PROJECT REPORT

Type of model used

We used Random Forest model to make predictions in our project.
Basically, a random forest is a collection of decision trees. Each decision tree is trained using a random sample of the dataset. Then, a prediction is made from the entire forest by a majority vote. So each tree's prediction is counted as a vote for one class. The label is predicted to be the class which receives the most votes.

How did changes of parameters controlling partitioning affect the running time?

A random forest has parameters that can affect performance:

- **Number of trees**: RF can automatically train trees until performance is maximized, or the user can specify a number of trees. Increasing the number of trees in a random forest did not result in overfitting.
  We increased the number of trees to 50 from 25(keeping the other parameters constant) and we observed that the accuracy fell down to 99.76% from 99.66%. This is due to the fact that it was overfitting the data and hence 10 was an ideal number for my algorithm. The running time increased from 20 minutes to 45 minutes
- **Maximum depth of tree**: Higher values increased the quality of the prediction, but lead to overfitting and hence less accuracy on the validation data. Also high values also increased the training and prediction time of our model.

  We increased the depth of the tree from 4 to 8 but it took me 20 more minutes(keeping the other parameters constant), but the accuracy was generally about the same hovering around 99.6%.. Hence to reduce training time we kept the depth as 4 as it was generally giving me the same accuracy with less training time.

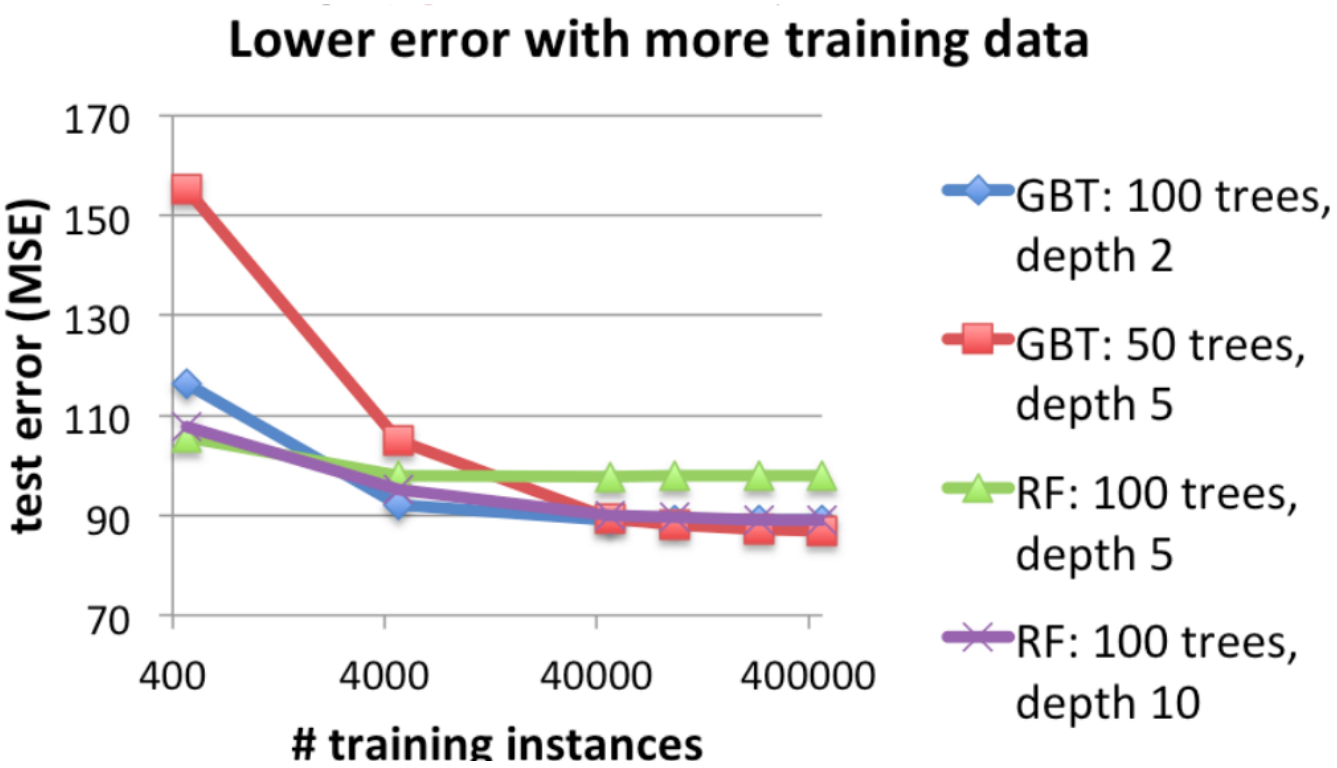
- **Impurity**: Studies have shown that the choice of impurity measure has little effect on the performance of decision tree classification algorithms. This is because many impurity measures are quite consistent with each other.

  We used Gini impurity than entropy, as it doesn't require to compute logarithmic functions (which is calculated in entropy), which are computationally intensive.

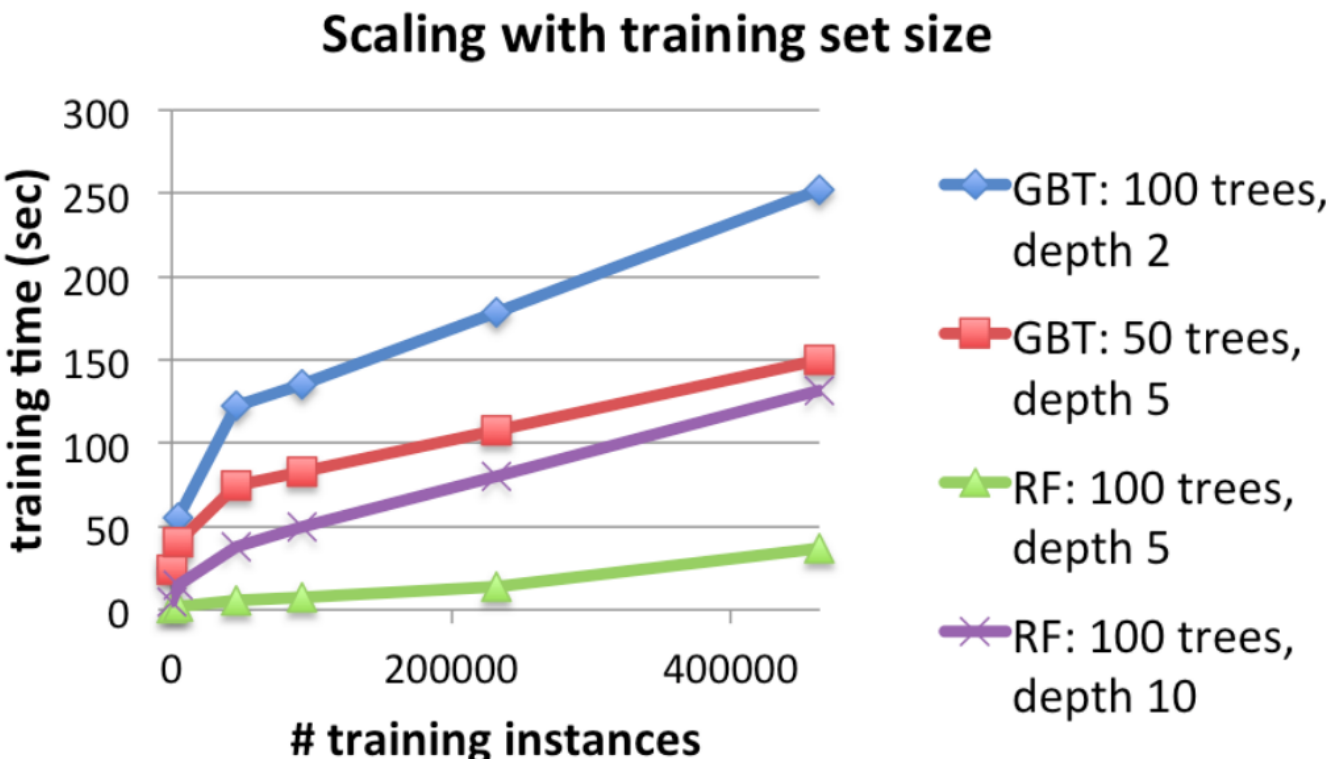
- **CategoricalFeaturesInfo:**We chose continuous feature as CategoricalFeaturesinfo because it does not require any significant preprocessing

unlike the categorical features. Also we require all the neighborhood brightness values of the pixel to determine the brightness value of the pixel. Specifying a subset of those as categorical attributes didn't make any sense to us because we don' t know for sure whether the subset of the neighborhood values would give us the same prediction than if we had the entire neighborhood of pixels.

ANALYSIS OF OUR MODEL



As this graph points it out, as we generally increase the depth of the tree with the number of training instance our test error rate decreases. This is due to the fact that the more number of training records we have, we have more data to split on and hence depth is increased to combat with the increase the accuracy of training records. While 400 records don't need to have many split points so we don't want to have large depth for a small number of records as it might overfit the dataset and would give us less accuracy
.



As we see in the graph, increasing the size of training dataset increases the running time. And also trees of greater depth will also take considerable training time. GBT's take more time to train than RF because they train only one tree at a time.

If you performed any preprocessing, briefly summarize what you did and why you did it?

We sampled out only 0.1% of the number of zeroes in the training data because it might overfit the data due to the numbers imbalance between the number of ones and zeroes.

Random Forest Classifier PseudoCode

class RandomClassifier()

1) we first make a pairRDD(label,features) of given dataset. No shuffling happens here as it just maps the input to each partition of the dataset. This required 482 tasks.

val rdd_data(label,features) = make pairRDD of the given datset

2) we filtered out all the ones and zeroes from rdd_data. This does not involve shuffling and it included 964 tasks.

3) we randomly split the number of zeroes in the rdd_data into training and test zeroes in the ratio 01:0.9 . We just chose 0.1% of the zeroes for training the dataset to prevent overfitting as there are more number of zeroes than ones. This is a shuffling stage where each data goes to either partition:trainingZeroes or testZeroes. This involved 964 tasks.

```
val Array(trainingZeroes, testZeroes) = zeroes.randomSplit(Array(0.1,
0.9), seed = 11L)
```

4) Then we aggregated the total number of ones in the dataset with a small percentage of zeroes and put them in the training data. Then we converted it into RDD of labelled points so that we can use it to train the model.

Similarly,we do that for the test data also but here we do union of ones with testZeroes instead of trainingZeroes.

```
  val training: RDD[LabeledPoint] = ones.union(trainingZeroes)

                                        .map{case (key, value) =>
                                  LabeledPoint(key ,Vectors.dense(value))}
```

5) We adjust the parameters of the classifier such as number of trees, featureSubsetStrategy, impurity, max_depth of the tree, categoricalFeaturesInfo.

6) We train RandomForest model using the values of the parameters used in the previous step.

```
    val model =
RandomForest.trainClassifier(training,num_of_classes,categoricalFeaturesInfo,

numTrees,featureSubsetStrategy,impurity,max_depth,maxBins)
```

7) Then we run the test data on the RandomForest model to get the predicted outcomes.

```
  val predictionAndLabels = test.map { case LabeledPoint(label, features) =>
       val prediction = model.predict(features)
       (prediction, label)
     }
```

8) Then we filter out the true positives, true negatives, false negatives and false positives from predictionAndLabels RDD. The filter operations do not require shuffling of data, hence they only require 152 tasks.

9) Then we compute ROC and accuracy of the model.

Running time and speedup results for the model training and the prediction phase.

Number Of Machines	Running Time(in minutes)
11	32.23
20	27.14
30	23.45

The speedup = Running time on 11 machines/ Running Time on 20 machines

= 32.23 / 27.14

= 1.18

Depth, Number of Trees	Running Time
4,10	26.45
10,10	38.23
4,50	42.44

As we see in the table above, increasing the depth and number of trees of the tree increases the running time. But it improved accuracy.

Prediction PseudoCode

1) We read the model from the file and saved it in a val..

```
val sameModel = RandomForestModel.load(sc, args(1))
```

2) We then made a pairRDD(label,features) of given dataset. No shuffling happens here as it just maps the input to each partition of the dataset. This required 482 tasks.

val rdd_data(label,features) = make pairRDD of the given datset

3) Then we make the prediction based on the model. This stage does not require shuffling and required 982 tasks to complete.

```
    val predictionAndLabelsForValidation = new_rdd_data.map { case
LabeledPoint(label, features) =>
       val prediction = sameModel.predict(features)
       (prediction.toInt)
     }
```

References: https://databricks.com/blog/2015/06/22/understanding-your-spark-application-through-visualization.html
https://spark.apache.org/docs/2.2.0/mllib-ensembles.html
https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm