CSCI 4/5588: Machine Learning II
Chapter #6: Random Forests

(Ref: Book [1], [2], [3])

Objective: in this chapter, our target is to get an idea of Random Forest. We will also go through bootstrap methods, bagging, boosting, and decision tree to understand the theme.

Bootstrap Methods

Bootstrap is a general tool for assessing statistical accuracy. Here we describe the bootstrap in general.

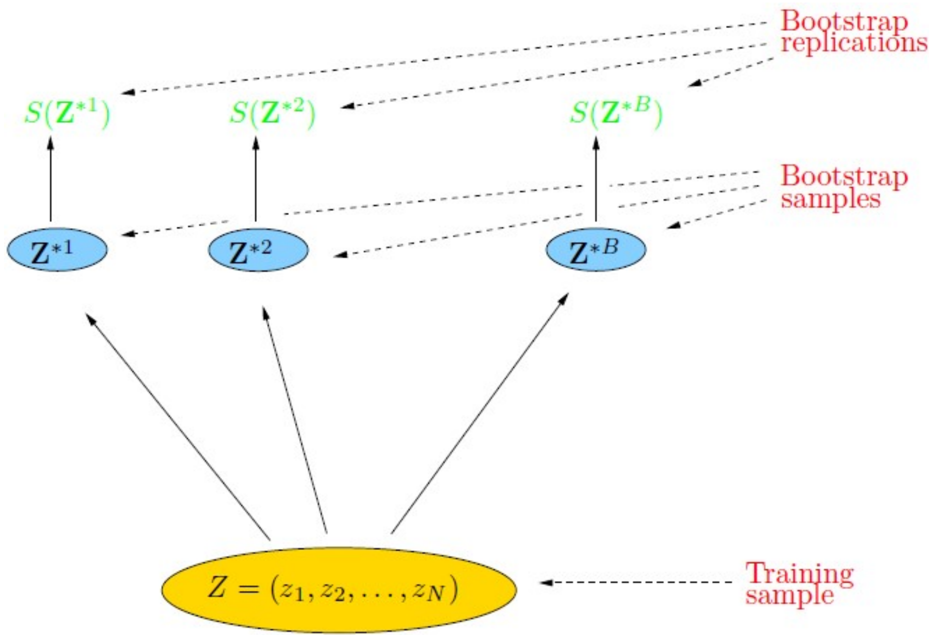


Figure 7.12: Schematic of the bootstrap process. We wish to assess the statistical accuracy of a quantity $S(Z)$ computed from our dataset. B training sets Z^{*b}, $b = 1, \ldots, B$ each of size N, are drawn with replacement from the original dataset. The quantity of interest $S(Z)$ is computed from each bootstrap training set, and the values $S(Z^{*1})$, …, $S(Z^{*B})$ are used to assess the statistical accuracy of $S(Z)$.

Suppose we have a model fit to a set of training data. We denote the training set by $Z = (z_1, z_2, \ldots, z_N)$ where $z_i = (x_i, y_i)$. The basic idea is to

(1) randomly draw datasets ***with replacement*** from the training data,

(2) this is done B times (B = 100 say), producing B bootstrap datasets, as shown in Figure 7.12.

(3) Then we refit the model to each of the bootstrap datasets and examine the fits over the B replications' behavior.

In the figure, $S(Z)$ is any quantity computed from the data Z, for example, the prediction at some input point. From the bootstrap sampling, we can estimate any aspect of the distribution (say, distribution function $\hat{F}$ for the data ($z_1, z_2, \ldots, z_N$)) of $S(Z)$, for example, its variance,

$$Var[S(Z)] = \frac{1}{B-1}\sum_{b=1}^{B}(S(Z^{*b}) - \bar{S}^*)^2 \tag{7.53}$$

where mean $\bar{S}^* = \sum_{b=1}^{B} S(Z^{*b})/B$.

The larger the number of B of bootstrap samples, the more satisfactory is the estimate of a statistic and its variance. One of the benefits of bootstrap estimation is that B can be adjusted to the computational resources.

How can we apply the bootstrap to estimate prediction error? One approach would be to fit the model in question on a set of bootstrap samples and then keep track of how well it predicts the original training set. If $\hat{f}^{*b}(x_i)$ is the predicted value at x_i, from the model fitted to the bth bootstrap dataset, our estimate is

$$Err_{boot} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N} L(y_i, \hat{f}^{*b}(x_i)) \tag{7.54}$$

Bagging

Bagging is basically *bootstrap aggregation*. Bootstrap has been shown to be a way of assessing the accuracy of a parameter estimate or a prediction. Here we show how to use the bootstrap to improve the estimate or prediction itself.

Consider first the regression problem. Suppose we fit a model to our training data $Z = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, obtaining the prediction $\hat{f}(x)$ at input x. *Bootstrap aggregation* or bagging averages this prediction over a collection of bootstrap samples, thereby reducing its variance. For each bootstrap sample Z^{*b}, b = 1, 2, …, B, we fit our model, giving a prediction $\hat{f}^{*b}(x)$. The bagging estimate is defined by

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x) \tag{8.51}$$

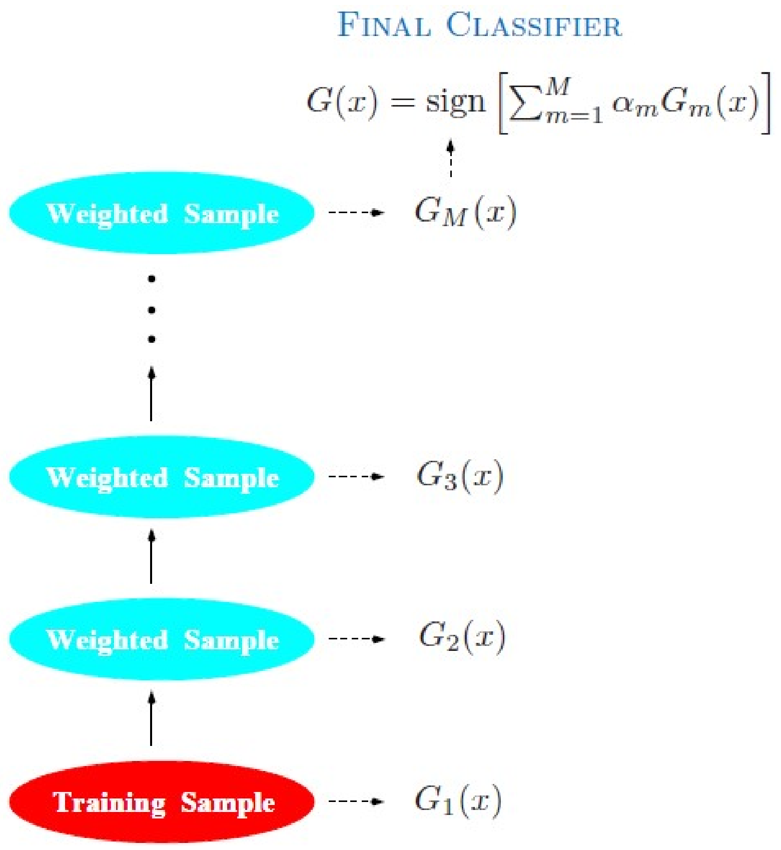


FIGURE 10.1. Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset and then combined to produce a final prediction.

Boosting Method

Boosting is one of the most powerful learning ideas introduced in the last twenty years. It was originally designed for classification problems, but it can be used for regression as well.

The motivation for boosting was a procedure that combines the outputs of many "weak" classifiers to produce a powerful "committee."

From this perspective boosting bears a resemblance to bagging and other committee-based approaches. However, we shall see the differences.

Here, we describe the most popular boosting algorithm called "AdaBoost.M1" (Freund and Schapire, 1997).

Consider a two-class problem, with the output variable coded as $Y \in \{-1, 1\}$. Given a vector of predictor variables *X*, a classifier *G*(*X*) produces a prediction taking one of the two values $\{-1, 1\}$. The error rate on the training sample is

$$\overline{err} = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq G(x_i)),$$

and the expected error rate on future predictions is $E_{XY} I(Y \neq G(X))$.

A weak classifier is one whose error rate is only slightly better than random guessing. 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 $G_m(x)$, $m = 1, 2, \ldots, M$.

The predictions from all of them are then combined through a weighted majority vote to produce the final prediction:

$$G(x) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right) \tag{10.1}$$

Here $\alpha_1, \alpha_2, \ldots, \alpha_M$ are computed by the boosting algorithm and weigh the contribution of each respective $G_m(x)$. Their effect is to give higher influence to the more accurate classifiers in the sequence. Figure 10.1 shows a schematic of the AdaBoost procedure.

Algorithm 10.1 AdaBoost.M1.

```
1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to $M$
    (a) Fit a classifier $G_m(x)$ to the training data using weights $w_i$.
    (b) Compute
        $err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$
    (c) Compute $\alpha_m = \log((1 - err_m)/err_m)$.
    (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1,2,\ldots,N$.
3. Output $G(x) = \text{sign}\,[\sum_{m=1}^{M} \alpha_m G_m(x)]$.
```

The data modifications at each boosting step consist of applying weights $w_1, w_2, \ldots, w_N$ to each of the training observations (x_i, y_i), $i = 1, 2,$

…, N. Initially, all of the weights are set to $w_i = 1/N$ so that the first step simply trains the classifier on the data in the usual manner. For each successive iteration $m = 2, 3, \ldots, M$ the observation weights are individually modified, and the classification algorithm is reapplied to the weighted observations. At step m, those observations that were misclassified by the classifier $G_{m-1}(x)$ induced at the previous step have their weights increased, whereas the weights are decreased for those that were classified correctly. Thus as iterations proceed, observations that are difficult to classify correctly receive ever-increasing influence. Each successive classifier is thereby forced to concentrate on those training observations that are missed by previous ones in the sequence.

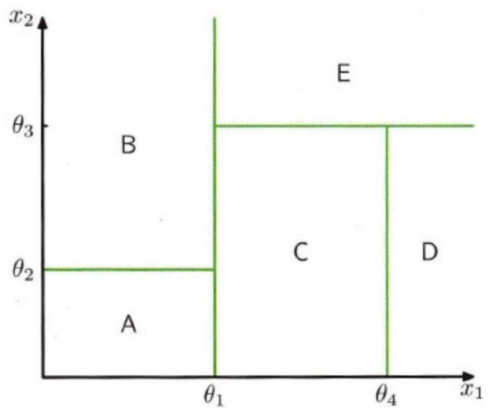
Algorithm 10.1 shows the details of the AdaBoost.M1 algorithm. The current classifier $G_m(x)$ is induced on the weighted observations in line 2a. The resulting weighted error rate is computed in line 2b. Line 2c calculates the weight α_m given to $G_m(x)$ in producing the final classifier $G(x)$ (line 3). The individual weights of each of the observations are updated for the next iteration at line 2d. Observations misclassified by $G_m(x)$ have their weights scaled by a factor $exp(\alpha_m)$, increasing their relative influence for inducing the next classifier $G_{m+1}(x)$ in the sequence.

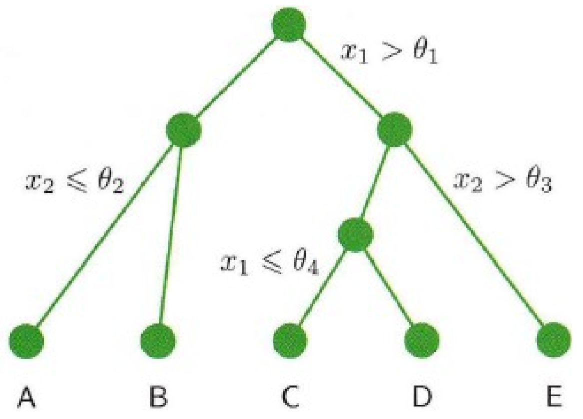
The AdaBoost.M1 algorithm is known as "Discrete AdaBoost" in Friedman et al. (2000) because the base classifier $G_m(x)$ returns a discrete class label. If the base classifier instead returns a real-valued prediction (e.g., a probability mapped to the interval $[-1, 1]$), AdaBoost can be modified appropriately (see "Real AdaBoost" in Friedman et al. (2000)).

The AdaBoost can even dramatically increase the performance of a very weak classifier. Since its introduction, much has been written to explain the success of AdaBoost in producing accurate classifiers. Most of this work has centered on using classification trees as the "base learner" G(x), where improvements are often most dramatic. In fact, Breiman (NIPS Workshop, 1996) referred to AdaBoost with trees as the "best off-the-shelf classifier in the world" (see also Breiman (1998)).

Decision Trees

There are various simple but widely used models that work by partitioning the input space into cuboid regions, whose edges are aligned with the axes, and then assigning a simple model (for example, a constant) to each region. They can be viewed as a model combination method in which only one model is responsible for making predictions at any given point in input space. The process of selecting a specific model, given a new input x, can be described by a sequential decision-making process corresponding to the traversal of a binary tree (one that splits into two branches at each node). Here we focus on a particular tree-based framework called **c**lassification **a**nd **r**egression **t**rees, or CART (Breiman *et al.*, 1984), although there are many other variants going by such names as ID3 and C4.5 (Quinlan, 1986; Quinlan, 1993).





(a) **(b)**

Figure 1(a): Illustration of a two-dimensional input space that has been partitioned into five regions using axis-aligned boundaries. **1(b)**: Binary tree corresponding to the partitioning of the input space shown in Figure 1(a).

Figures 1(a) and 1(b) show an illustration of a recursive binary partitioning of the input space, along with the corresponding tree structure. In this example, the first step divides the whole of the input space into two regions according to whether $x_1 \leq \theta_1$ or, $x_1 > \theta_1$ where θ_1 is a parameter of the model. This creates two sub-regions, each of which can then be subdivided independently. For instance, the region $x_1 \leq \theta_1$ is further subdivided according to whether $x_2 \leq \theta_2$ or, $x_2 > \theta_2$, giving rise to the regions denoted *A* and *B*. The recursive subdivision can be described by the traversal of the binary tree shown in Figure 1(b). For any new input *x*, we determine which region it falls into by starting at the top of the tree at the root node and following a path down to a specific leaf node according to the decision criteria at each node. Note that such decision trees are not probabilistic graphical models.

Within each region, there is a separate model to predict the target variable. For instance, in regression, we might simply predict a constant over each region, or in classification, we might assign each region to a specific class. A key property of tree-based models, which makes them popular in fields such as medical diagnosis, for example, is that they are readily interpretable by humans because they correspond to a sequence of binary decisions applied to the individual input variables. For instance, to predict a patient's disease, we might first ask, "is their temperature greater than some threshold?". If the answer is yes, then we might next ask, "is their blood pressure less than some threshold?" Each leaf of the tree is then associated with a specific diagnosis.

In order to learn such a model from a training set, we have to determine the structure of the tree, including which input variable is chosen at each node to form the split criterion as well as the value of the threshold parameter θ_i for the split. We also have to determine the values of the predictive variable within each region.

Consider first a regression problem in which the goal is to predict a single target variable $\boldsymbol{t}$ from a D-dimensional vector $\mathbf{x} = (x_1, \ldots, x_D)^T$ of input variables. The training data consists of input vectors $\{\mathbf{x}_1, ..., \mathbf{x}_N\}$ along with the corresponding continuous labels $\{t_1, \ldots, t_N\}$. If the partitioning of the input space is given, and we minimize the sum-of-squares error function, then the optimal value of the predictive variable within any given region is just given by the average of the values of t_n for those data points that fall in that region.

Now consider how to determine the structure of the decision tree. Even for a fixed number of nodes in the tree, the problem of determining the optimal structure (including choice of input variable for each split as well as the corresponding thresholds) to minimize the sum-of-squares error is usually computationally infeasible due to the combinatorially large number of possible solutions. Instead, a greedy optimization is generally done by starting with a single root node, corresponding to the whole input space, and then growing the tree by adding nodes one at a time. At each step, there will be some number of candidate regions in input space that can be split, corresponding to the addition of a pair of leaf nodes to the existing tree. For each of these, there is a choice of which of the D input variables to split, as well as the value of the threshold. The joint optimization of the choice of region to split, and the choice of input variable and threshold, can be done efficiently by exhaustive search, noting that, for a given choice of split variable and threshold, the optimal choice of the predictive variable is given by the local average of the data, as noted earlier. This is repeated for all possible choices of the variable to be split, and the one that gives the smallest residual sum-of-squares error is retained.

Given a greedy strategy for growing the tree, there remains the issue of when to stop adding nodes. A simple approach would be to stop when the reduction in residual error falls below some threshold. However, it is found empirically that often none of the available splits produces a significant reduction in error, and yet after several more splits, a substantial error reduction is found. For this reason, it is common practice to grow a large tree, using a stopping criterion based on the number of data points associated with the leaf nodes, and then prune back the resulting tree. The pruning is based on a criterion that balances residual error against a measure of model complexity. If we denote the starting tree for pruning by T_0, then we define $T \subset T_0$ to be a subtree of T_0 if it can be obtained by pruning nodes from T_0 (in other words, by collapsing internal nodes by combining the corresponding regions). Suppose the leaf nodes are indexed by $\tau = 1, \ldots, |T|$, with leaf node τ representing a region R_τ of input space having N_τ data points, and $|T|$ denoting the total number of leaf nodes. The optimal prediction for region R_τ is then given by

$$y_\tau = \frac{1}{N_\tau} \sum_{x_n \in R_\tau} t_n^{\ 2}$$

and the corresponding contribution to the residual sum-of-squares is then

$$Q_\tau(T) = \sum_{x_n \in R_\tau} \{t_n - y_\tau\}^2$$

The pruning criterion is then given by

$$C(T) = \sum_{\tau=1}^{|T|} Q_\tau(T) + \lambda \, | T |$$

The regularization parameter λ determines the trade-off between the overall residual sum-of-squares error and the complexity of the model as measured by the number $|T|$ of leaf nodes, and its value is chosen by cross-validation.

For classification problems, the process of growing and pruning the tree is similar, except that the sum-of-squares error is replaced by a more appropriate measure of performance. If we define $p_{\tau k}$ to be the proportion of data points in region R_τ assigned to class k, where $k = 1, \ldots, K$, then two commonly used choices are the negative cross-entropy:

$$Q_\tau(T) = \sum_{k=1}^{K} p_{\tau k} \ln p_{\tau k}$$

and the *Gini index*

$$Q_{\tau}(T) = \sum_{k=1}^{K} p_{\tau k}(1 - p_{\tau k})$$

These both vanish for $P_{\tau k} = 0$ and $P_{\tau k} = 1$ and have a maximum at $P_{\tau k} = 0.5$. They encourage the formation of regions in which a high proportion of the data points are assigned to one class. The cross-entropy and the Gini index are better measures than the misclassification rate for growing the tree because they are more sensitive to the node probabilities. Also, unlike the misclassification rate, they are differentiable and hence better suited to gradient-based optimization methods. For subsequent pruning of the tree, the misclassification rate is generally used.

Random Forest

We experienced that *bagging* or *bootstrap aggregation* is a technique for reducing the variance of an estimated prediction function. Bagging seems to work especially well for high-variance, low-bias procedures, such as trees. For regression, we simply fit the same regression tree many times to bootstrap-sampled versions of the training data and average the result. For classification, a committee of trees each cast a vote for the predicted class.

Boosting was initially proposed as a committee method as well, although unlike bagging, the committee of weak learners evolves over time, and the members cast a weighted vote. Boosting appears to dominate bagging on most problems and became the preferred choice.

Random forests (Breiman, 2001) is a substantial modification of bagging that builds a large collection of de-correlated trees and then averages them. On many problems, the performance of random forests is very similar to boosting, and they are simpler to train and tune. Therefore, random forests are popular, and are implemented in a variety of packages.

Defining of Random Forests: The essential idea in bagging is to average many noisy but approximately unbiased models and hence reduce the variance. Trees are ideal candidates for bagging since they can capture complex interaction structures in the data, and if grown sufficiently deep, have a relatively low bias. Since trees are notoriously noisy, they benefit greatly from the averaging. Moreover, since each tree generated in bagging is identically distributed (*i.d.*), the expectation of an average of B such trees is the same as the expectation of any one of them. This means the bias of bagged trees is the same as that of the individual trees, and the only hope of improvement is through variance reduction. This is in contrast to boosting, where the trees are grown in an adaptive way to remove bias and hence are not *i.d.*

Algorithm 15.1 *Random Forest for Regression or Classification.*

```
1. For b = 1 to B:
        (a) Draw a bootstrap sample $Z^*$ of size N from the training data.
        (b) Grow a random-forest tree $T_b$ to the bootstrapped data by recursively repeating
        the following steps for each terminal node of the tree until the minimum node size
        $n_{\min}$ is reached.
                i) Select m variables at random from the p variables.
                ii) Pick the best variable/split-point among them.
                iii) Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.
```

To make a new prediction at a new point *x*:

Regression: $\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the b^{th} random-forest tree.

Then $\hat{C}_{rf}^B(x) =$ majority vote $\{\hat{C}_b(x)\}_1^B$.

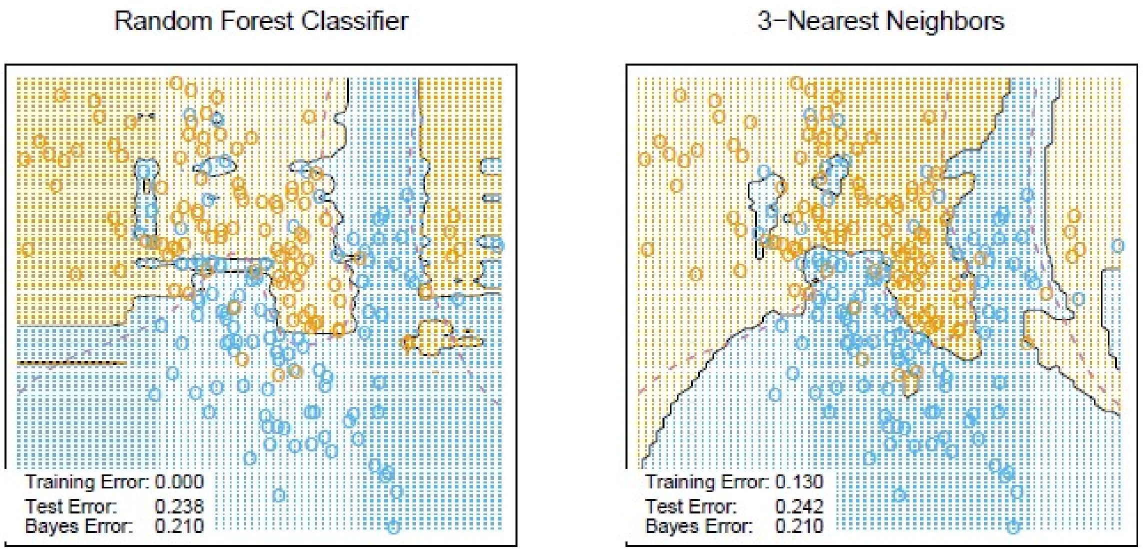


FIGURE 15.11. Random forests versus 3-NN on the mixture data. The axis-oriented nature of the individual trees in a random forest leads to decision regions with an axis-oriented flavor.

The idea in random forests (Algorithm 15.1) is to improve the variance reduction of bagging by reducing the correlation between the trees without increasing the variance too much.

A random-forest versus 3-Nearest Neighbor has been given next in Figure 15.11 for the mixture data.

Note: The Weka, http://www.cs.waikato.ac.nz/ml/weka/, offers a free java implementation of random forests.

Ensemble Learning

The idea of ensemble learning is to build a prediction model by combining the strengths of a collection of simpler base models. We have already seen a number of examples that fall into this category.

Bagging and random forests are ensemble methods for classification, where a *committee* of trees each cast a vote for the predicted class. Boosting in was initially proposed as a committee method as well, although unlike random forests, the committee of *weak learners* evolves over time, and the members cast a weighted vote. In fact, one could characterize any dictionary method, such as regression splines, as an ensemble method, with the basis functions serving the role of weak learners.

Bayesian methods for nonparametric regression can also be viewed as ensemble methods: a large number of candidate models are averaged with respect to the posterior distribution of their parameter settings (e.g. (Neal and Zhang, 2006)).

Ensemble learning can be broken down into two tasks:

(1) developing a population of base learners from the training data, and
(2) then combining them to form the composite predictor.

And the composite predictor is expected to perform better than the individual.

References:

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*: Springer, 2009.
[2] C. M. Bishop, *Pattern Recognition and Machine Learning*: Springer, 2009.
[3] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*: Wiley, 2000.

--- x ---