CS 534 – Implementation Assignment 3

Due 11:59pm, November 12th, 2016

---

1 Group members

- Chunxiao Wang (Contribution: 50%)
- Miao Yang (Contribution: 50%)

2 Decision Tree

2.1 Algorithm

We implement a decision tree for the iris data, in which we have 120 training examples and 30 test examples. There are 4 features an one class:

- x_1: sepal length in cm
- x_2: sepal width in cm
- x_3: petal length in cm
- x_4: petal width in cm
- y: species. There are three classes: setosa (Class 0), versicolour (Class 1), virginica (Class 2).

We use information gain as a criterion, which measures how well a given attribute separates the training examples according to their targeting classification. Since our features are all continuous, we will use different threshold as different tree nodes, and pick the nodes with the largest information gain.

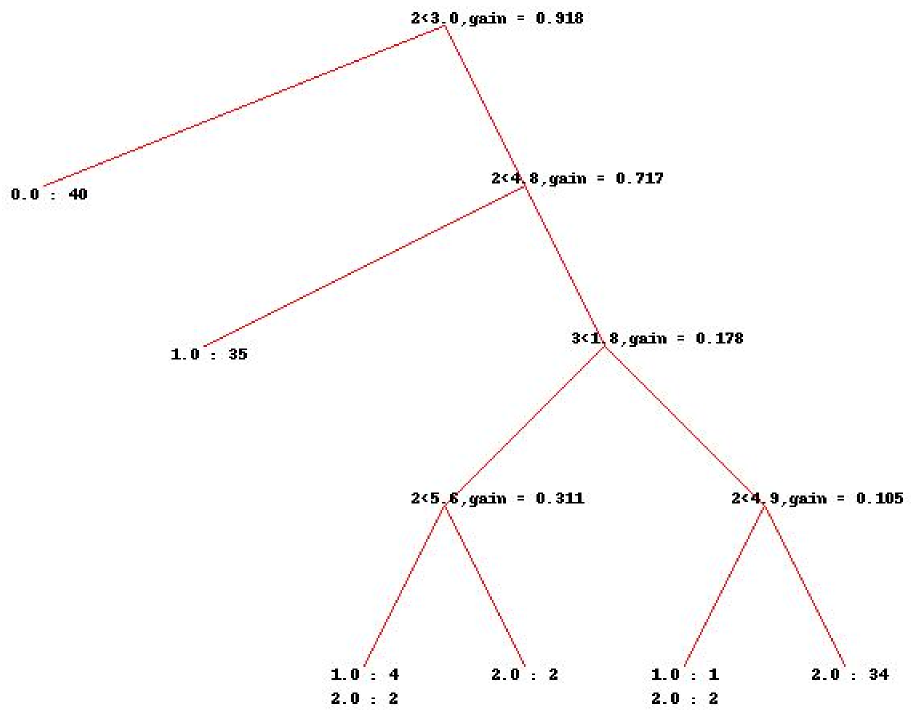
The stopping criteria here is the number of instances at leave nodes. We choose an integer k, and if the number of examples at a node is less than k, we must stop and turn it into a leave node. Therefore in some situations, a leave node may contain more than one classes, in which we will choose the class with the most training examples.

2.2 Decision tree with fixed $k = 2$

We implement the above algorithm, and set $k = 2$, i.e., the maximum number of examples a leaf can have is 2. And the tree is given in Figure 1.

In the tree, each root node is given as the form of "$i < z, gain = IG$", i.e., an integer i smaller than some number z, followed by a number IG denoting the information gain. The integer i denotes the feature x_{i+1}, i.e., $0, 1, 2, 3$ correspond to x_1, x_2, x_3, x_4, respectively. And the number z is the selected threshold.

Figure 1: The decision tree when $k = 2$



Each leaf node also contains information. Those are the numbers in each classification. For example,

$$1.0 : 4$$

$$2.0 : 2$$

means there are 4 training examples from Class 1, and 2 from Class 2. Since there are more Class 1 examples than the other 2 classes, based on our algorithm, we will classify this as Class 1.

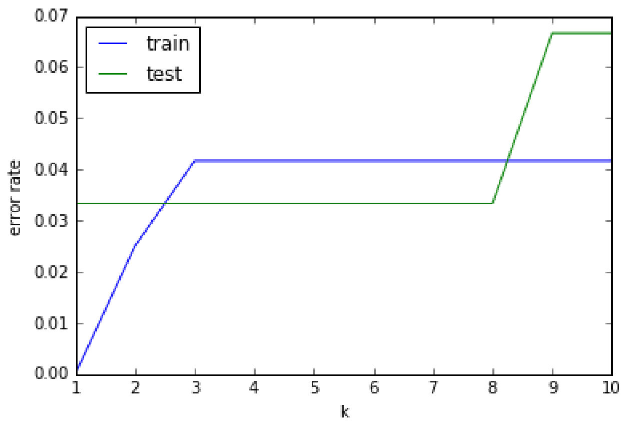
2.3 Decision tree with different k

We use different values of k, and implement the similar trees as the one above. For each tree, we calculate the training and testing errors. Figure 2 illustrates how errors change when k changes.

As we can see from the figure, as k increases, the training error increases. This is because:

- The smaller number (k) of examples each leaf has, the less likely it will contain more than 1 class from the training example, leading to smaller training error. i.e., small k will fit the decision tree pretty well for the training data, which is called overfitting. Consider an extreme case $k = 1$, each example can be categorized into a leaf, the training error would be 0.
- For large number of k, the tree will have fewer nodes and leaves, therefore the prediction errors for training set would increase. Consider another extreme case with $k = 120$, we will

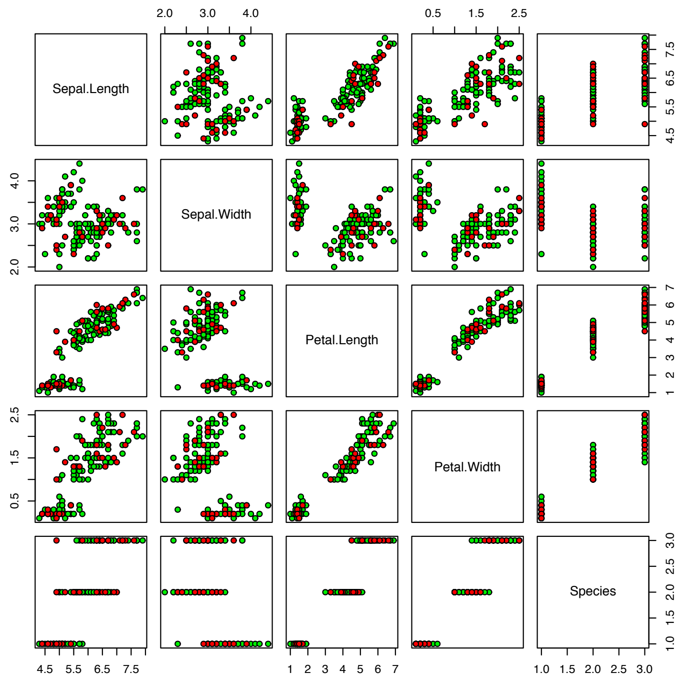
Figure 2: Error rate vs k



class all training set into one class, therefore the training error would be very large due to this bad-behaved tree.

For the testing error, it's also increasing with k increasing. We have plotted the training and testing data together, see Figure 3.

Figure 3: Plot of training (green) and testing (red) data together



From the figure, we can see that the testing data are surrounded by the training data, meaning they are pretty close to each other. That's why these two prediction errors behave very similarly.

Another interesting trend here is: if we look at the absolute difference of error rates between training and testing data sets, it first decreases and then increases with k. The reason for that is we have an overfitted tree when k is small, therefore the training error tends to be small, but testing error is relatively large; with a large k, the tree won't fit the training data well, thus both prediction errors would be higher, but the testing error would be comparatively larger since the tree was obtained via the training data.

3 Random Forest

3.1 Algorithm

We modified the learning algorithm in Section 2 as follows:

- **Feature bagging:** 2 random features out of 4 are selected at each candidate split.
- **Bootstrap aggregating:** each time when we built a tree, instead of using the whole training data, we draw a bootstrap resample from the training set, i.e., draw a resample with replacement from training, and the resample has the same size with the training set.
- **Majority voting**: after fitting L such decision trees, the predicting for any sample is the mode of the classes predicted by L trees.

This modified algorithm is called random forest.

3.2 Random Forest

We choose different values of L in $(5, 10, 15, 20, 25, 30)$. For each L, we still use different k, the number of instances at leave nodes. After using the modified algorithm, we can fit a random forest for each L, k combination. Apply the fitted forest to predict the training and testing data set, and the error rate for difference cases are shown in Figure 4 and 5.

From the figures, we can see that the general trend is as k increases, both prediction errors increase, which is consistent with our observations in Section 2. However as L increases, the error is expected to be decreasing. From the figure, there does seem to exist a decreasing trend of error vs L, but the trend is not so obvious. The reason for that is we only have 10 random runs, which is a very small proportion of all the possible bootstrap resamples. Therefore the behavior of these 10 random runs doesn't show an exact decreasing trend of error vs L. We believe if we add more runs, this pattern will become more obvious.

Figure 4: Training error rate vs *k*

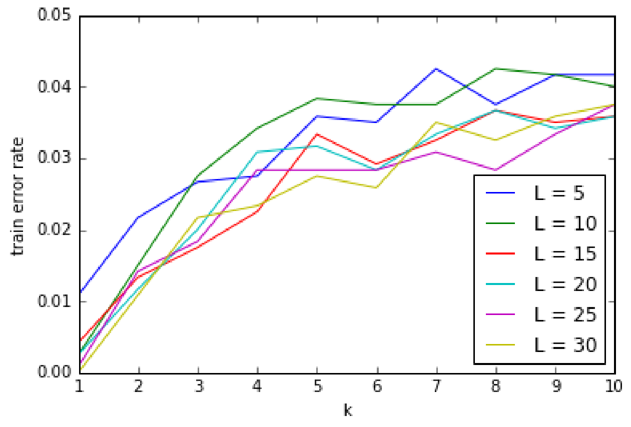
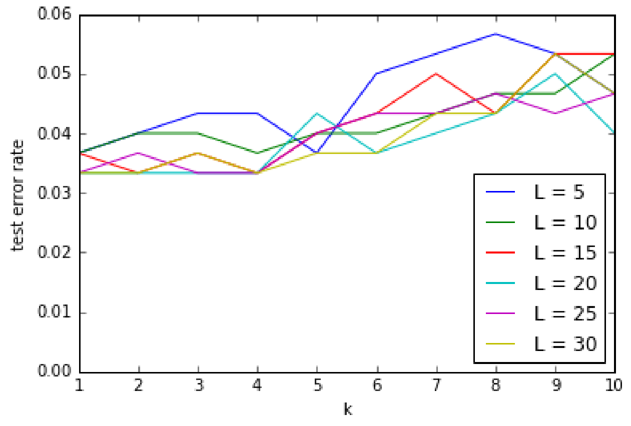


Figure 5: Testing error rate vs *k*


test error rate
k
L = 5
L = 10
L = 15
L = 20
L = 25
L = 30