Class Number: CS6240 Parallel Data Processing in Map Reduce
HW Number: Image Prediction Project
Name: Neha Pradhan & Rakesh Buchankrishnamurthy

CS6420 – Image Prediction Project Report

With the knowledge of various classification techniques discussed in class, we intuitively were aware that the probability of better predictions would increase using ensemble methods. Therefore, our research was focused on implementations of ensemble methods and we chose Random Forest model as it seemed less prone to overfitting, had parameter that were intuitively easy to tune and most resources vouched for this model.

Having decided the model we got down to analyzing the dataset provided for training the model. Working with such a large data set warranted random sampling of the input data set to view features and to ensure balanced frequency of both foreground and background labeled data to train the model(since 99.9943% of the validation data is labeled as background). Initial discussions for deciding crucial features for each record is described in detail in the report.

Questions:

1. **How many tasks are created during each stage of the model training process?**
   Stages in Spark are delineated by the occurrences of shuffles. While explicit programs written for training (prediction requires shuffling when repartitioning to save the prediction data) do not shuffle data, the Random Forest algorithm internally shuffles the data while constructing the model. Intuitively one could state that the number of partitions in each stage would determine the number of tasks in each stage and on careful observation of logs, we found our intuition to be consistent with the algorithm having around 70 stages with the number of partitions in each stage dictating the number of tasks in that stage, a total of 18,151 tasks.

2. **Is data being shuffled?**
   During Model Training:
   There is no explicit action/transformation that causes the data to be shuffled in our program, but Spark's implementation of random forest model requires data to be shuffled as it uses functions like collectToMap().

   During Prediction:
   The predict method may require data to be shuffled but as this happens internally we cannot say for sure, there is however data shuffle occurring when we do a repartition(1) to accumulate the predictions across all partitions.

3. **How many iterations are executed during model training (for methods that have multiple iterations)?**
   Random Forest ensemble learning method does not require any explicit parameter for tuning the number of iterations, but while checking logs we could find certain methods of the algorithm being called multiple times at the same line number, this could either mean that the model internally runs iteratively or that the calls are representative of each partition's execution of the algorithm.

4. **You also need to find out, how to control performance.**
   Performance is dependent on multiple hyper parameters, like numTrees, maxDepth, featureSubsetStrategy and maxBins which is discussed in following sections of the report. To summarize, reducing the numTrees and maxBins reduces the execution time (good performance) but could lead to poor precision for lack of features over which to split on nodes.

5. **How did changes of parameters controlling partitioning affect the running time? E.g., for bagging, was it better to partition the model file, the test data, or both?.**
   As per the suggested approach in questionnaire, we used whole image files as training data and tested it on another whole image file. Though it was minor change, but still provided a better prediction results when compared to random splitting of the input files into training/testing data.

<u>Model Used:</u>

We used the Random Forests ensemble learning method, with the hyper parameters configured to following values:

- numClasses = 2 (1 or 0 – being the possible labels to predict)
- numTrees = 40 (the number of trees used to make a decision, majority predicts the label)
- featureSubsetStrategy = "auto" (is set to “all”, if numTrees = 1 else will be set to “sqrt”)
- impurity = "gini" (allows split at node if information gain > 0.0)
- maxDepth = 15 (the depth of each tree, more the depth of the tree more are the feature considered during splitting)
- maxBins = 140 (allows to categorize continuous features and sets the conditions for splitting at each node)

<u>Working of Random Forest Algorithm:</u>

1. The random forest algorithm creates as many number of trees as specified by the parameter numTrees in parallel.
2. It selects certain features from the provided feature vector while creating decision trees. This is given by the featureSubsetStrategy parameter which in our case selects the sqrt of total number of features and uses this for creating splits at each tree node.
3. Features could be categorical or continuous which determines the branching of the decision trees. The features of our dataset could be described as being more discrete than categorical or continuous, but since the value of brightness feature could range from 0-255 we consider these features as continuous using inequality as the basis for branching.
4. The impurity parameter set as gini ensures that splitting at each level happens if information gain > 0.0.
5. Steps 3 and 4 are repeated until maxDepth is reached.

Class Number: CS6240 Parallel Data Processing in Map Reduce
HW Number: Image Prediction Project
Name: Neha Pradhan & Rakesh Buchankrishnamurthy

Parameters explored:

The Experiments section of the report provides the results of tuning of the parameters used by the random forest model. We tried different combinations of the hyper-parameters to achieve greater accuracy. The tuning of these parameters were mostly done with a hit and trial approach changing one parameter while keeping others constant to achieve the maximum possible accuracy and then manipulating another parameter keeping all others constant and repeating this process until we could find the best possible combination of parameters.

During the data analysis phase, we assumed that the pixel present at the center column (1543) of each record was the main factor contributing to the label. So, we first filtered only the columns – center column and their nearest neighbors (present at the columns - 1521, 1522, 1523, 1542, 1544, 1563, 1564, 1565, 1962, 1963, 1964, 1983, 1984, 1985, 2004, 2005, 2006, 1080, 1081, 1082, 1101, 1102, 1103, 1122, 1123, 1124)

The above approach was extended to include the next level of neighbors (present at columns - 1499, 1500, 1501, 1502, 1503, 1520, 1524, 1541, 1545, 1562, 1566, 1583, 1584, 1585, 1586, 1587, 1940, 1941, 1942, 1943, 1944, 1961, 1965, 1982, 1986, 2003, 2007, 2024, 2025, 2026, 2027, 2028, 1058, 1059, 1060, 1061, 1062, 1079, 1083, 1100, 1104, 1121, 1125, 1142, 1143, 1144, 1145, 1146), improve the accuracy which did provide better results.

The concern we faced with this approach was that we could be overfitting the model which may give poor results with the validation data and hence we decided to consider at all 3087 features present in each column of the input record.

Pseudocode:

Model Training:

```
1. Read the training data from training_data folder
2. Read the test data from testing_data folder
3. Loop over each record of training_data set
    a. Split on commas(,)
    b. Create a LabeledPoint pair with label (last split) and featureVector (all remaining splits)
4. Loop over each record of testing_data set
    a. Split on commas(,)
    b. Create a LabeledPoint pair with label (last split) and featureVector (all remaining splits)
5. Set parameters for Random forest training model
6. Train the model with training_data
7. Test the model with testing_data by only providing its featureVector to the model.
8. Predicted label can be compared with actual label to get accuracy.
9. Save the generated model.
```

Class Number: CS6240 Parallel Data Processing in Map Reduce
HW Number: Image Prediction Project
Name: Neha Pradhan & Rakesh Buchankrishnamurthy

Prediction:

1. Read the input data from prediction_data folder
2. Loop over each record in the input data
    a. Split on commas(,)
    b. Create a LabeledPoint pair with label (0.0) and featureVector (all splits except the last)
3. Load the model saved during the training phase
4. Provide features of the input data to the model to get prediction results

Model Performance:

Job	Workers	Running Time (in mins)
Training	20	80
Prediction	20	9

Note: Running the training model on 11 machines for the entire dataset took more than 3 hours running requiring us to terminate the cluster with no output. Increasing the number of machines to 20 greatly reduced the running time to a little less than an hour. This indicates good speed up with the use of more machines. We could not use more than 20 machines due to account usage limits and did not want to go back to using lesser machines as the time to obtain results would increase.

Experiments:

We did few experiments to tune the model in order to achieve better accuracy, which involved changing Random Forests ensemble learning method's parameters and/or input record parameters as below:

Tests performed with sampled data set (Sampled for 95% foreground pixesl and 5% background pixesl):

numTrees	maxDepth	maxBins	Accuracy
20	10	40	0.976931
30	10	40	0.980552
40	10	40	0.981847
15	7	40	0.964912
15	8	40	0.970588
15	11	40	0.953704
15	15	40	0.955752
15	20	40	0.955682

From these results, we concluded that increasing the numTrees and maxDepth at 15 provided better accuracy, so we ran the entire input set on AWS with 10 workers and with settings numTrees = 80, maxDepth = 15 and maxBins = 250 (maxBins value would need to be approximately set to range of values that each feature could have which in our case ranges from 0 to 255) This resulted in very long execution time for model training.

Hence we decided to provide only a subset of the features from the data set (including only the center pixels and their nearest neighbors at two levels as discussed in the parameter section). Following table provides the results of the AWS run on the entire dataset parameter tuned accordingly:

numTrees	maxDepth	maxBins	Worker machines	filtered Data set	Execution Time(in min)	Accuracy
80	15	250	10	First level neighbors	53	0.997294408
80	15	250	10	First & Second level neighbors	63	0.997421152
75	15	250	20	First & Second level neighbors	33	0.997382311

Due to the afore-mentioned concern of overfitting the model we decided to include all the 3087 features of data set. To do this we had to reduce the numTrees and maxBins parameters which allowed a successful AWS run with decent run-times. Following table provides the details of AWS run on entire dataset with reduced number of trees:

numTrees	maxDepth	maxBins	Worker machines	Execution Time(in min)	Accuracy
40	15	40	20	59	0.99750599
45	15	40	20	62	0.99749065
50	15	40	20	69	0.99749576
40	15	60	20	53	0.99754278
40	15	80	20	62	0.99757856
40	15	100	20	67	0.99758469
40	15	140	20	80	0.99759695

Note: Accuracy/Precision values are obtained using the MultiClassMetrics class which provides the evaluation metrics for the prediction model.