Department of Statistics, University of California, Berkeley.

Cabrera, J.B.D., Gutierrez, C., and Mehra, R.K. (2005). Infrastructures and Algorithms for Distributed Anomaly-based Intrusion Detection in Mobile Ad-hoc Networks. In

Proceedings of the IEEE Conference on Military Communications, pp. 1831-1837. IEEE, Atlantic City, New Jersey, USA.

Chawla, N. and Bowyer, K. (2005). Designing Multiple Classifier Systems for Face Recognition. In N. Oza, R. Polikar, J. Kittler, and F. Roli, editors, *Proceedings of the Sixth International Workshop on Multiple Classifier Systems*, pp. 407-416. Springer-Verlag, Berlin.

Dietterich, T. (2000). An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization, *Machine Learning 40,* 139-158.

Freund, Y. & Schapire, R. (1996). Experiments with a new Boosting Algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 148-156. Morgan Kaufmann.

Friedman, J.H. & Popescu, B.E. (2003). Importance Sampled Learning Ensembles. Technical Report, Stanford University.

Jordan, M.I. & Jacobs, R.A. (1994). Hierarchical Mixture of Experts and the EM Algorithm. *Neural Computation, 6,* 181-214.

Kuncheva, L.I. (2004). *Combining Pattern Classifiers: Methods and Algorithms*, Wiley-Interscience.

Merz, C.J. (1999). A Principal Component Approach to Combining Regression Estimates. *Machine Learning, 36*, 9-32.

Oza, N.C. (2001). Online Ensemble Learning, Ph.D. thesis, University of California, Berkeley.

Oza, N.C. (2004). AveBoost2: Boosting with Noisy Data. In F. Roli , J. Kittler, and T.

Windeatt (Eds.), *Proceedings of the Fifth International Workshop on Multiple Classifier Systems*, pp. 31-40, Springer-Verlag, Berlin.

Oza, N.C. & Russell, S. (2001). Experimental Comparisons of Online and Batch Versions of Bagging and Boosting, *The Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, California.

Oza, N.C. & Tumer, K. (2008). Classifier Ensembles: Select Real-World Applications. *Information Fusion, Special Issue on Applications of Ensemble Methods*, 9(1), 4-20.

Prackeviciene, E., Baumgartner, R., and Somorjai, R. (2005). Using Domain Knowledge in the Random Subspace Method: Application to the Classification of Biomedical Spectra. In N. C. Oza, R. Polikar, J, Kittler, and F, Roli, editors, *Proceedings of the Sixth International Workshop on Multiple Classifier Systems*, pp. 336-345. Springer-Verlag, Berlin.

Rajan, S. and Ghosh, J. (2005). Exploiting Class Hierarchies for Knowledge Transfer in Hyperspectral Data. In N.C. Oza, R. Polikar, J. Kittler, and F. Roli, editors, *Proceedings of the Sixth International Workshop on Multiple Classifier Systems*, pp. 417-428. Springer-Verlag, Berlin.

Ratsch, G., Onoda, T., & Muller, K.R. (2001). Soft Margins for AdaBoost. *Machine Learning, 42,* 287-320.

Tumer, K. & Ghosh, J. (1996). Error Correlation and Error Reduction in Ensemble Classifiers. *Connection Science, Special Issue on Combining Artificial Neural Networks: Ensemble Approaches, 8(3 & 4),* 385-404.

Tumer, K. & Oza, N.C. (2003). Input Decimated Ensembles, *Pattern Analysis and Applications*, 6(1):65-77.

Zheng, Z. & Webb, G. (1998). Stochastic Attribute Selection Committees. In *Proceedings of the 11th Australian Joint Conference on Artificial Intelligence (AI'98),* pp. 321-332.

TERMS AND THEIR DEFINITIONS:

Batch Learning: Learning using an algorithm that views the entire dataset at once and can access any part of the dataset at any time and as many times as desired.

Ensemble: A function that returns a combination of the predictions of multiple machine learning models.

Decision Tree: A model consisting of nodes that contain tests on a single attribute and branches representing the different outcomes of the test. A prediction is generated for a new example by performing the test described at the root node and then proceeding along the branch that corresponds to the outcome of the test. If the branch ends in a prediction, then that prediction is returned. If the branch ends in a node, then the test at that node is performed and the appropriate branch selected. This continues until a prediction is found and returned.

Machine Learning: The branch of Artificial Intelligence devoted to enabling computers to learn.

Neural Network: A nonlinear model derived through analogy with the human brain. It consists of a collection of elements that linearly combine their inputs and pass the result through a nonlinear transfer function.

Online Learning: Learning using an algorithm that only examines the dataset once in order. This paradigm is often used in situations when data arrives continually in a stream and predictions must be obtainable at any time.

Principal Components Analysis (PCA): Given a dataset, PCA determines the axes of maximum variance. For example, if the dataset was shaped like an egg, then the long axis of the egg would be the first principal component because the variance is greatest in this direction. All subsequent principal components are found to be orthogonal to all previous components.

---

[i] This requirement is perhaps too strict when there are more than two classes. There is a multi-class version of AdaBoost (Freund and Schapire, 1997) that does not have this requirement. However, the AdaBoost algorithm presented here is often used even when there are more than two classes if the base model learning algorithm is strong enough to satisfy the requirement.