Chapter 12

Combining multiple learners

In general there are several algorithms for learning the same task. Though these are generally successful, no one single algorithm is always the most accurate. Now, we shall discuss models composed of multiple learners that complement each other so that by combining them, we attain higher accuracy.

12.1 Why combine many learners

There are several reasons why a single learner may not produce accurate results.

- Each learning algorithm carries with it a set of assumptions. This leads to error if the assumptions do not hold. We cannot be fully sure whether the assumptions are true in a particular situation.
- Learning is an ill-posed problem. With finite data, each algorithm may converge to a different solution and may fail in certain circumstances.
- The performance of a learner may be fine-tuned to get the highest possible accuracy on a validation set. But this fine-tuning is a complex task and still there are instances on which even the best learner is not accurate enough.
- It has been proved that there is no single learning algorithm that always produces the most accurate output.

12.2 Ways to achieve diversity

When many learning algorithms are combined, the individual algorithms in the collection are called the *base learners* of the collection.

When we generate multiple base-learners, we want them to be reasonably accurate but do not require them to be very accurate individually. The base-learners are not chosen for their accuracy, but for their simplicity. What we care for is the final accuracy when the base- learners are combined, rather than the accuracies of the bas-learners we started from.

There are several different ways for selecting the base learners.

1. **Use different learning algorithms**

   There may be several learning algorithms for performing a given task. For example, for classification, one may choose the naive Bayes' algorithm, or the decision tree algorithm or even the SVM algorithm.

   Different algorithms make different assumptions about the data and lead to different results. When we decide on a single algorithm, we give emphasis to a single method and ignore all others. Combining multiple learners based on multiple algorithms, we get better results.

2. **Use the same algorithm with different hyperparameters**

   In machine learning, a *hyperparameter* is a parameter whose value is set before the learning process begins. By contrast, the values of other parameters are derived via training.

   The number of layers, the number of nodes in each layer and the initial weights are all hyperparameters in an artificial neural network. When we train multiple base-learners with different hyperparameter values, we average over it and reduce variance, and therefore error.

3. **Use different representations of the input object**

   For example, in speech recognition, to recognize the uttered words, words may be represented by the acoustic input. Words can also be represented by video images of the speaker's lips as the words are spoken.

   Different representations make different characteristics explicit allowing better identification. In many applications, there are multiple sources of information, and it is desirable to use all of these data to extract more information and achieve higher accuracy in prediction. We make separate predictions based on different sources using separate base-learners, then combine their predictions.

4. **Use different training sets to train different base-learners**

   - This can be done by drawing random training sets from the given sample; this is called*bagging*.
   - The learners can be trained serially so that instances on which the preceding base-learners are not accurate are given more emphasis in training later base-learners; examples are *boosting* and *cascading*.
   - The partitioning of the training sample can also be done based on locality in the input space so that each base-learner is trained on instances in a certain local part of the input space.

5. **Multiexpert combination methods**

   These base learners work in parallel. All of them are trained and then given an instance, they all give their decisions, and a separate combiner computes the final decision using their predictions. Examples include voting and its variants.

6. **Multistage combination methods**

   These methods use a serial approach where the next base-learner is trained with or tested on only the instances where the previous base-learners are not accurate enough.

12.3 Model combination schemes

12.3.1 Voting

This is the simplest procedure for combining the outcomes of several learning algorithms. Let us examine some special cases of this scheme

1. **Binary classification problem**

   Consider a binary classification problem with class labels -1 and $+1$. Let there be L base learners and let x be a test instance. Each of the base learners will assign a class label to x. If the class label assigned is $+1$, we say that the learner votes for $+1$ and that the label $+1$ gets a vote. The number of votes obtained by the class labels when the different base learners are applied is counted. In the voting scheme for combining the learners, the label which gets the majority votes is assigned to x.

2. **Multi-class classification problem**

   Let there be n class labels $C_1, C_2, \ldots, C_n$. Let x be a test instance and let there be L base learners. Here also, each of the base learners will assign a class label to x and when a class label is assigned a label, the label gets a vote. In the voting scheme, the class label which gets the maximum number of votes is assigned to x.

3. **Regression**

   Consider L base learners for predicting the value of a variable y. Let $\hat{y}_i$ be the output predicted by the i-th base learner. The final output is computed as

   $$y = w_i\hat{y}_1 + w_2\hat{y}_2 + \cdots + w_L\hat{y}_L$$

   where $w_1, w_2, \ldots, w_L$ are called the weights attached to the outputs of the various base learners and they must satisfy the following conditions:

   $$w_j \geq 0 \text{ for } j = 1, 2, \ldots, L$$
   $$w_1 + w_2 + \cdots + w_L = 1.$$

   This is the *weighted voting scheme*. In *simple voting*, we take

   $$w_i = \frac{1}{L} \text{ for } j = 1, 2, \ldots, L.$$

12.3.2 Bagging

Bagging is a voting method whereby base-learners are made different by training them over slightly different training sets.

Generating L slightly different samples from a given sample is done by bootstrap, where given a training set X of size N, we draw N instances randomly from X with replacement (see Section **??**). Because sampling is done with replacement, it is possible that some instances are drawn more than once and that certain instances are not drawn at all. When this is done to generate L samples X_j, $j = 1, \ldots, L$, these samples are similar because they are all drawn from the same original sample, but they are also slightly different due to chance.

The base-learners are trained with these L samples X_j. A learning algorithm is an *unstable algorithm* if small changes in the training set causes a large difference in the generated learner. Bagging, short for bootstrap aggregating, uses bootstrap to generate L training sets, trains L base-learners using an unstable learning procedure and then during testing, takes an average. Bagging can be used both for classification and regression. In the case of regression, to be more robust, one can take the median instead of the average when combining predictions.

Algorithms such as decision trees and multilayer perceptrons are unstable.

12.3.3 Boosting

In bagging, generating complementary base-learners is left to chance and to the unstability of the learning method. In boosting, we actively try to generate complementary base-learners by training the next learner on the mistakes of the previous learners. The original boosting algorithm combines three weak learners to generate a strong learner. A weak learner has error probability less than 1/2, which makes it better than random guessing on a two-class problem, and a strong learner has arbitrarily small error probability.

The boosting method

1. Let d_1, d_2, d_3 be three learning algorithms for a particular task. Let a large training set X be given.

2. We randomly divide X into three sets, say X_1, X_2, X_3.

3. We use X_1 and train d_1.
4. We then take X_2 and feed it to d_1.
5. We take all instances misclassified by d_1 and also as many instances on which d_1 is correct from X_2, and these together form the training set of d_2.
6. We then take X_3 and feed it to d_1 and d_2.
7. The instances on which d_1 and d_2 disagree form the training set of d_3.
8. During testing, given an instance, we give it to d_1 and d_2 if they agree, that is the response; otherwise the response of d_3 is taken as the output.

It has been shown that this overall system has reduced error rate, and the error rate can arbitrarily be reduced by using such systems recursively. One disadvantage of the system is thaaaaaat it requires a very large training sample. An improved algorithm known as AdaBoost (short for "adaptive boosting"), uses the same training set over and over and thus need not be large. AdaBoost can also combine an arbitrary number of base-learners, not three.

12.4 Ensemble learning*

The word "ensemble" literally means "a group of things or people acting or taken together as a whole, especially a group of musicians who regularly play together."

In machine learning, an ensemble learning method consists of the following two steps:

1. Create different models for solving a particular problem using a given data.
2. Combine the models created to produce improved results.

The different models may be chosen in many different ways:

- The models may be created using appropriate different algorithms like k-NN algorithm, Naive-Bayes algorithm, decision tree algorithm, etc.
- The models may be created by using the same algorithm but using different splits of the same dataset into training data and test data.
- The models may be created by assigning different initial values to the parameters in the algorithm as in ANN algorithms.

The models created in the ensemble learning methods are combined in several ways.

- Simple majority voting in classification problems: Every model makes a prediction (votes) for each test instance and the final output prediction is the one that receives more than half of the votes.
- Weighted majority voting in classification problem: In weighted voting we count the prediction of the better models multiple times. Finding a reasonable set of weights is up to us.
- Simple averaging in prediction problems: In simple averaging method, for every instance of test dataset, the average predictions are calculated.
- Weighted averaging in prediction problems: In this method, the prediction of each model is multiplied by the weight and then their average is calculated.

12.5 Random forest*

A *random forest* is an ensemble learning method where multiple decision trees are constructed and then they are merged to get a more accurate prediction.

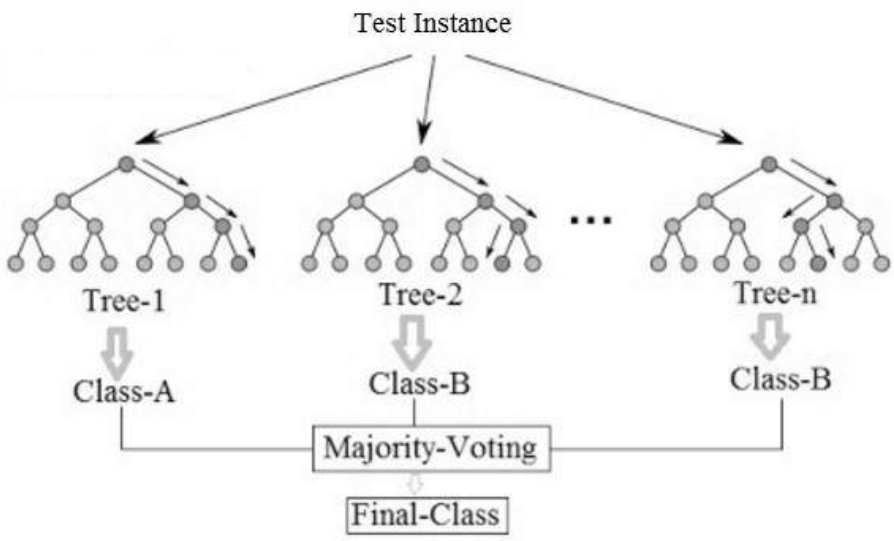

Figure 12.1: Example of random forest with majority voting

12.5.1 Algorithm

Here is an outline of the random forest algorithm.

1. The random forests algorithm generates many classification trees. Each tree is generated as follows:
   (a) If the number of examples in the training set is N, take a sample of N examples at random - but with replacement, from the original data. This sample will be the training set for generating the tree.
   (b) If there are M input variables, a number m is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the generation of the various trees in the forest.
   (c) Each tree is grown to the largest extent possible.
2. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification

12.5.2 Strengths and weaknesses

Strengths

The following are some of the important strengths of random forests.

- It runs efficiently on large data bases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- Generated forests can be saved for future use on other data.

- Prototypes are computed that give information about the relation between the variables and the classification.
- The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.
- It offers an experimental method for detecting variable interactions.
- Random forest run times are quite fast, and they are able to deal with unbalanced and missing data.
- They can handle binary features, categorical features, numerical features without any need for scaling.
- There are lots of excellent, free, and open-source implementations of the random forest algorithm. We can find a good implementation in almost all major ML libraries and toolkits.

Weaknesses

- A weakness of random forest algorithms is that when used for regression they cannot predict beyond the range in the training data, and that they may over-fit data sets that are particularly noisy.
- The sizes of the models created by random forests may be very large. It may take hundreds of megabytes of memory and may be slow to evaluate.
- Random forest models are black boxes that are very hard to interpret.

12.6 Sample questions

(a) Short answer questions

1. Explain the necessity of combining several algorithms for accomplishing a particular task.
2. What is a base learner? How do we select base learners?

(b) Long answer questions

1. Explain the following: (i) voting (ii) bagging (iii) boosting.
2. Explain what is meant by random forests.