Random Forest

Chaoqun Huang ch2958

1 General Description

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.[1]

2 Random Forest

2.1 mathematical formula

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, ..., x_n$ with responses $Y = y_1, ..., y_n$, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:
For b = 1, ..., B:

1. Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2. Train a classification or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x')$$

or by taking the majority vote in the case of classification trees.
This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on x':

$$\sigma = \sqrt{\frac{\sum_{b=1}^{B}(f_b(x') - \hat{f})^2}{B-1}}.$$

The above procedure describes the original bagging algorithm for trees. Random forests differ in only one way from this general scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated. An analysis of how bagging and random subspace projection contribute to accuracy gains under different conditions is given by Ho. Typically, for a classification problem with p features, √p (rounded down) features are used in each split. For regression problems the inventors recommend p/3(rounded down) with a minimum node size of 5 as the default.[1]

That was about Decision Tree, but it also applies for Random Forest. The difference is that for Random Forest we use Bootstrap Aggregation. It has no model underneath, and the only assumption that it relies is that sampling is representative. But this is usually a common assumption. For example, if one class consist of two components and in our dataset one component is represented by 100 samples, and another component is represented by 1 sample - probably most individual decision trees will see only the first component and Random Forest will misclassify the second one.[1]

2.2 Discriminative Algorithm

Random Forest Algorithm is a **discriminative** classification algorithm, because when building the tree, it learn the decision boundary by recursively partitioning the space in a manner that maximizes the information gain.[2]

2.3 Assumptions

The only assumption Random Forest relies on is that the sampling is representative.

2.4 Dependent & Independent Variable

There are no restrictions on both dependent and independent variable in random forest. They can be categorical and/or numerical, scale, variance.

3 Dataset

The titanic survival data set is used to do classification. The columns used are shown below:

Name	Description	Value
Survived	Whether the passenger survived or not	0 or 1
Pclass	Ticket class	1 - 3
Sex	Gender of the passenger	Male or Female
Age	Age of the passenger	Numeric
Fare	Passenger Fare	Numeric

```
> summary(dataset)
    Survived          Pclass          Sex                 Age              Fare
 Min.   :0.0000   Min.   :1.000   Length:714         Min.   : 0.42   Min.   :  0.00
 1st Qu.:0.0000   1st Qu.:1.000   Class :character   1st Qu.:20.12   1st Qu.:  8.05
 Median :0.0000   Median :2.000   Mode  :character   Median :28.00   Median : 15.74
 Mean   :0.4062   Mean   :2.237                      Mean   :29.70   Mean   : 34.69
 3rd Qu.:1.0000   3rd Qu.:3.000                      3rd Qu.:38.00   3rd Qu.: 33.38
 Max.   :1.0000   Max.   :3.000                      Max.   :80.00   Max.   :512.33
```

Figure 3-1 Summary of the titanic datatset

4 Run in SparkML

4.1 Split Train and Test Dataset

In order to compare the results from both R and spark. A r script is used to first generate the training set and testing set from titanic-train.csv provided in NYU Classes. The code is shown below:

```
data <-
read.csv('http://christianherta.de/lehre/dataScience/machineLearning/data/titanic-train.csv',header=T,strings
AsFactors = F)
keeps <- c("Survived", "Pclass", "Sex","Age","Fare")
dataset <- data[keeps]
dataset<-dataset[complete.cases(dataset),]
sex <- ifelse(dataset$Sex=="male", 1, 0)
dataset[,"Sex"] <- sex
set.seed(12345)
```

```
# Set Seed so that same sample can be reproduced in future also
#Now Selecting 80% of data as sample from total 'n' rows of the data
sample <-sample(714,156,replace=FALSE)
train.dataset <- dataset[-sample,]
test.dataset <- dataset[sample,]
# Save train and test dataset into csv files for spark
write.csv(test.dataset, file="split-test.csv")
write.csv(train.dataset, file="split-train.csv")
```

Code 4-1 Split Train and Test Dataset

80% of the data is used to train the random forest model and 20% is used to test the results.

4.2 Read and Convert csv file to JavaRDD

In order to classify the data set, the first step is to read the csv file into java RDD. The following code shows reading train and test csv file into JavaRDD.

```
SparkConf sparkConf = new SparkConf().setAppName("RandomForestTitanic").setMaster("local[1]");

    JavaSparkContext jsc = new JavaSparkContext(sparkConf);

    JavaRDD<String> testData = jsc.textFile("split-test.csv");
    JavaRDD<String> trainData = jsc.textFile("split-train.csv");
    JavaRDD<LabeledPoint> trainPoints = trainData.map(line -> {
       String[] params = line.split(COMMA_DELIMITER);
       double label = Double.valueOf(params[0]);
       double[] vector = new double[4];
       vector[0] = Double.valueOf(params[1]);
       vector[1] = Double.valueOf(params[2]);
       vector[2] = Double.valueOf(params[3]);
       vector[3] = Double.valueOf(params[4]);
       return new LabeledPoint(label, new DenseVector(vector));
    });

    JavaRDD<LabeledPoint> testPoints = testData.map(line -> {
       String[] params = line.split(COMMA_DELIMITER);
       double label = Double.valueOf(params[0]);
       double[] vector = new double[4];
       vector[0] = Double.valueOf(params[1]);
       vector[1] = Double.valueOf(params[2]);
       vector[2] = Double.valueOf(params[3]);
       vector[3] = Double.valueOf(params[4]);
       return new LabeledPoint(label, new DenseVector(vector));
    });
```

Code 4-2 Read csv into JavaRDD

4.3 Train the Random Forest Model

Use java code from spark ml tutorial to train the random forest model. In order to get a desired result. The model is used the recommended parameters provided by Spark tutorial[4]. The configuration parameters in shown below.

Key	Value
Number of Classes	2
Number of Trees	5
Impurity	gini
Random Seed	12345
Max Depth of Each Tree	4

```
// Train a RandomForest model
// Empty categoricalFeaturesInfo indicates all features are continuous
Integer numClasses = 2;
Map<Integer, Integer> categoricalFeaturesInfo = new HashMap<>();
Integer numTrees = 5; // Use more in practice.
String featureSubsetStrategy = "auto"; // Let the algorithm choose.
String impurity = "gini";
Integer maxDepth = 4;
Integer maxBins = 32;
Integer seed = 12345;

RandomForestModel model = RandomForest.trainClassifier(trainPoints, numClasses,
        categoricalFeaturesInfo, numTrees, featureSubsetStrategy, impurity, maxDepth, maxBins,
        seed);
```

Code 4-3 Train the Random Forest Model[4]

4.4 Analysis The Results

Calculate Test Error:

```
// Evaluate model on test instances and compute test error
JavaPairRDD<Object, Object> predictionAndLabel =
        testPoints.mapToPair(p -> new Tuple2<>(model.predict(p.features()), p.label()));
double testErr =
        predictionAndLabel.filter(pl -> !(pl._1()).equals(pl._2())).count() / (double) testData.count();
System.out.println("Test Error: " + testErr);
//System.out.println("Learned classification forest model:\n" + model.toDebugString());
```

Code 4-4 Calculate Test Error

Calculate Confusion Matrix:

```
long truePositive = predictionAndLabel.filter(pl -> (pl._1()).equals(pl._2()) && pl._1().equals(1d)).count();
System.out.println("True Positive: " + truePositive);

long trueNegative = predictionAndLabel.filter(pl -> (pl._1()).equals(pl._2()) && pl._1().equals(0d)).count();
System.out.println("True Negative: " + trueNegative);
```

```
long falsePositive = predictionAndLabel.filter(pl -> !(pl._1()).equals(pl._2()) && pl._2().equals(0d)).count();
System.out.println("False Positive: " + falsePositive);

long falseNegative = predictionAndLabel.filter(pl -> !(pl._1()).equals(pl._2()) && pl._2().equals(1d)).count();
System.out.println("False Negative:" + falseNegative);

System.out.println("\nConfusion Matrix:");
System.out.println("n: " + testData.count() + "   0" + "   1");
System.out.println("0: " + "   " + trueNegative + "   " + falseNegative);
System.out.println("1: " + "   " + falsePositive + "   " + truePositive + "\n");
```

Code 4-5 Calculate Confusion Matrix

Calculate AUC[5]:

```
BinaryClassificationMetrics metrics =
      new BinaryClassificationMetrics(predictionAndLabel.rdd());
// AUPRC
System.out.println("Area under precision-recall curve = " + metrics.areaUnderPR());

// AUROC
System.out.println("Area under ROC = " + metrics.areaUnderROC());
```

Code 4-6 Calculate AUC number[5]

The results:

```
[ch2958@F4Linux1 bin]$ ./spark-submit --master local[*] --class RandomForestTitanic ~/Rand
omForestTitanic-1.0.jar
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
total train data:558
total test:156
Test Error: 0.1858974358974359
True Positive: 31
True Negative: 96
False Positive: 11
False Negative:18

Confusion Matrix:
n: 156   0   1
0:    96   18
1:    11   31

Area under precision-recall curve = 0.7430664573521717
Area under ROC = 0.7649246614533665
[ch2958@F4Linux1 bin]$
```

5 Run in R

5.1 Read csv into R and Train the Model

```
library(class)
library(randomForest)
library(ROCR)
library(caret)
set.seed(12345)
train.dataset <- read.csv('~/projects/bigdata/split-train.csv', header=T)
test.dataset <- read.csv('~/projects/bigdata/split-test.csv', header=T)
fit <- randomForest(as.factor(Survived) ~ Pclass + Sex + Age + Fare, data=train.dataset, importance=TRUE,
ntree=5, maxnodes=16, nodesize=16)
```

Code 5-1 Train Model in R

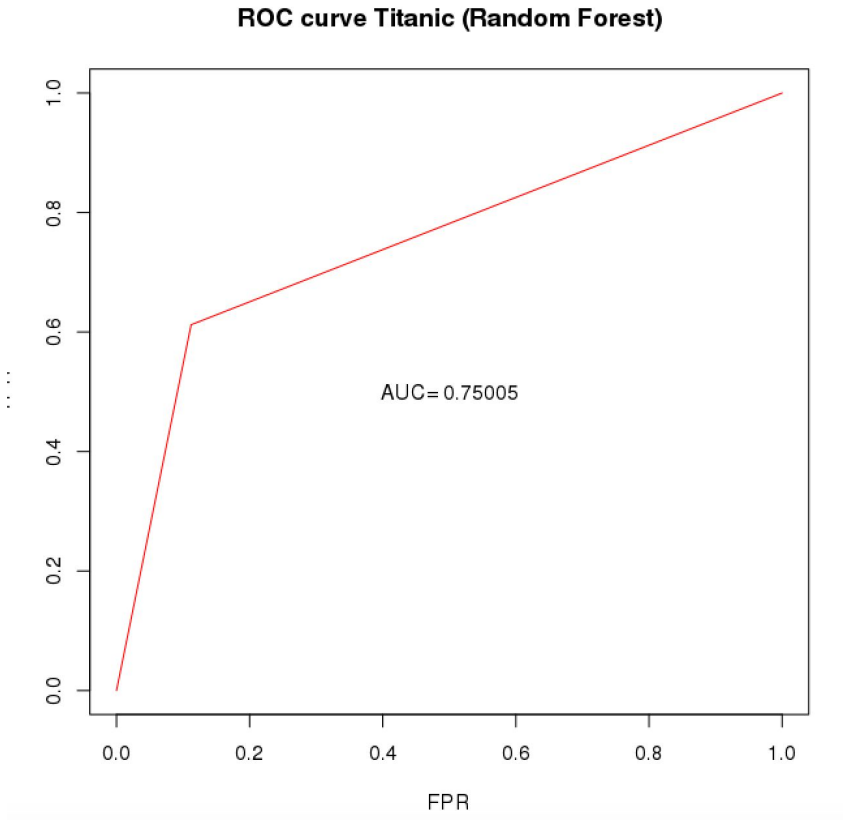
5.2 Analysis The Results

```
Prediction <- predict(fit, test.dataset)

Prediction <- as.numeric(levels(Prediction)[as.integer(Prediction)])
rf_auc_1<-prediction(as.numeric(Prediction),test.dataset$Survived)
rf_prf<-performance(rf_auc_1, measure="tpr", x.measure="fpr")
rf_slot_fp<-slot(rf_auc_1,"fp")
rf_slot_tp<-slot(rf_auc_1,"tp")

rf_t <- table(test.dataset$Survived, Prediction)

# Model Accuracy
testError <- 1- (rf_t[1, 1] + rf_t[2, 2]) / sum(rf_t)

# Print accuracy
cat("Test Error: ", testError)

rf_perf_AUC=performance(rf_auc_1,"auc")
rf_AUC=rf_perf_AUC@y.values[[1]]
cat("AUC: ", rf_AUC)

confusionMatrix <- confusionMatrix(as.numeric(Prediction),test.dataset$Survived)
print(confusionMatrix)
X11()
plot(rf_prf, col=rainbow(7), main="ROC curve Titanic (Random Forest)",
    xlab="FPR", ylab="TPR")
text(0.5,0.5,paste("AUC=",format(rf_perf_AUC@y.values[[1]],digits=5, scientific=FALSE)))
```

Code 5-2 Analysis the results

The results:

```
Test Error:  0.1987179
AUC:  0.7500477
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 95 19
         1 12 30

               Accuracy : 0.8013
                 95% CI : (0.73, 0.8608)
    No Information Rate : 0.6859
    P-Value [Acc > NIR] : 0.0008722

                  Kappa : 0.5202
 Mcnemar's Test P-Value : 0.2811981

            Sensitivity : 0.8879
            Specificity : 0.6122
         Pos Pred Value : 0.8333
         Neg Pred Value : 0.7143
             Prevalence : 0.6859
         Detection Rate : 0.6090
   Detection Prevalence : 0.7308
      Balanced Accuracy : 0.7500

       'Positive' Class : 0

> 
```



6 Comparing Results

Accuracy & TestError & AUC Compare:

	SparkML	**R**
Accuracy	81.41%	80.13%
TestError	0.1859	0.1987
AUC (under ROC)	0.76	0.75

Confusion Matrix Compare:

SparkML	0	1	**R**	0	1
0	96	18	0	95	19
1	11	31	1	12	30

7 Reference

[1] https://en.wikipedia.org/wiki/Random_forest
[2] https://stats.stackexchange.com/questions/12421/generative-vs-discriminative
[3] https://en.wikipedia.org/wiki/Discriminative_model
[4] https://spark.apache.org/docs/2.2.0/ml-classification-regression.html
[5] https://spark.apache.org/docs/2.2.0/mllib-evaluation-metrics.html