**Ensemble Methods and Their Applications**

**Nisha Iyer**

**G39258651**

**CS6907- Big Data and Analytics**

**Spring 2016**

**Ensemble Methods and Their Applications**

Data science has become a very popular term over the last decade, even more so over the last five years. The ability to combine statistics and computer science in a meaningful way, where information is pulled from the abundance of data that is increasingly available, is very attractive to a wide range of industries. Building predictive models is one area of data science, which undoubtedly combines statistics and computer science to build upon ‘machine learning algorithms’. Some of the most popular machine learning algorithms use methods that make up ensemble models. These methods such as bagging, boosting, and combining models, aim to achieve less variance and more accuracy in prediction. Ensemble methods have been showcased in famous data science competitions such as KDD(\*), where the winning models have been ensemble models for multiple years in a row. This illustrates the power of ensemble models.

**1. Introduction**

Ensemble methods are those that consist of the three areas discussed above, bagging, boosting and other ways of combining multiple models. This paper will discuss Ensemble methods, their role in Machine Learning, and the importance and value of using them. I currently work for Discovery Channel on a young Data Science team. I say that it is a young team to highlight that the structure, ideas and modeling techniques used are still very malleable. We are currently working with large amounts of data (Big Data) to build look-alike models and uplift type models. I chose to look further in to Ensemble methods because of the type of models I think would best suit the data (and frequently any time you are working with big data) that we work with at Discovery.

The aim of this paper is to provide an overall and introductory explanation to ensemble modeling, the various methods used when building ensemble models and to dive deeper in to a few of those specific models. Lastly, I will run through a demo to illustrate functionality and compare two models using a simple data set; Random Forest and AdaBoost. My goal is to understand the ensemble methodology as well as a thorough understanding of how to apply ensemble models to big data so that the best possible predictive outcome, particularity classification, is achieved.

The paper is organized as follows: I will first discuss the background and methodologies needed to understand ensemble methods; bootstrapping, bagging and boosting as well of a few well known dependent ensemble methods,Random Forest and AdaBoost, that use bagging and boosting respectively. Next, I will outline the idea of Ensemble Methods in detail, discussing the two main classes which both stem from ‘base learners’ (Zhi-Hua, 2012) either to form ‘homogenous ensembles’ or ‘heterogenous ensembles’. In the case of heterogenous ensembles, the idea of individual learners vs. base learners will also be discussed. Finally, I will walk through a sample dataset using various methods discussed and outlining results.

The objective of this paper is to help the reader understand ensemble methods, have a broad overview of the different types of ensemble method and a slightly deeper dive into a few of them.. The motivation behind writing this paper is observing many people (within my MS program – MS Business Analytics, at work and when working on competitions and looking at model entries) that do not take advantage of Ensemble methods and the powerful models that they create. The reader of this paper should be able to walk away feeling confident that next time they approach a large data set, or a data mining problem of any kind, they will consider using an ensemble model approach.

**2. Background**

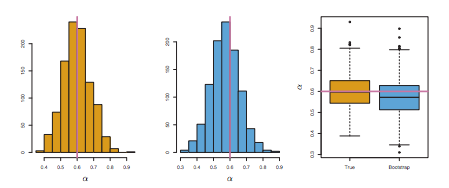
Ensemble methods have been around for many years, as multiple sources are unable to cite the exact start of when they first came about. This is due to the broad nature of the methodology, which at high level, is combining multiple explanations and building upon these to create the strongest explanation. “The Greek philosopher Epicurus (341-270 B.C.) introduced the *principle of multiple explanations’* which advocated to keep all hypotheses that are consistent with empirical observations.” (Zhi-Hua, 2012, p. 16). There are three main concepts that are necessary to understand before going into the idea of ensemble methods as a whole:; bootstrapping, bagging, and boosting.

**2.1 Bootstrapping**

Bootstrapping is a concept well known in the field of statistics. It was first founded by Efron and Tibshirani in the 1980s. The basic concept of bootstrapping is as follows:

Let {X1 + X2 + … + XN} be a random sample of size N with each Xi independently drawn from the same population that has a cumulative distribution function G that is characterized by a parameter θ. Our estimator θ is symbolized as θhat, and we wish to know its sampling distribution. For the random variables, {X1 + X2 + … + XN}, we observe a given sample {x1, x2, …, xn}. The bootstrap method samples from the population defined by the *empirical* distribution function GN, to estimate the sampling distribution of θhat. The empirical distribution function, GN­, is that distribution with mass 1/N on x1, x2, …, xN. That is, we form a bootstrap sample, {X1 + X2 + … + XN}, by taking N independent draws *with replacement* from {x1, x2, …, xn}. The bootstrap estimate, θhat\*, is computed using the same formula as that for θhat, but is calculated using the bootstrap sample. Repeating this process B times gives θhat\*(1),θhat\*(2),….θhat\*(B). From these bootstrap replicates of θhat we estimate the bootstrap distribution of θhat including its mean and variance.” (Bollen & Stine, 1992, p. 207)

Displayed below is an example of a histogram of 1,000 bootstrap estimates of α, each computed using a distinct bootstrap data set. This panel was constructed on the basis of a single data set, and hence could be created using real, attainable data opposed to building multiple data sets with new data for each set. This example was taken from Introduction to Statistical Learning with Application in R (James, Witten, Hastie, & Tibshirani, 2013, p. 189).



*Left: A histogram of the estimates of α obtained by generating 1,000 simulated data sets from the true population. Center: A histogram of the estimates of α obtained from 1,000 bootstrap samples from a single data set. Right: The estimates of α displayed in the left and center panels are shown as boxplots. In each panel, the pink line indicates the true value of α.*

Bootstrapping is so well received because it can take a sample and make it something that you are able to work with and model off of, even though at first glance the sample may not meet the assumptions or properties that you are looking for. By bootstrapping, mean and variance are controlled and the assumptions for multiple models are met. Also, boostrapping mitigates the effects of having a small sample size or an unequal distribution of the binary target variable, e.g. when there is a bias towards one class in the target. Bootstrapping also is a concept that is used within the main two methods of ensemble modeling; bagging and boosting. Without bootstrapping, we would need to continuously take multiple samples from a population. A task that is both time consuming and cost ineffective.

Bootstrapping can be used as a method to ‘oversample’ data in order to normalize mean and variance, as discussed above. However it is also an important component in ensemble methods ‘bagging’ or *bootstrap aggregation’.* While bootstrapping is an important and effective way in “assessing the accuracy of a parameter estimate or prediction”, bagging uses bootstrapping to “improve the estimate or prediction itself” (Hastie, Tibshirani, & Friedman, 2008, p. 212).

2.2 Bagging:

Bagging is “a general-purpose procedure for reducing the variance of a statistical learning method” (James, Witten, Hastie, & Tibshirani, 2013). Bagging was first introduced to the world of statistics and machine learning in 1994 by well known statistician Leo Breiman of the University of California, Berkley. Bagging incorporates bootstrapping within a model so that the instead of assessing the accuracy after prediction, the prediction itself is improved by applying bootstrapping to the predictors of the model. The Elements of Statistical Learning gives a definition along with a simple example that in summary describes bagging as “averaging the prediction over a collection of bootstrap samples, thereby reducing its variance” (Hastie, Tibshirani, & Friedman, 2008). In a regression example, the bagged estimate will differ from the original estimate if the original estimate is non-linear.

Usually, bagging is more interesting when seen within classification trees. It is the method used in popular classification ensemble methods such as Random Forest. When used with decision trees, the basic idea of bagging is to bootstrap the training set and take multiple samples, building trees on each one of these bootstrapped samples. Combining the output of each of these trees and then taking the average prediction from the *B* number of trees gives you the ‘bagged estimate’. Because these trees are each grown deep and not pruned, each individual *B* tree will have a high variance, but low bias. This is the final equation obtained, the method of choice trained on the bootstrapped sets and then averaged to come up with the final prediction:



The process of averaging the trees then reduces the overall variance, accounting for the over fitted smaller trees by averaging their predictions to get the final prediction. (Breiman, 2001)

As stated by (James, Witten, Hastie, & Tibshirani, 2013, p. 316), “ in other words, averaging a set of observations reduces variance. Hence a natural way to reduce the variance and hence increase the prediction accuracy of a statistical learning method is to take many training sets from the population, build a separate prediction model using each training set, and average the resulting predictions”. Some other terms that are associated with bagging are ‘Out of Bag’ error estimation and ‘Variable Importance Measures’. These will be discussed further on in the paper.

2.3 Random Forests and AdaBoost

Random Forest was also developed by Leo Breiman of UC Berkley. He published a paper ‘Random Forests’ and formally introduced the learning method in October 2001. Breiman defines random forests as:

“Definition 1.1. A random forest is a classifier consisting of a collection of tree-structured classifiers {h(x, k ), k = 1,...} where the {k } are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x. “ (Breiman, 2001)

Random Forests uses bagging primarily but adds a step that helps with decorrelation. The multiple trees are built on a bootstrapped sample from the training set. Then, the Random Forest algorithm chooses a random sample of *m* predictors as split candidates from the total set of *p* predictors. ‘The split is allowed to use only one of those m predictors. A fresh sample of m predictors is taken at each split, and it’s typically *m* ≈ √*p*—that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors (James, Witten, Hastie, & Tibshirani, 2013). This differentiates Random Forest from simply using bootstrap aggregation since the predictors for each tree are randomly chosen from the full set of predictors, and the bootstrapped training sample can only be split on those predictors. Whether they be the worst predictors within the full set of predictors or not, this actually helps balance out the choice of variables to split on and therefore decorrelates the data set. Strong predictors that may always be chosen if the trees were simply bagged, which would mean that the number of variables available to split on for each tree would be *m = p* . Where as setting *m* ≈ √*p* allows for strong predictors to be left out of some of the predictions, leading to less correlated predictions from each tree and therefore overall variance will be decreased by the averaging of predictions more so than simply bagging.

“Adaboost is a random forest” says Leo Breiman in his article ‘Random Forests’ from the Machine Learning Journal (Breiman, 2001). Although they are not the same algorithm, Adaboost uses boosting methods while Random Forest uses bagging methods, Breiman speaks about an experiment he conducted:

In our experiment, random forests were produced as follows: Adaboost was run 75 times on a data set producing sets of non-negative sum-one weights w(1), w(2), . . . , w(50) (the first 25 were discarded). The probability for the kth set of weights is set proportional to Q(wk ) = log[(1 − error(k))/error(k)] where error(k) is the w(k) weighted training set error of the kth classifier. Then the forest is run 250 times. This was repeated 100 times on a few data sets, each time leaving out 10% as a test set and then averaging the test set errors. On each data set, the Adaboost error rate was very close to the random forest error rate. A typical result is on the Wisconsin Breast Cancer data where Adaboost produced an average of 2.91% error and the random forest produced 2.94%... (Breiman, 2001, p. 20)

He goes on to say that the reasoning for the similarity is that Adaboost is equivalent to a random forest “where the weights on the training set are selected at random from the distribution”. The one caveat according to Breiman is that Adaboost distributes weights dependent on the training set while random forest does not rely on the training set for the distribution of the random vectors of predictors. This will overall help with bias- variance tradeoff in random forest’s favor. However, adaboost is a very popular algorithm and this brings us to out next ensemble learning method, boosting.

2.4 Boosting

Boosting is a similar ensemble method to bagging, it also effectively reduces bias and variance. Boosting came about a little before the concept of bagging. Robert E. Schapire responded to a question, “can a set of weak learners create a single strong learner?” posed by Michael Kearns and Leslie Valiant (Kerns & Valiant, 1989) . In his article, “the strength of weak learnability” , Schapire addresses the the idea of improving weak learners by combining them together and coming to a learner that achieves high accuracy. The basic concept of boosting is that instead of bootstrapping the training set to get multiple samples, the samples are derived from previous samples. It can be employed on both classification and regression problems, so seems to be better suited all around. Bagging often does better within classification problems.

“The purpose of boosting is to sequentially apply the weak classification algorithm to repeatedly modified versions of the data, thereby producing a sequence of weak classifiers *Gm(x),m=*1, 2, …, *M*. The predictions from all of them are then combined through a weighted majority vote to produce the final predictions:



Here α1, α2, …,αM are computed by the boosting algorithm, and weight the contribution of each respective *G­m(x).* Their effect is to give higher influence to the more accurate classifiers in the sequence” (Hastie, Tibshirani, & Friedman, 2008, p. 338). Boosting is a slow learner, as it takes time and computational effort to look in to which observations are misclassified and then train these observations by increasing weights, so that they are classified correctly. Slow learners, in statistics, do tend to perform better than other ensemble methods.

ISLR (James, Witten, Hastie, & Tibshirani, 2013) describes the three tuning parameters of boosting:

1. **The number of trees B.** Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.

2. **The shrinkage parameter λ, a small positive number.** This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.

3. **The number d of splits in each tree, which controls the complexity of the boosted ensemble.** Often d = 1 works well, in which case each tree is a stump, consisting of a single split. In this case, the boosted stump ensemble is fitting an additive model, since each term involves only a single variable. More generally d is the interaction depth, and controls interaction the interaction order of the boosted model, since *d* splits can involve at most *d* variables.

2.5 Other Ensemble Methods

In addition to the three methods discussed above, bootstrapping, bagging and boosting, and respective models that illustrate these methods: Random Forest and Ada Boost, there are other methods that are employed within ensemble modeling. Two that will be discussed are Model Averaging and Stacking and Bumping.

2.5.1 Model averaging and stacking:

Bootstrap and bagging are both methods of Bayesian model averaging with specific parameters imposed. We can also look at Bayesian model averaging in a general way to get a more broad view of the method and how it could be applied to areas where you may not be using bootstrap or bagging. The basic idea is take *Mm, m =* 1,. . ., *M,* this will be your training set Z. Z isa set of models, whether they be of the same type using different parameters (regression using different subsets) or different models with the same target/outcome (neural networks and regression trees). Each model is run and predicted, and the Bayesian prediction comes in as a weighted average of the individual predictors with weights proportional to the posterior probability of each model. (Hastie, Tibshirani, & Friedman, 2008, p. 289)

*Committee methods* is one of the strategies that the above formulation leads to. It consists of taking a simple unweighted average of the predictions from each model. This gives each model the same probability or weight.

*Stacking methods* are methods that apply weights to models based on their complexity. The way this is done is by a least squares regression (or generally any learning method) on the set of models to decide the weights to be given to each model. Stacking uses cross-validation predictions to ensure a highly complex model will not be weighted too high.

2.5.2 Bumping

This method does not involve combining or averaging models, but instead it incorporates a technique for finding a better single model. “*Bumping* uses bootstrap sampling to move randomly through model space. For problems where fitting method finds many local minima, bumping can help the method to avoid getting stuck in poor solutions" (Hastie, Tibshirani, & Friedman, 2008, p. 290). Like bagging, we take bootstrapped sets from the training set and fit a different model to each bootstrap set. Now, instead of taking the average of these predictions, we use the model with the lowest mean squared error, and apply this model to the entire bootstrapped training set. We make sure to include the original training set as one of the sets that we run the model on. The idea is that bumping will then find the data set that best fits the best model (maybe a bootstrapped set that does not contain certain data points performs best). Bumping can help in problems where it is difficult to optimize fitting criteria. One thing to watch out for with bumping is that model complexity is about the same. When first fitting multiple models to bootstrap training sets, if model complexity is not similar, bumping will not be effective.

**3. Ensemble Methods**

In this section, ensemble methods will be covered in depth. First, I will discuss combination methods, which include averaging, voting and stacking. Then I will touch upon infinite ensembles and support vector machines. Next, I will go over three popular ensemble learning algorithms in detail: Random Forest, AdaBoost and general stacking methods – which usually involves combining multiple models of similar complexity. I will then walk through a sample data set using each algorithm discussed about, and evaluate performance.

3.1 Independent and Dependent Ensemble Methods

Ensemble methods generally fit in to two categories, dependent or independent, which are formed by the extent each classifier affects the other classifiers. In the dependent methods, the ‘base learner’ or base model affects the creation of the next model. Independent methods are the opposite of this where there are several ‘base learners’ and their outputs are combined in one of many ways (which will be discussed in following sections). There are three different ‘types’ that researchers refer to when discussing independency and dependency. These types are ‘successive’ which refers to dependent and ‘cooperative’ which refers to independent models. The last type is ‘supervisory’ and this refers to either dependent or independent models, where one model controls the other model.

3.1.1 Dependent Methods

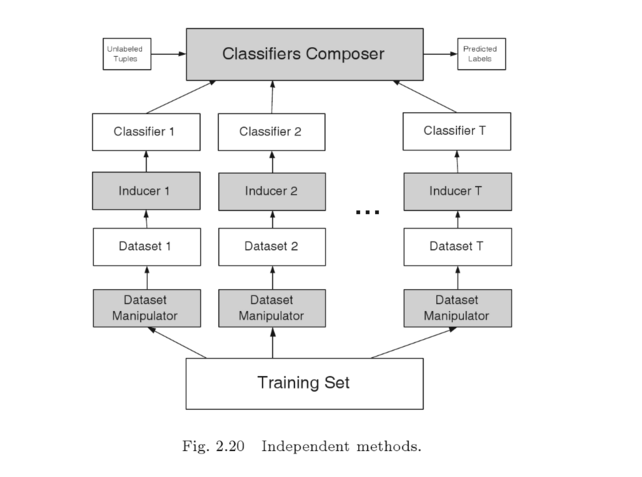
Dependent methods are those where there is interaction between the learners as they are created. So the base learner is not independent of the model created after it and so on. There are two main classes within dependent learners, model guided instance selection and basic boosting algorithms. Model guided instances are those where error from the initial or previous model is modeled upon, so that the residuals may be ‘attacked’. The area of concentration is the error of the previous model in this methodology.

Boosting algorithms are the second dependent methodology. The background of boosting was covered in the Background section of this paper, and boosting will be illustrated in detail in upcoming sections. The idea of boosting is to combine multiple weak learners to produce a strong learner. “The method works by repeatedly running a weak learner (such as classification or decision trees), on various distributed training data. The classifiers produced by the weak learners are then combined into a single composite strong classifier in order to achieve a higher accuracy than the weak learner’s classifiers would have had” (pattern classification using ensemble methods) (Rokach, 2010).

A third form of dependent methodology is incremental batch learning. This is a Bayesian type method where classification produced in one iteration is given as “prior knowledge” to the learning algorithm in the following iteration. The new learning algorithm is conditional on the previous learning algorithm. The last classifier constructed in the iteration is the final classifier.

3.1.2 Independent methods

Independent methods cover a larger portion of ensemble modeling. These methods are when “the original dataset is partitioned into several subsets from which multiple classifiers are induced” (Rokach, 2010, p. 51). An illustration (below) taken from Pattern Classification Using Ensemble Methods visualizes the general process of independent methods:



*The subsets created from the original training set may be disjointed (mutually exclusive) or overlapping. A combination procedure is then applied in order to produce a single classification of a given instance.*

A very popular algorithm that come from independent learning is bagging, which was briefly covered in the Background section, and will be covered in depth in upcoming sections.

3.2 Combination Methods

Combination methods are when ensemble methods start working their magic to improve final outcome. Base learners have been built and now comes the point where combining them will hopefully improve the prediction or accuracy of the final model. This means that after generating base learners, instead of finding the best base learner out of the group, ensemble methods will combine these base learners to “achieve a strong generalization ability” (Zhi-Hua, 2012) , this is where combination plays a significant role.

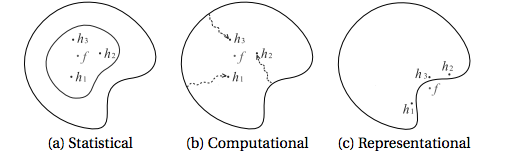
There are three fundamental reasons why combination methods and ensemble modeling may work better at prediction than a single best model. The reasons behind them consist of statistical, computational and representational issues. To understand these issues, one must be familiar with the term ‘hypotheses space’. The Encyclopedia of Machine Learning defines hypothesis space as, “The hypothesis space used by a machine learning system is the set of all hypotheses that might possible be returned by it” (Sammut & Webb, 2011, pp. 511-513).

Statistical issue: “by combining the hypotheses, the risk of choosing the wrong hypothesis can be reduced” (Zhi-Hua, 2012, p. 67). When the hypothesis space is too large to explore, the single learner could choose the incorrect hypothesis on the training set. When combining multiple hypothesis from bootstrapped training sets, the risk is reduced of choosing the wrong hypothesis.

Computational issue: “by combining the hypotheses, the risk of choosing a wrong local minimum can be reduced” (Zhi-Hua, 2012, p. 67). Since many learning algorithms perform a ‘local’ search for local optima, the learner may get stuck on a local optima point that is actually not close to the global optima. When diversifying the starting point of the learner by using multiple bootstrapped training sets, this increases the chances of the global optima or true, unknown hypothesis.

Representational issue: “by combining the hypotheses, it may be possible to expand the space of representable functions, and thus the learning algorithm may be able to form a more accurate approximation to the true unknown hypothesis” (Zhi-Hua, 2012, p. 67). The case may be that the training set does not hold the true hypothesis, so by combining multiple predictions of what that is, we are more likely to get close to the actual unknown hypothesis than basing on the accuracy of one champion model.

Below is a diagram taken from the book Ensemble Methods that illustrates the three main issues:



*Three fundamental reasons for combination: (a) the statistical issue, (b) the computational issue, and (c) the representational issue. The outer curve represents the hypothesis space, and the inner curve in (a) represents the hypotheses with the same accuracy on the training data. The point label f is the true hypothesis, and hi’s are the individual hypotheses. (Plot based on a similar figure in [Dietterich, 2000a].)*

From the three issues stems bias-variance issues that we are constantly looking to improve. We see that from the statistical issue, high variance can occur if not treated by ensemble models, from the computational, high computational variance and from the representational, high bias. “These three fundamental issues are the most important ways in which existing learning algorithms fail. Hence, ensemble methods have the promise of reducing (and perhaps eliminating) these three key shortcomings of standard learning algorithms” (Dietterich, 2000).

3.2.1 Averaging

There are two main types of averaging, simple and weighted. Simple averaging is one of the most common applications of ensemble modeling to real world problems. Individual learners are combined by averaging the outputs directly. Taken from Ensemble Methods (Zhi-Hua, 2012), the equation below illustrates Simple Averaging, where the combined output *H(x)*  is illustrated as:



Where the output of each learner is written as:



It is a quite simple method of combining models, and based on this itself is the reason for its broad usage within machine learning community. One assumption of this method is that residuals of each model are not correlated. However, when working with independent learners on the same data set, errors will tend to be correlated. Therefore the optimum performance of simple averaging, which is to get ensemble error to be smaller than averaged error of individual learners by a factor of *T*, may not be met. But simple averaging will still usually perform better than using a single learner.

Weighted averaging is the same idea as simple averaging, except now adding weights to each of the individual learners. By using the Lagrange multiplier method, optimal weights for each learner is found. The Lagrange multiplier method is beyond the scope of this paper. However, the assumption is that error terms are not correlated, and once again they will tend to be correlated in ensemble method instance since the individual learners are all on the same data set.

Therefore, this method and the validity of the Lagrange multiplier weights may not always produce an optimal outcome. Simple averaging may be viewed as a special case of weighted averaging, where every model gets equal weight. Note that weighted averaging may not always be better than simple averaging since it’s computationally difficult to produce accurate weights because of the issues discussed. “In general, it is widely accepted that simple averaging is appropriate for combining learners with similar performances, whereas if the individual learners exhibit nonidentical strength, weighted averaging with unequal weights may achieve a better performance” (Zhi-Hua, 2012, p. 70).

3.2.2 Voting

For classification problems that will have a nominal output, voting is the most popular and fundamental combination method. The idea of voting is described using a basic classification example. Suppose we are given a set of *T* individual classifiers {h1, . . ., hT}, and our task is to combine the prediction of class labels

{c1, . . ., cl} for each of those classifiers {h1, . . ., hT}, . By combining the outcome of each of these classifiers, we are able to come up with our idea of the best prediction of {c1, . . ., cl} overall. There are two methods of labeling. Crisp label is when the classifier will either take a one if it matches the class label of interest and a zero otherwise. Class probability will rank the class label given to the classifier as an estimate of the posterior probability of the class label given x (class vector). The idea of combining models to predict overall classifier show that probability class probability is usually much more accurate when used as a combination method than when used with a single learner. There are four main types of voting: majority, plurality, weighted and soft voting.

Majority voting is the most popular type of voting method. Each classifier votes for one class label, and when combined, the class label with more than half the ‘votes’ from classifiers will be the chosen class label. If none of the class labels receive more than half the votes, then a rejection option is given and the combined classifier does not make any prediction.

Plurality voting, in contrast to majority voting which takes the class label with over half the votes, simply takes the class label with the most votes. Ties are broken arbitrarily.

Weighted voting will be used if the case is that the individual classifiers are with unequal performance. In this case, the strongest classifier will be given a larger weight than the next level classifier and so on. When weights are chosen accurately, weighting can be a more accurate voting method than majority voting. How the weights are chosen are an important piece of weighted voting. Although the exact method of choosing weights is beyond the scope of this paper, the issue with weighted voting is that it assumes that the outputs of individual classifiers are independent. In ensemble modeling, this will usually not be the case as independent classifiers are all using the same dataset. It also does not take prior probabilities of classes in to account. Therefore, the assumptions are not usually met and weighting usually will not be as accurate as the majority voting method.

Soft voting is generally used for those classifiers that do not produce crisp labels, but instead produce probability outputs. There is a simple soft voting method that will combine the probability of each of the individual classifiers, and the final probability is the average of all the individual probabilities. Usually soft voting methods are used for homogenous ensembles; class probabilities cannot be simply averaged for heterogeneous ensembles. More careful thought and process must go into combining probability from different independent learners. In these situations, class probabilities of each model are usually converted to crisp labels by implementing a cutoff for probabilities that go to 1 and label the remaining 0. Then voting methods for crisp labels can be applied.

3.2.3 Stacking

Stacking, unlike voting, is a technique where “a learner is trained to combine the individual learners. Here, the individual learners are called the first-level learners, while the combiner is called the second-level learner, or meta-learner” (Zhi-Hua, 2012, p. 83).

The first level learners are typically constructed on a separate data set than the second level learners. The procedure as follows, a first learner is trained on the original training set, then a new second data set is generated for the second learner. The original labels are used for the new data set, however the inputs are the outputs of the first level learner. Therefore, the second level learner is modeling the first learner’s output. This encompasses the stacking method, ‘stacking’ models on top of each other. Usually, stacking is employed with heterogeneous models, however the same model can also be used.

3.2.4 Infinite Ensemble

Infinite ensembles are based on using support vector machines. Lin and Li from the California Institute of Technology, proposed this as a solution to increase the finite limit of base hypotheses space. In their paper, they describe the method used in infinite ensembles:

“The key of the framework is to embed an infinite number of hypotheses into an SVM kernel. Such a framework can be applied both to construct new kernels for SVM, and to interpret some existing ones (Lin, 2005). Furthermore, the framework allows us to compare SVM and ensemble learning algorithms in a fair manner using the same base hypothesis set” (Li & Li, 2008).

Without getting in to too much detail, an overview of the method is that there is a stump kernel and a perceptron kernel. The stump kernel embodies infinite decision stumps (trees that are cut of after one split) and the perceptron kernel embodies infinitely many perceptrons. The outcome is infinite base hypotheses space where SVMs use these kernels to perform ensemble stacking. In Lin and Li’s paper, they state that this method is proven to perform better than ensemble methods with a finite base hypotheses space. The remainder of the paper goes in depth about the methodology of incorporating SVMs within the two kernel spaces, stump and perceptron, to build and infinite ensemble model. This is beyond the scope of this paper.

3.3 Ensemble Method Algorithms

The Random Forest and AdaBoost algorithms outlined below were discussed briefly in the Background section of this paper. They are key ensemble learning algorithms and can be used as dependent ensembles; modeling multiple of one model, or independent ensembles; modeling randomforest and adaboost together as heterogeneous ensemble methods. Here they will be outlined in detail by performing each algorithm on a sample set of data and explaining each step.

3.3.1 Random Forest

The general idea of random forests has been covered in the background section. This is an ensemble method that uses bagging, in addition to bagging, the predictors used to split each individual tree are selected at random from the full set of predictors. In simple bagging, the predictors would equal the full set of predictors for each individual tree. This is where the random forest differentiates itself and has an edge over simply bagging decision trees. Random forests are used for prediction accuracy as well as in preliminary stages of modeling during dimensionality reduction.

Two important concepts that come from random forests are variable importance and out of bag error. Out of bag refers to the data that is outside of the bagged tree. multiple trees are built in random forests using the bagging method. So within a training set, a bootstrapped sample is taken, a tree is built and the ‘out-of-bag’ data is that which is not included in the bootstrapped sample. Using these out of bag observations, the algorithm is able to internally validate itself, by computing a misclassification rate for the singular tree within the larger spectrum of the training data. Variable importance is calculated through the output of the out of bag error. The percent increase in misclassification rate as compared to the out-of-bag rate (with all variables in tact) makes up the output. Other ways of measuring the difference in misclassification is by using Gini or Entropy which both reference node purity. The discrepancy in the Gini rate when the variable at question is in the tree and when it is removed, tells us how important that variable is. For example, if a tree always splits on a certain variable when it is within the group of randomly selected predictors, and therefore the tree has a higher node purity number – this would demonstrate that this variable is higher amongst the importance rate. Finally, decrease in mean square error can also be used, when dealing with regression trees vs. classification trees. This would be that the mean square error seems to increase when the variable at question is not in the chosen predictors.

I will now walk through a random forest model, built on the Boston Housing data set, in R.

### , followed by an AdaBoost model. I want to use the same data set for comparison of these two methods.

library(MASS)  
library(caret)

## Loading required package: lattice  
## Loading required package: ggplot2

library(randomForest)

## randomForest 4.6-12  
## Type rfNews() to see new features/changes/bug fixes.  
##   
## Attaching package: 'randomForest'  
##   
## The following object is masked from 'package:ggplot2':  
##   
## margin

library(ROCR)

## Loading required package: gplots  
##   
## Attaching package: 'gplots'  
##   
## The following object is masked from 'package:stats':  
##   
## lowess

library(miscTools)  
  
#load Boston Housing data   
#This example will be regression trees since the target variable is continuous (median value).  
data(Boston)  
head(Boston)

## crim zn indus chas nox rm age dis rad tax ptratio black  
## 1 0.00632 18 2.31 0 0.538 6.575 65.2 4.0900 1 296 15.3 396.90  
## 2 0.02731 0 7.07 0 0.469 6.421 78.9 4.9671 2 242 17.8 396.90  
## 3 0.02729 0 7.07 0 0.469 7.185 61.1 4.9671 2 242 17.8 392.83  
## 4 0.03237 0 2.18 0 0.458 6.998 45.8 6.0622 3 222 18.7 394.63  
## 5 0.06905 0 2.18 0 0.458 7.147 54.2 6.0622 3 222 18.7 396.90  
## 6 0.02985 0 2.18 0 0.458 6.430 58.7 6.0622 3 222 18.7 394.12  
## lstat medv  
## 1 4.98 24.0  
## 2 9.14 21.6  
## 3 4.03 34.7  
## 4 2.94 33.4  
## 5 5.33 36.2  
## 6 5.21 28.7

dim(Boston)

## [1] 506 14

#Use createDataPartition from the caret package to create train and test sets  
set.seed(123)  
split <- createDataPartition(y=Boston$medv, p = 0.7, list=FALSE)  
train <- Boston[split,]  
test<- Boston[-split,]  
  
#A note about 'createDataPartition; createResample can be used to make a simple bootstrap and createFolds for   
#cross-validation groupings.

### So now training and test sets have been created. Now we will run randomForest on the training data. The randomForest algorithm below includes target variable (medv) '.' which is all the predictors, the data is BostonTrain, number of trees to create using 'bagging' method is 100 and what makes the random forest algorithm set apart from boosting; number of predictors randomly chosen from full set of predictors is 5. In regression trees, the recommended number for mtry is the total number of predictors divided by three. In classification (an example to follow) the recommended number is the square root of predictors.

#Build the model:  
rf <- randomForest(medv~., data=train, mtry=6, importance = TRUE)  
rf

##   
## Call:  
## randomForest(formula = medv ~ ., data = train, mtry = 6, importance = TRUE)   
## Type of random forest: regression  
## Number of trees: 500  
## No. of variables tried at each split: 6  
##   
## Mean of squared residuals: 11.21406  
## % Var explained: 86.21

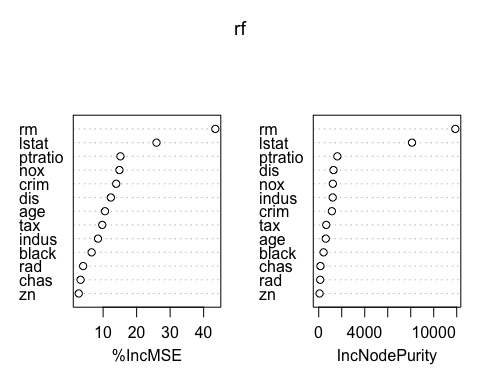
#Now make predictions on the test set:  
yhat <- predict(rf, test)  
  
#The MSE:  
mean((yhat - test$medv)^2)

## [1] 16.23085

#Look at variable importance; looking at increase in MSE by looking at  
#mean decrease in accuracy in predictions on out  
#of bag samples, when the given variable is excluded from the model and  
#increase in node purity by looking at the total decrease in node purity   
#resulting from the given variable, averaged over all trees.  
importance(rf)

## %IncMSE IncNodePurity  
## crim 13.905168 1144.45272  
## zn 2.721400 72.83503  
## indus 8.463850 1213.97948  
## chas 3.270861 147.91585  
## nox 14.871973 1227.67252  
## rm 43.496630 11880.58675  
## age 10.555111 611.36956  
## dis 12.314255 1285.96179  
## rad 4.030756 128.15980  
## tax 9.742466 652.82292  
## ptratio 15.170913 1619.17704  
## black 6.569988 420.56903  
## lstat 25.927797 8111.12283

#Now plot it:  
varImpPlot(rf)



3.3.2 AdaBoost

AdaBoost is a boosting based algorithm that was also briefly touched upon in the backgroung section of this paper. The basis of AdaBoost is that a group of weak learners can be built in to a strong learner by eliminating residuals, in a very slow and computational process. This process involves building a base learner on a bootstrapped sample of training data and then taking the error from the sample and building the next bootstrapped with those error terms and the labels from the original model. By doing this, the errors of the weak learner are eventually diminished and the model becomes a strong learner. The intuition behind AdaBoost, and boosting in general, is that it is easier to find a strong learning by correcting weaknesses than to look for the one that performs as a strong learner by itself. AdaBoost is also good about not overfitting models, as the process is iterative and slower to get rid of the residuals. Now we will walk through a quick example of AdaBoost on the Iris data set, in R. At the end of the R section, I will quickly compare the AdaBoost and randomforest models:

We will now use the iris data set to demonstrate the AdaBoost ensemble algorithm.

library(adabag)

## Loading required package: rpart  
## Loading required package: mlbench  
## Loading required package: caret  
## Loading required package: lattice  
## Loading required package: ggplot2

library(randomForest)

## randomForest 4.6-12  
## Type rfNews() to see new features/changes/bug fixes.  
##   
## Attaching package: 'randomForest'  
##   
## The following object is masked from 'package:ggplot2':  
##   
## margin

data(iris)  
##adaBoost  
data(iris)  
head(iris)

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 1 5.1 3.5 1.4 0.2 setosa  
## 2 4.9 3.0 1.4 0.2 setosa  
## 3 4.7 3.2 1.3 0.2 setosa  
## 4 4.6 3.1 1.5 0.2 setosa  
## 5 5.0 3.6 1.4 0.2 setosa  
## 6 5.4 3.9 1.7 0.4 setosa

set.seed(123)  
split <- createDataPartition(y=iris$Species, p = 0.7, list=FALSE)  
train <- iris[split,]  
test<- iris[-split,]  
train$Species <- factor(train$Species)  
adaboost<-boosting(Species ~ . , data=train, boos=TRUE, mfinal=20, coeflearn='Breiman')  
summary(adaboost)

## Length Class Mode   
## formula 3 formula call   
## trees 20 -none- list   
## weights 20 -none- numeric   
## votes 315 -none- numeric   
## prob 315 -none- numeric   
## class 105 -none- character  
## importance 4 -none- numeric   
## terms 3 terms call   
## call 6 -none- call

# Above we use the Adaboost algorithm. M-final is the number of times the boosting algorithm is run. Breiman  
# as the 'coeflearn' suggests that we are using the M1 algorithm proposed by Breiman. The M1 algorithm  
# is a classification algorithm where each class can attain a weight of no more than 1/2.  
adaboost$importance

## Petal.Length Petal.Width Sepal.Length Sepal.Width   
## 67.471117 26.002360 6.526522 0.000000

#This gives importance of the variables.  
errorevol(adaboost,test)

## $error  
## [1] 0.06666667 0.06666667 0.06666667 0.04444444 0.06666667 0.06666667  
## [7] 0.06666667 0.06666667 0.06666667 0.06666667 0.06666667 0.06666667  
## [13] 0.06666667 0.06666667 0.06666667 0.06666667 0.06666667 0.06666667  
## [19] 0.06666667 0.06666667  
##   
## attr(,"class")  
## [1] "errorevol"

#This gives the error of the variables  
predictions <- predict(adaboost,test)  
predictions$confusion

## Observed Class  
## Predicted Class setosa versicolor virginica  
## setosa 15 0 0  
## versicolor 0 13 1  
## virginica 0 2 14

predictions$error

## [1] 0.06666667

#Let's compare AdaBoost to RandomForest. To do this, I will quickly run randomforest on the iris dataset.  
rf\_iris <- randomForest(Species ~ ., data = train)  
rf\_iris

##   
## Call:  
## randomForest(formula = Species ~ ., data = train)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 2  
##   
## OOB estimate of error rate: 3.81%  
## Confusion matrix:  
## setosa versicolor virginica class.error  
## setosa 35 0 0 0.00000000  
## versicolor 0 34 1 0.02857143  
## virginica 0 3 32 0.08571429

yhat\_iris <- predict(rf\_iris, test)  
#Random forest Confusion Matrix:  
table(yhat\_iris, test$Species)

##   
## yhat\_iris setosa versicolor virginica  
## setosa 15 0 0  
## versicolor 0 13 1  
## virginica 0 2 14

#AdaBoost Confusion Matrix:  
predictions$confusion

## Observed Class  
## Predicted Class setosa versicolor virginica  
## setosa 15 0 0  
## versicolor 0 13 1  
## virginica 0 2 14

As we see from the final confusion matricies of the AdaBoost model and the Random Forest model, both models seem to work equally well. They also both do a good job of predicting classes. Moving forward, the use of either boosting (AdaBoost) or bagging (Random Forest) would matter more on your actual data set, what you are looking for as a result of the model; predictive or dimensionality reduction, and which method you are most comfortable with. To continue this research, I plan on using the caret package in R ‘caretEnsemble’ to try using independent ensemble modeling and combine Random Forest and AdaBoost together to build a final prediction. I will not cover that in this paper.

**4. Conclusion**

After going through the process of bootstrapping, bagging and boosting, the reader should have an understanding of the base ensemble methods that are used when describing ensemble modeling. The idea behind the modeling can depend on many ideas which were discussed in the section Ensemble Models. Ensemble methods are used widely and after a more thorough understanding of the broad usage and application of these methods, the reader should walk away comfortable with applying this methodology on their next data mining project. Not only do ensemble methods use the same learners as what would be used just using a single ensemble model, they add versatility to these learners. Either by accounting for bias and variance or by attacking residuals through an iterative process. One of the great things about ensemble methods is that although the actual computation level increases by the added repetition of model building, the statistical packages available to us today in R and other programs make this process a lot more manageable, and not too much more time consuming than simply building one learning algorithm.

My objective is to use ensemble methods for my next project, in my statistics class – Data Analysis as well as apply these methods when building models at work. I suggested ensemble methods to a colleague for our upcoming modeling project and this is the method we will be trying out. After reading multiple papers and articles, I am fully convinced that ensemble modeling is more beneficial than a simple baseline model. As stated by Giovvani Seni and John F. Elder, “Ensemble methods have been called the most influential development in Data Mining and Machine Learning in the past decade. They combine multiple models into one usually more accurate than the best of its components.” (Seni & Elder, 2010)

The few restraints would be time constraints on large data sets, but I have read on ways to parallelize R so that each model is running on a separate core (depending on how many cores your machine has, this may or may not be a fruitful solution). Another restraint would be model interpretability. At my current job, we will be using Amazon Web Services (AWS) so parallelizing R across the remote machine is definitely an option. The use of multiple models, stacking and combination techniques, will make the modeling process a lot more of a ‘black box’ and less presentable. However, predictive accuracy is increased, so the predictive vs. interpretability trade-off will have to be assessed to use ensemble methods. I am excited for the new doors ensemble methods and their applications have opened for me, as far as modeling technique.

# Bibliography

Bollen, K. A., & Stine, R. (1992). *Bootstrapping Goodness of Fit Measures in Structural Equation Models* (Vol. 21). Chapel-Hill: Sage Periodicals Press.

Breiman, L. (2001). Random Forests. *Machine Learning* *, 45* (1), 5-32.

Dietterich, T. (2000). Ensemble Methods in Machine Learning. *Multiple Classifier Systems Lecture Notes in Computer Science* , 1-15.

Hastie, T., Tibshirani, R., & Friedman, J. (2008). *The Elements of Statistical Learning Data Mining, Inference, and Prediction.* Stanford, CA: Springer.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning: with applications in R.* New York, New York: Springer.

Kerns, M., & Valiant, L. (1989). Crytographic limitations on learning Boolean formulae and finite automata. *Symposium on Theory of computing (ACM)* , 433-444.

Li, H.-T., & Li, L. (2008). Support Vector Machinery for Infinite Ensemble Learning. *Journal of Machine Learning Research 9* .

Rokach, L. (2010). *Pattern Classification Using Ensemble Methods.* Singapore: World Scientific.

Sammut, C., & Webb, G. (2011). *Encyclopedia of machine learning.* New York: Springer.

Seni, G., & Elder, J. (2010). Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions . *Synthesis Lectures on Data Mining and Knowledge Discovery* , 1-126.

Zhi-Hua, Z. (2012). *Ensemble methods: foundations and algorithms.* Boca Raton, FL: Chapman & Hall/CRC .