CHAPTER

10

Model Ensembles

Model Ensembles, or simply "ensembles," are combinations of two or more predictions from predictive models into a single composite score. The chapters on building supervised and unsupervised learning models focused on how to build the best single model and how to determine which of the single models you should select for deployment. The ensemble approach turns this thinking around. Rather than building models and selecting the single best model to deploy, why not build many models and use them all in deployment?

This chapter describes how to build model ensembles and why understanding how to build them is important—some would say essential—for predictive modelers.

Motivation for Ensembles

Practitioners build model ensembles for one reason: improved accuracy. In study after study over the past two decades, ensembles nearly always improve model predictive accuracy and rarely predict worse than single models. In the 1990s, when ensembles were beginning to appear in the data-mining literature,

the improved accuracy of ensembles on held-out data began to appear in an almost magical way. By the 2000s, ensembles became essential to winning data mining competitions.

Consider two examples, and these are not unique. First, the Pacific-Asia Conference on Knowledge Discovery (PAKDD) has a data mining competition each year called the PAKDD Cup. In 2007, twenty-eight submissions were received and the top five solutions were model ensembles, including three submissions with ensembles of trees, one with an ensemble of Probit models (similar to logistic regression models), and one neural network ensemble.

A second recent example of ensembles winning competitions is the Netflix prize, an open competition that solicited entries to predict user ratings of films based on historical ratings. The prize was $1,000,000 for any team that could reduce the root means squared error (RMSE) of the existing Netflix internal algorithm by 10 percent or more. The winner, runner-up, and nearly all the leaders used model ensembles in their submissions. In fact, the winning submission was the result of an ensemble containing hundreds of predictive models.

Model ensembles not only can improve model accuracy, but they can also improve model robustness. Through averaging multiple models into a single prediction, no single model dominates the final predicted value of the models, reducing the likelihood that a flaky prediction will be made (so long as all the models don't agree on the flaky prediction).

The improved accuracy comes with costs, however. First, model interpretation, if it ever existed, is lost. A single decision tree is a favorite choice of predictive modelers when model interpretation is paramount. However, if an ensemble of 1,000 trees is the final model, the determination of why a prediction was made requires assessing the predictions of 1,000 models. Techniques exist for aiding in interpreting the ensemble but the transparency of a single decision tree is gone.

Second, model ensembles can become too computationally expensive for deployment. In the Netflix prize, the winning submission was never used directly to make movie recommendations because of its complexity and inability to scale and make the movie predictions quickly enough. Sometimes accuracy itself isn't enough for a model or ensemble of models to be useful.

The Wisdom of Crowds

In the popular book *The Wisdom of Crowds* (Random House, 2005), author James Surowiecki proposes that better decisions can be made if rather than relying on a single expert, many (even uninformed) opinions can be aggregated into a decision that is superior to the expert's opinion, often called *crowdsourcing*.

Surowiecki describes four characteristics necessary for the group opinion to work well and not degenerate into the opposite effect of poor decisions as

evidenced by the "madness of crowds": diversity of opinion, independence, decentralization, and aggregation (see Figure 10-1). The first three characteristics relate to how the individual decisions are made: They must have information different than others in the group and not be affected by the others in the group. The last characteristic merely states that the decisions must be combined.

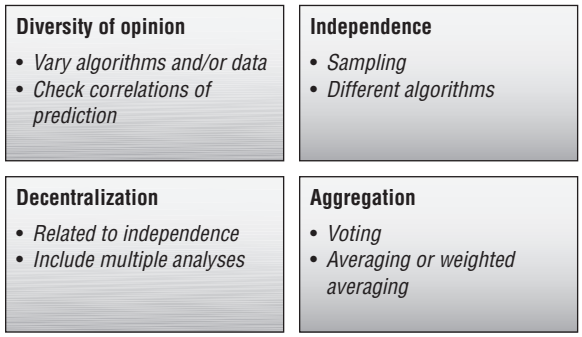

Figure 10-1: Characteristics of good ensemble decisions

Model ensembles have very similar characteristics. Each predictive model has a voice in the final decision. The diversity of opinion can be measured by the correlation of the predictive values themselves: If all of the predictions are highly correlated with one another, or in other words, if the models nearly all agree, there is no point in combining them. The decentralization characteristic can be achieved by resampling data or through case weights: Each model uses either different records from a common data set or at least uses the records with weights that are different from the other models.

Bias Variance Tradeoff

Before describing model ensembles, it is useful to understand the principle in statistical literature of the *bias-variance tradeoff*. Bias refers to model error and variance refers to the consistency in predictive accuracy of models applied to other data sets. The best models have low bias (low error, high accuracy) *and* low variance (consistency of accuracy from data set to data set).

Unfortunately, there is always a tradeoff between these two building predictive models. You can achieve low bias on training data, but may suffer from high variance on held-out data because the models were overfit. The k-NN algorithm with k=1 is an example of a low bias model (perfect on training data), but susceptible to high variance on held-out data. Conversely, you can achieve low variance from data set to data set but at the cost of higher bias. These are typically simple models that lack the flexibility and power to predict accurately, such as single split trees called decision stumps.

In a simple example, Figure 10-2 shows a data set with a simple linear regression model.

This model will have relatively low variance as it is a smooth predictor, although it has bias. A second curve fit is shown in Figure 10-3, this time with low bias, but because of the complexity of the model it is in danger of overfitting the data and will likely have large errors on subsequent data. These larger errors are shown in Figure 10-4. Four vertical, dashed lines show the errors of these four data points, clearly large errors induced by the differences between the diamond data points the model was built from and the second box data. Every new data set, with comparable variations in the data, will also have large errors like those shown in Figure 10-4, resulting in high error variance from this model.

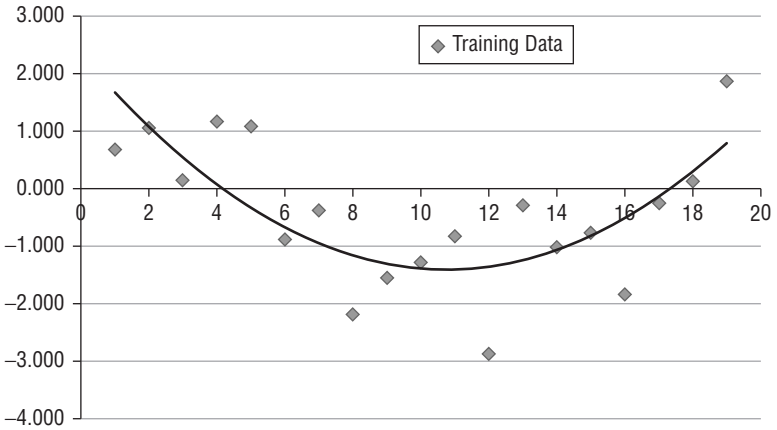

Figure 10-2: Low variance fit to data

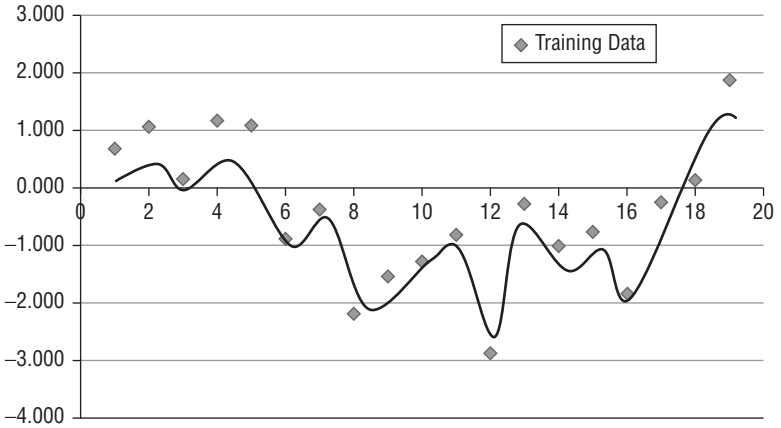

Figure 10-3: Low bias fit to data

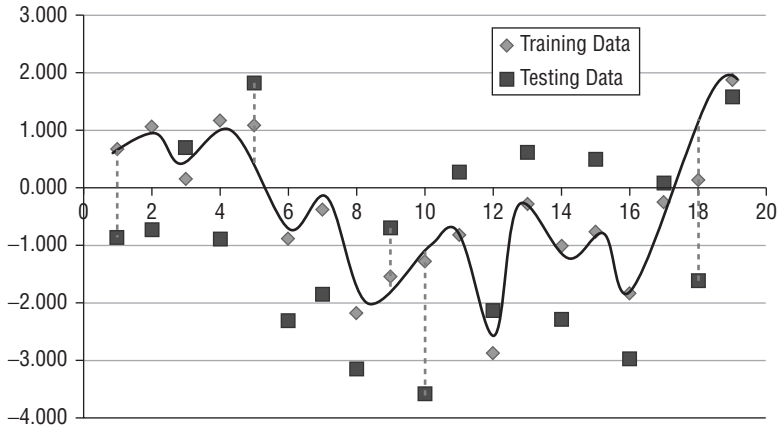

Figure 10-4: Errors on new data

Bagging

Leo Brieman first published a description of the Bootstrap Aggregating (bagging) algorithm in 1996. The idea is quite simple yet powerful: Build multiple decision trees from resampled data and combine the predicted values through averaging or voting. The resampling method Breiman used was bootstrap sampling (sampling with replacement), which creates replicates of some records in the training data, although on average, 37 percent of the records will not be included at all in the training data.

Although bagging was first developed for decision trees, the idea can be applied to any algorithm that produces predictions with sufficient variation in the predicted values. Other algorithms that are good candidates include neural networks, Naïve Bayes, k-nearest neighbor (for low values of k), and to a lesser degree, even logistic regression. k-nearest neighbor is not a good candidate for bagging if the value of k is already large; the algorithm already votes or averages predictions and with larger values of k, predictions are already very stable with low variance.

How many bootstrap samples, sometimes called replicates, should be created? Brieman stated, "My sense of it is that fewer are required when y is numerical and more are required with an increasing number of classes." He typically used 10–25 bootstrap replicates, with significant improvements occurring with as few as 10 replicates.

Overfitting the models is an important requirement to building good bagged ensembles. By overfitting each model, the bias is low, but the decision tree will

generally have worse accuracy on held-out data. But bagging is a variance reduction technique; the averaging of predictions smoothes the predictions to behave in a more stable way on new data.

For example, consider models based on the KDD Cup 1998 data set. Thirty (30) bootstrap samples were created from a stratified sample of the target variable. From these 30 samples, 30 decision trees were built. Only LASTGIFT and RFA_2F were included as inputs to simplify visualization of results. Each model was a CART-styled decision tree that was somewhat overfit, per Breiman's recommendation.

Results shown are based on test data only, not training data or out-of-sample data from the bootstrap samples. Table 10-1 shows the average model Area under the Curve (AUC) for these 30 models. The 30 models were remarkably consistent in their AUC metrics; the AUC values ranged from 0.530 to 0.555. If the models built to be included in the ensemble are poor predictors, or in other words have high bias, it is better to exclude those models from the ensemble as they will degrade the accuracy of the ensemble.

Bagged ensembles were created from averaging the model predicted probabilities rather than voting. A 10-model ensemble is shown in the table as well: Its AUC is better than even the best individual tree. Finally, a 30-model ensemble was created and had the best AUC on test data. Bagged ensembles with more than 30 trees were not any better than using 30 trees. While the gains are modest on this data set, they are statistically significant. These results are summarized in Table 10-1.

Table 10-1: Bagging Model AUC

METHOD	AUC
Average tree	0.542
Best tree	0.555
10-tree ensemble	0.561
30-tree bagging ensemble	0.566

The diversity of model predictions is a key element in creating effective ensembles. One way to measure the diversity of predictions is to examine the correlation of predicted values. If the correlations between model predictions are always very high, greater than 0.95, there is little additional predictive information each model brings to the ensemble and therefore little improvement in accuracy is achievable. Generally, it is best to have correlations less than 0.9 at most. The correlations should be computed from the model propensities or predicted probabilities rather than the {0,1} classification value itself.

In the modeling example, the 30 models produce pairwise correlations that range from 0.212 to 0.431: all quite low. When estimates produce similar performance scores (such as the AUC here) but are not correlated with each other, they achieve their accuracy by predicting accurately on different records, the ideal scenario for effective ensembles. Records that are easy to classify have no need for ensembles; any single model will classify these records correctly. Likewise, accuracy on records that are never classified correctly in any model cannot be improved; these records are lost causes. It is the disagreement between models that provides the potential. The hope is that the majority of models in the ensemble will predict these correctly and thus "outvote" the poor decisions.

A final way to examine how the Bagging Ensemble turns individual models into a composite prediction can be seen through the decision boundaries created by the models. Figure 10-5 shows decision regions for nine of the models. The x-axis in each plot is LASTGIFT; the y-axis is RFA_2F. RFA_2F values (1, 2, 3, or 4) have been randomly jittered so they can be seen more clearly. A predicted class "1" is shown as a light gray square and a predicted class "0" by a dark gray square. It is easy to see from these plots why the correlations between models is so low: each model creates different decision regions.

After averaging 30 models, the resulting decision regions that produced the increase in AUC are shown in Figure 10-6. The regions are generally more homogeneous than in single models.

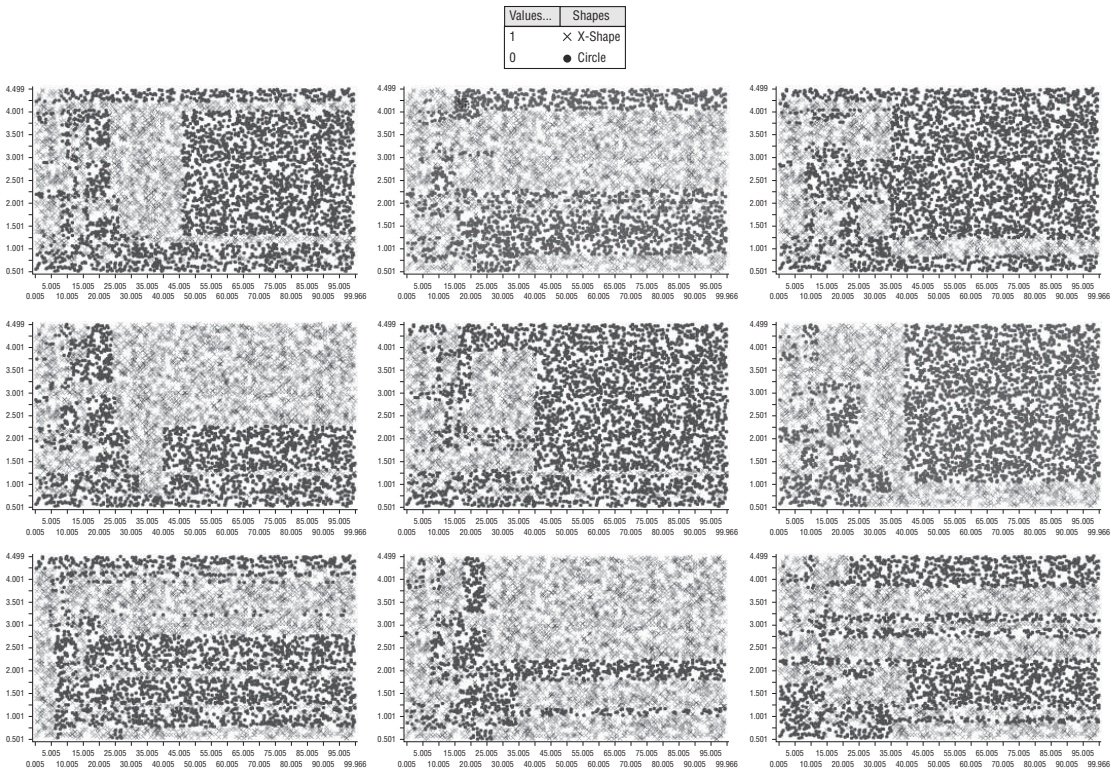

Figure 10-5: Decision regions for nine bagged trees

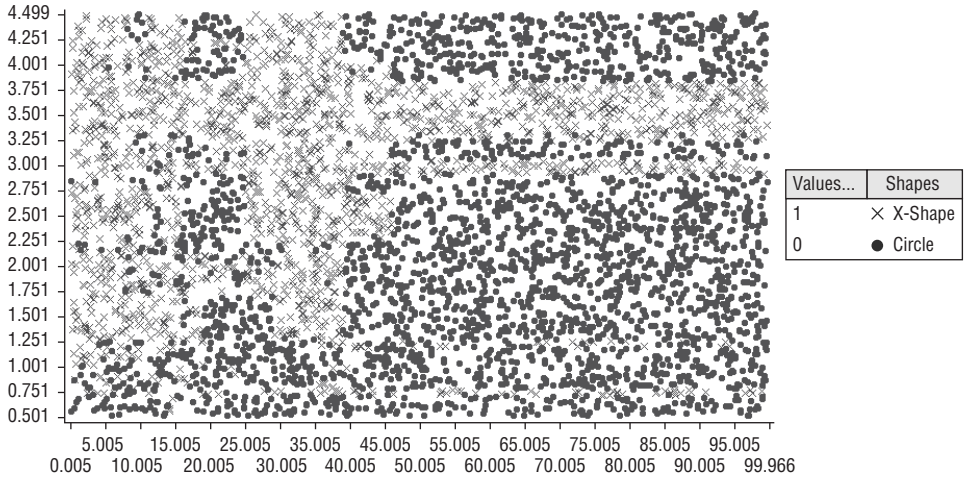

Figure 10-6: Decision region for bagging ensemble

Contrast these figures to a heat map of the actual response rates plotted on the same axes as shown in Figure 10-7. Since the target variable, TARGET_B, only has values 0 and 1, continuous values comparable to the model probabilities are created by binning the two variables (LASTGIFT and RFA_2F) and computing the average response rate in each bin. Each square was colored using colors comparable to the points in the scatterplots shown in Figures 10-5 and 10-6. The bagging ensemble clearly has identified the trends in the data (Figure 10-6), especially the light gray region at the upper left and the dark gray region at the bottom right. The vertical stripe in Figure 10-6 can be seen in the vertical band for LASTGIFT having values between 7 and 10 dollars.

Bootstrap sampling in bagging is the key to introducing diversity in the models. One can think of the bootstrap sampling methodology as creating case weights for each record: Some records are included multiple times in the training data—their weights are 1, 2, 3, or more—and others are not included at all—their weights are equal to 0.

Some software implementations of decision trees will include an option for bagging or provide the capability to construct bagged models. Even if this is not the case, however, bagging is straightforward to implement in most predictive analytics software packages. For example, if a modeler is building logistic regression models, creating bagged ensembles of logistic regression models requires only the creation of 10 samples, 10 models, and logic to combine the predictions. The same is true for creating bagged ensembles using any algorithm.

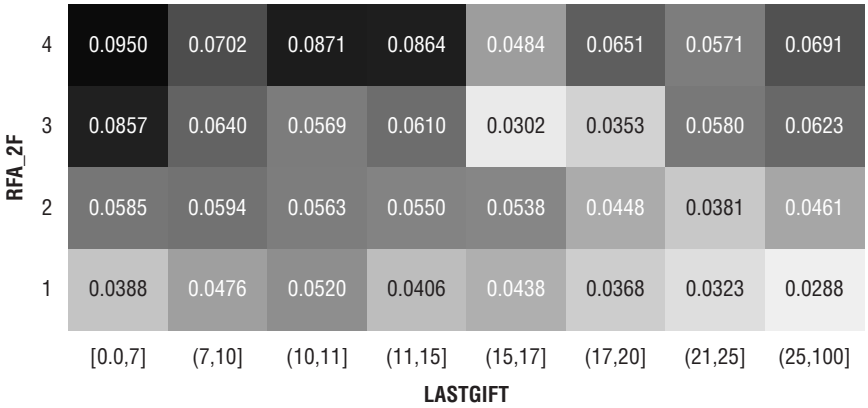

Figure 10-7: Decision region for actual target variable

Other techniques also exist for varying case weights rather than using a bootstrap sample. One could select randomly which records to include in training and which to exclude, which assigns case weights of 1 and 0 to each record respectively. Cross-validation sampling is one way to accomplish this weighting. One could even create a random, real-valued case weight for each record. These are not as commonly found in commercial software but are easy to implement through custom manipulation of the data.

Boosting

Robert Shapire and Yoav Freund introduced the boosting algorithm in separate publications in 1990. In a 1995 joint publication they first described the AdaBoost algorithm (pronounced "add-a-boost"). The idea, like bagging, is also straightforward.

First, one builds a simple classification model; the model only needs to be slightly better than random chance, so for a binary classification problem, only slightly better than a 50 percent correct classification. In this first pass, each record is used in the algorithm with equal case weights, as one would do normally in building a predictive model. The errors in the predicted values are noted. Records correctly classified have their case weights reduced and records incorrectly classified have their case weights increased, and a second simple model is built. In other words, for the second model, records that were incorrectly classified are encouraged through case weights to be considered more strongly in the construction of the model. The records that are difficult to classify, meaning that initially, the models classify them incorrectly, keep getting case

weights increased more and more, communicating to the algorithm to pay more attention to these records until, hopefully, they are finally classified correctly.

Boosting is often repeated tens or even hundreds of times. After the tens or hundreds of iterations, the final predictions are made based on a weighted average of the predictions from all the models.

Consider the simple example in Figure 10-8 based on only 10 data points. The data contains two class values represented by a dark gray circle and a light gray square. The size of the symbol represents the case weight for that data point, with the initial size representing a weight of 1. In the first decision stump, cases to the left of the split, represented by the vertical dotted line, are predicted to be light gray, and those to the right are predicted to be dark gray. Therefore, there are three misclassified data points (all light gray), and the remaining data points, including all the dark gray points, are classified correctly. These misclassified data points are given increased weight and the correctly classified data points decreased weight, as shown in the upper right of the figure and labeled "Reweighting." The amount of the reweighting shown in Figure 10-8 only represents the increase or decrease and should not be interpreted as showing a precise reweighting.

The second decision stump is then computed based on the reweighted data, and the process of reweighting the data points is repeated, resulting in the plot to the right of the second stump split. After the third reweighting, you can see that the light gray data point at the upper right has now been misclassified three times and therefore has the largest weight of any data point. Two data points have been classified correctly three times, and their weights are the smallest; they are at the top left (light gray) and bottom right dark gray.

If you examine all three splits from the three decision stumps, you can see in Figure 10-9 that there are six decision regions defined by the three splits. Each region will have a different model score based on the weighted sum of the three models, with the region at the upper left having a probability of being light gray equal to 1 and the region to the lower right having a probability of being light gray equal to 0. If additional iterations of the boosting algorithm were to take place, the high weight of the light gray square at the upper right would undoubtedly result in a decision stump that separates this data point from the others in the data to classify it correctly.

As a reminder, this process is done automatically in software implementations of boosting and does not have to be done by the modeler in a step-by-step manner. For example, the C5 classification tree algorithm has a boosting algorithm built in, using a variant of the original AdaBoost algorithm.

Boosting methods are designed to work with weak learners, that is, simple models; the component models in a boosted ensemble are simple models with high bias, though low variance. The improvement with boosting is better, as

with Bagged models, when algorithms that are unstable predictors are used. Decision trees are most often used in boosted models. Naïve Bayes is also used but with fewer improvements over a single model.

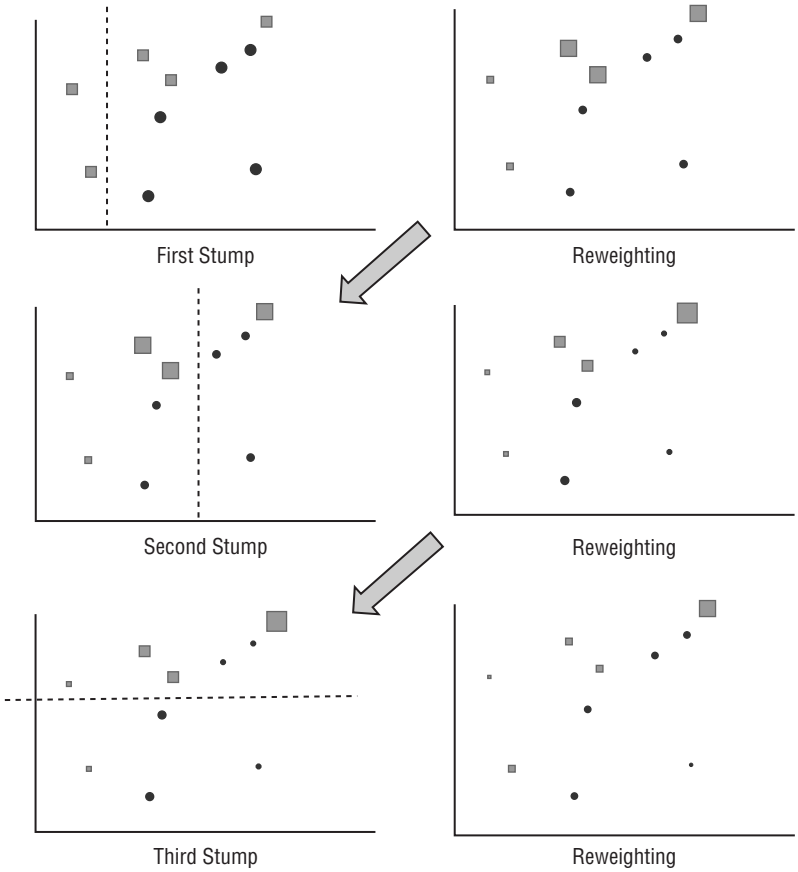

Figure 10-8: AdaBoost reweighting

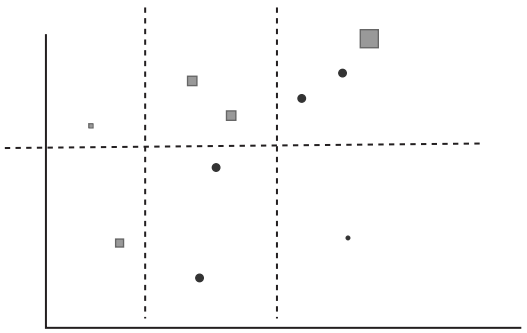

Figure 10-9: AdaBoost decision regions for a simple example

Model accuracy from boosting typically is better than single decision trees or even from bagging. Figure 10-10 shows the AUC for 30 individual trees, averaging predictions from 3 groups of 10 trees (non-overlapping), bagging, and AdaBoost built from 100 iterations. All results are from test data. In each tree, only LASTGIFT and RFA_2F were used as inputs. Clearly, AdaBoost has the highest accuracy; this was true even with only 10 iterations. The box and whiskers plot for the single models shows the minimum and maximum AUC values. Note that bagging has significantly higher AUC than even the best single model, and AdaBoost significantly higher than bagging.

Boosted trees created from the same KDD Cup 1998 data as the results shown in Figures 10-6 have far simpler decision regions than individual trees and the bagged trees, even after 100 iterations. Figure 10-11 shows the decision regions for the AdaBoost predictions. The regions from AdaBoost have smoother transitions between the decisions and the regions themselves are more homogeneous.

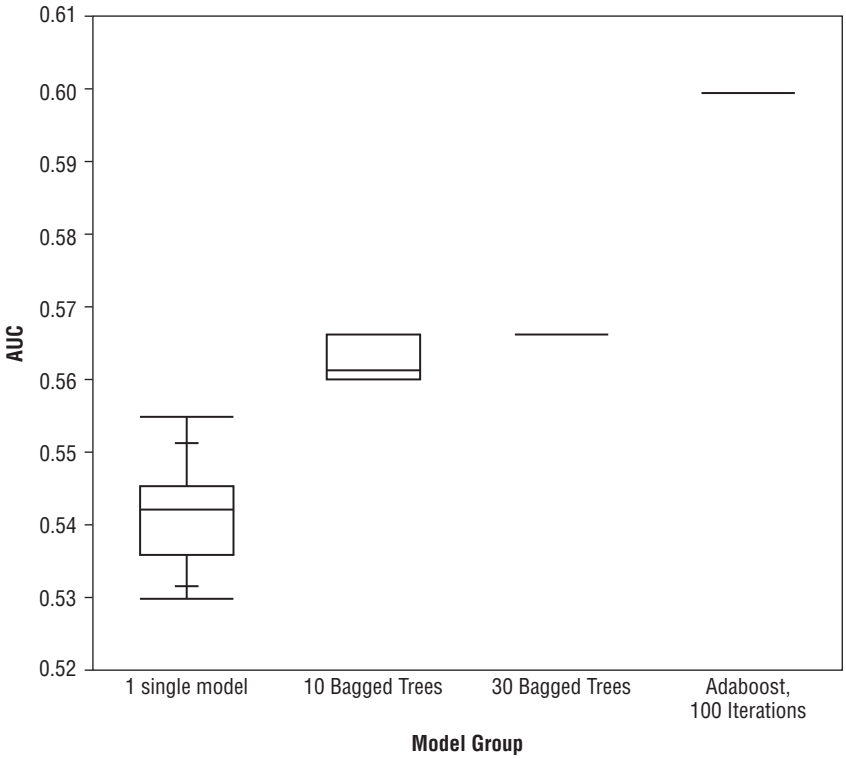

Figure 10-10: Comparison of AUC for individual trees and ensembles

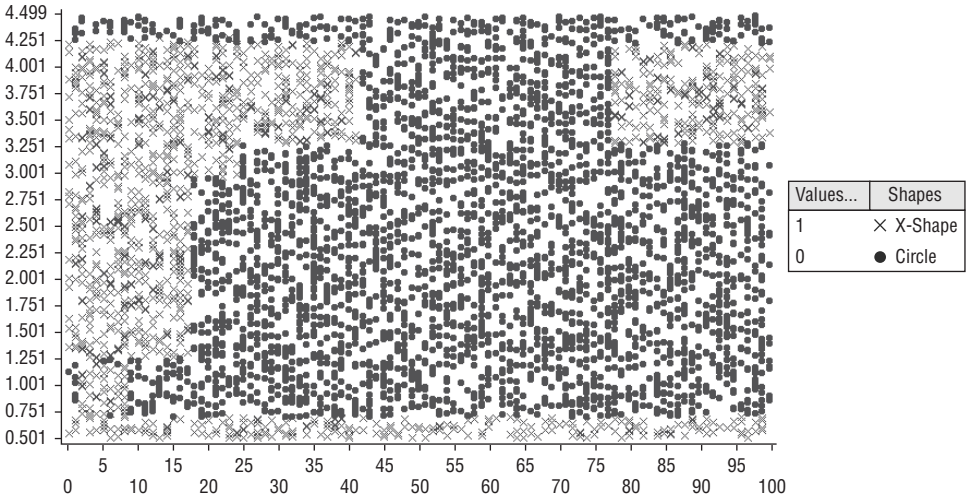

Figure 10-11: Decision regions for the AdaBoost ensemble

Improvements to Bagging and Boosting

Bagging and boosting were the first ensemble methods that appeared in predictive analytics software, primarily with decision tree algorithms. Since their introduction, many other approaches to building ensembles have been developed and made available particularly in open source software. The three most popular and successful approaches are random forests, stochastic gradient boosting, and heterogeneous ensembles.

Random Forests

Random Forests (RF) was introduced by Breiman in 2000 as a modification to the bagging algorithm. As with bagging, the algorithm begins with bootstrap sampled data, and one decision tree is built from each bootstrap sample. There is, however, an important twist to the RF algorithm: At each split in the tree, rather than considering all input variables as candidates, only a random subset of variables is considered.

The number of variables one considers is a parameter in the algorithm specification, but the default is to use the square root of the total number of candidate inputs. For example, if there were 100 candidate inputs for the model, a random 10 inputs are candidates for each split. This also means that it is unlikely that the same inputs will be available for splits at parent and children nodes in the

tree, forcing the tree to find alternate ways to maximize the accuracy of subsequent splits in the tree. Therefore, there is a two-fold diversity in the trees. First, diversity is encouraged by case weights: Some patterns are over-emphasized and others under-emphasized in each tree. Additionally, diversity is encouraged through input weighting ("on" vs. "off").

RF models are typically more accurate than bagging and are often more accurate than AdaBoost.

Stochastic Gradient Boosting

AdaBoost is only one of many boosting algorithms currently documented in the literature, though it and the C5 boosting algorithm are most common in commercial predictive modeling software; dozens of boosting algorithms can be found in the open source software packages.

One interesting boosting algorithm is the Stochastic Gradient Boosting (SGB) algorithm created by Jerry Friedman of Stanford University. Friedman developed an algorithm published in 2001 that he called Multiple Additive Regression Trees (MART) and was later branded as TreeNet® by Salford Systems in its software tool. Like other boosting algorithms, the MART algorithm builds successive, simple trees and combines them additively. Typically, the simple trees are more than stumps and will contain up to six terminal nodes.

After building the first tree, *errors* (also called *residuals*) are computed. The second tree and all subsequent trees then use residuals as the target variable. Subsequent trees identify patterns that relate inputs to small and large errors. Poor prediction of the errors results in large errors to predict in the next tree, and good predictions of the errors result in small errors to predict in the next tree. Typically, hundreds of trees are built and the final predictions are an additive combination of the predictions that is, interestingly, a piecewise constant model because each tree is itself a piecewise constant model. However, one rarely notices because typically hundreds of trees are included in the ensemble.

The TreeNet algorithm, an example of stochastic gradient boosting, has won multiple data mining modeling competitions since its introduction and has proven to be an accurate predictor with the benefit that very little data cleanup is needed for the trees before modeling.

Heterogeneous Ensembles

The ensemble techniques described so far achieve variety through case weights, whether by resampling (bootstrap sampling) or reweighting records based on prediction errors (boosting). These techniques use the same algorithm for every model, typically decision trees. However, variety can also be achieved through varying the algorithm as well, creating a *heterogeneous ensemble*, comprised of two or more different algorithms.

As you saw in Chapter 8, algorithms sometimes achieve the same accuracy on data sets in different ways, disagreeing on individual records even while, on average, having the same accuracy. We observed bagged trees where the correlation between model predictions ranged from 0.212 to 0.431 even though they had the same accuracy.

Heterogeneous ensembles are combined similarly to bagged trees, most often through simple averaging or voting. However, one can also experiment with other ways to combine the models, such as selecting the maximum probability of the models or weighting the predictions by model accuracy (the better models contribute more to the weighted average).

In a simple example using data known as the glass identification data set, Figure 10-12 shows minimum, maximum, and average error rates for six different algorithms on validation data. The best single model on validation data had a 28.1 percent error rate, the worst a 37.5 percent error rate, and the average error rate is 30.8 percent.

The figure shows all possible ensembles built from these six models: all possible two-way combinations, three-way combinations, four-way, five-way, and the single six-way combination. Note that the average error rates on validation data decreases as more models are included in the ensembles. In fact, the *worst* three-model ensemble is better than the *average* single model. The six-model ensemble is among the best of all the ensembles, but it is not the best possible; you never know which particular combination will be the best on validation data.

Ensembles are described as methods that increase accuracy. Based on Figure 10-12, it is clear that ensembles also reduce the risk of deploying a poor model.

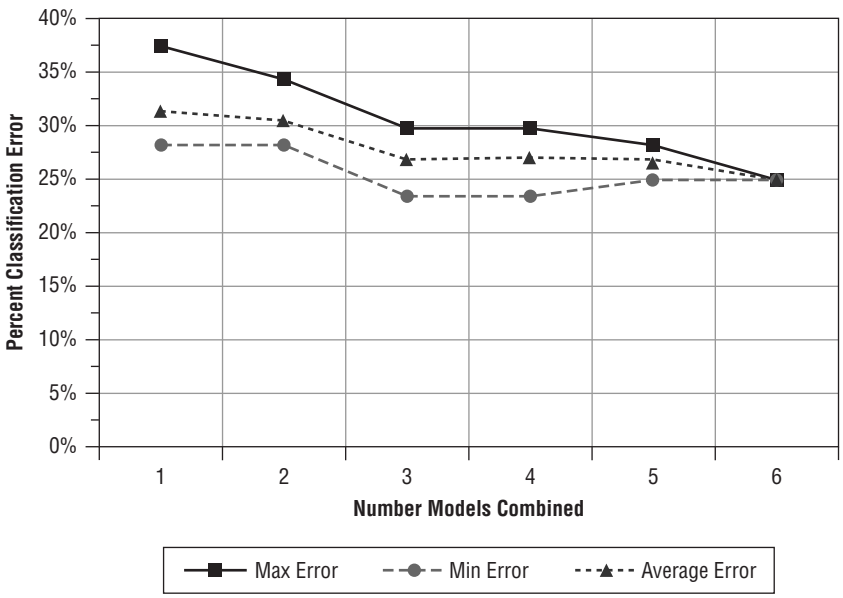

Figure 10-12: Heterogeneous ensemble example

Model Ensembles and Occam's Razor

Occam's razor is a core principle many predictive modelers use. The idea is that simpler models are more likely to generalize better, so it is better to regularize complexity, or in other words, simplify the models so that the inclusion of each term, coefficient, or split in a model is justified by its reducing the error sufficiently to justify its inclusion. One way to quantify the relationship between accuracy and complexity is taken from information theory in the form of information theoretic criteria, such as the Akaike information criterion (AIC), the Bayesian information criterion (BIC), and Minimum Description Length (MDL). Statisticians sometimes use these criteria for model selection during stepwise linear regression, for example. Information theoretic criteria require a reduction in model error to justify additional model complexity.

Do model ensembles violate Occam's razor? Ensembles, after all, are much more complex than single models. Random Forests and Boosted Trees could have hundreds of models combined into the final prediction. Neural network ensembles could contain dozens or hundreds of networks, each of which contain hundreds to thousands of weights. If the ensemble accuracy is better on held-out data than single models, yet these models are far more complex than single models, do we need another paradigm to help us choose which models will generalize well? The answer is "no" as long as we think about the complexity of a model in different terms.

John Elder describes the concept of Generalized Degrees of Freedom (GDF) that measures the *behavior* of models, or as he writes, the *flexibility* of a modeling process. The complexity of linear regression models increases linearly with the number of terms in the model. GDF confirms this as well: A linear increase in coefficients in a model produces a linear increase in the complexity of behavior in the model as well. However, this is not the case for all algorithms. Elder shows an example where Bagged trees built from 50 bootstrap samples have a lower GDF measure than a single tree. Bagging, in fact, smoothes the predictions as a part of the averaging of predicted probabilities, making the model's behavior smoother and therefore simpler.

We should not fear that adding computational complexity (more terms, splits, or weights) will necessarily increase the behavioral complexity of models. In fact, sometimes the ensemble will significantly reduce the behavioral complexity.

Interpreting Model Ensembles

The interpretation of ensembles can become quite difficult. If you build an RF ensemble containing 200 trees, how do you describe why a prediction has a particular value? You can examine each of the trees individually, though this is clearly not practical. For this reason, ensembles are often considered black

box models, meaning that what they do is not transparent to the modeler or domain expert.

Nevertheless, one way to determine which inputs to the model are most important is to perform a sensitivity analysis much like one does with neural networks: When inputs are changed by some deviation from the mean, and all other inputs are held constant, how much does output probability change? One can then plot the sensitivity of the change versus the value to visualize the sensitivity, seeing if the relationship between the input values and the predictions are linear or nonlinear, or if they are monotonic or non-monotonic.

A simple sensitivity analysis can be achieved by building single-split decision trees using an algorithm that allows for multi-way splits (such as CHAID) for each variable. All the variables can be scored, and the variable with the highest chi-square statistic can be considered the most important single variable. As an alternative, if the inputs to the model are continuous variables, an ANOVA can be computed with the predicted target variable class as the grouping variable.

Consider the KDD Cup 98 data set, this time including 12 candidate input variables to the ensemble models. Table 10-2 shows the ranking of the 12 variables based on an ANOVA, where the *F* statistic is the test of significance. The *F* statistic is computed on the test data set with 47,706 records. All of the variables have statistically significant differences in mean values, largely because of the large number of records in the test set. RFA_2F is by far the best predictor, its *F* statistic being more than seven times larger than the second variable. This gives you a sense for which variables are the largest contributors to the ensemble model.

Table 10-2: AdaBoost Variable Ranking According to ANOVA (*F* Statistic)

VARIABLE	F STATISTIC, ADABOOST	P VALUE, ADABOOST	RANK, ADABOOST
RFA_2F	53,560.3	0.0	1
NGIFTALL	7,862.7	0.0	2
MINRAMNT	6,400.0	0.0	3
LASTGIFT	5,587.6	0.0	4
FISTDATE	3,264.9	0.0	5
NUMPROM	2,776.7	0.0	6
MAXRAMNT	2,103.0	0.0	7
RAMNTALL	621.4	0.0	8
NUMPRM12	343.8	0.0	9
HIT	60.9	0.0	10
WEALTH1	60.0	0.0	11
AGE	36.5	0.0	12

A histogram of the top predictor, RFA_2F, color coded by the predicted target variable from the AdaBoost ensemble is shown in Figure 10-13. The figure shows the strong relationship between RFA_2F and the predictions; if RFA_2F is 4, the vast majority of predicted values are 1 (light gray).

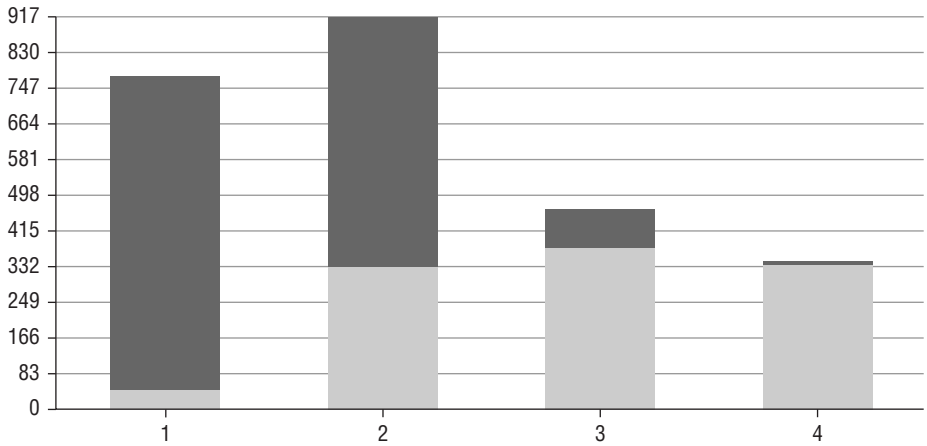

Figure 10-13: RFA_2F histogram with AdaBoost ensemble overlay

If one computes an ANOVA on testing data for Random Forests, AdaBoost, and bagging models, the resulting rankings in Table 10-3. There is great consistency in the story told by all three ensembles: RFA_2F is the top predictor, and the top six predictors are consistently top predictors.

Table 10-3: Comparison of Variable Ranking for Three Ensemble Methods by ANOVA (F Statistic)

VARIABLE	RANK, RANDOM FORESTS	RANK, ADABOOST	RANK, BAGGING
RFA_2F	1	1	1
NGIFTALL	2	2	3
MINRAMNT	3	3	4
LASTGIFT	5	4	2
NUMPROM	4	6	6
FISTDATE	6	5	7
MAXRAMNT	8	7	5
NUMPRM12	7	9	8
RAMNTALL	9	8	11
AGE	10	12	9
HIT	11	10	12
WEALTH1	12	11	10

Summary

Model ensembles are a new frontier for predictive modelers who are interested in accuracy, either by reducing errors in the models or by reducing the risk that the models will behave erratically. Evidence for this is clear from the dominance of ensembles in predictive analytics and data mining competitions: Ensembles always win.

The good news for predictive modelers is that many techniques for building ensembles are built into software already. The most popular ensemble algorithms (bagging, boosting, and random forests) are available in nearly every commercial or open source software tool. Building customized ensembles is also supported in many software products, whether based on a single algorithm or through a heterogeneous ensemble.

Ensembles aren't appropriate for every solution—the applicability of ensembles is determined by the modeling objectives defined during Business Understanding—but they should be in every predictive modeler's toolkit.