Foreground-Background classification in high-resolution brain scans

~Chandrika Sharma~
~Meng Tao~

Pseudo Code

We have used Spark ML library for Machine learning algorithms. We decided to go ahead with Random Forest implementation as it uses an ensemble of decision trees.

PREPROCESSING:

Create an rdd from the csv files of the images.
Split by comma to create an rdd of type RDD[Array[Double]]

ROTATION to create extra training records:

To rotate, we first took each record which was of length 3088 and took the label out, leaving a only the brightness values of length 3087.

Mirroring along y axis:

1. Group each record by 441 elements, this creates 7 arrays of each record thereby RDD[Array[Array[Double]]]
2. Map on each of the 7 arrays of 441 elements and group by 21
3. Each array with 21 elements is reversed, flattened back into array of 441 elements.

Rotation 90 degree:

1. Group each record by 441 elements, this creates 7 arrays of each record thereby RDD[Array[Array[Double]]]
2. Loop over each 441 elements say arr[](441):

```
var d = new ArrayBuffer[Double](441)
for (i<-0 to 20){
        d.append(arr(i))
        for(j<-1 to 20){
        var f = i+(j*21)
        d.append(arr(f))
        }
}
```

3. Array d is the new 90 degree rotated array.

Rotation 270 degree:

1. Take input record and Rotate 90 degrees and flatten
2. Group by 21 and reverse the order of the each element in the arrays and flatten again

Rotate 180 degrees:

1. Group the entire record by 441 and convert to array
2. Inside each of the 441 length arrays, group by 21

3. Reverse the order of these 21 arrays and flatten

Random Forest Model training

For each of the rotations and their mirrors, 1 model (total 8 models) was created with the following parameters:

Parameters	**Values**
Number of Classes	2
Categorical Features Information	Map[Int,Int]()
Number of Trees	10
Feature Subset Strategy	auto
Impurity	gini
Max Depth	10
Max Bins	32

RandomForest.trainClassifier(trainingData1, numClasses, categoricalFeaturesInfo,numTrees, featureSubsetStrategy, impurity, maxDepth, maxBins)
**Save each model

Prediction Program(Bagging):

1. Load the saved models form the memory
2. Load test data from memory
3. For each record in the test set, predict label on each model
   model1.predict(point.features)
   model2.predict(point.features)
   model3.predict(point.features)
   model4.predict(point.features)
   model5.predict(point.features)
   model6.predict(point.features)
   model7.predict(point.features)
   model8.predict(point.features)
4. Result for each record would be the majority vote from the 8 labels obtained.

Discussion

As shown by the “Tasks: Succeeded/Total” column from the table below, in each stage of model training process, there are 384 tasks generated.

Completed Stages (99)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
98	count at DecisionTreeMetadata.scala:118 +details	2018/04/22 01:31:50	8.0 min	384/384	23.9 GB			
97	take at DecisionTreeMetadata.scala:112 +details	2018/04/22 01:31:49	0.8 s	1/1	16.0 KB			
96	collectAsMap at RandomForest.scala:563 +details	2018/04/22 01:31:40	8 s	152/152			820.0 MB	
95	mapPartitions at RandomForest.scala:534 +details	2018/04/22 01:31:08	32 s	384/384	45.3 GB			820.0 MB
94	collectAsMap at RandomForest.scala:563 +details	2018/04/22 01:31:02	6 s	152/152			605.5 MB	
93	mapPartitions at RandomForest.scala:534 +details	2018/04/22 01:30:37	25 s	384/384	45.3 GB			605.5 MB
92	collectAsMap at RandomForest.scala:563 +details	2018/04/22 01:30:32	4 s	152/152			449.5 MB	
91	mapPartitions at RandomForest.scala:534 +details	2018/04/22 01:30:21	11 s	384/384	45.3 GB			449.5 MB
90	collectAsMap at RandomForest.scala:563 +details	2018/04/22 01:30:18	3 s	152/152			338.3 MB	
89	mapPartitions at RandomForest.scala:534 +details	2018/04/22 01:30:10	8 s	384/384	45.3 GB			338.3 MB

As shown by the “Shuffle Read” and “Shuffle Write” columns from the table above and below, the data is shuffled in each stage of model training process, whereas there are no data shuffling in prediction process.

Active Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
33	count at Prediction.scala:79 +details (kill)	2018/04/22 14:00:08	3 s	17/98	4.1 MB			

Completed Stages (33)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
32	count at Prediction.scala:78 +details	2018/04/22 13:58:01	2.1 min	98/98	6.1 GB			

Since we are using Random Forest as the training model, we don’t have multiple iterations for model training.

During model training process, we use mapPartition, which adapts according to the number of machines accordingly. By using mapPartition, instead of collecting all partitions together then mapping, different mapping processes happen in each individual partitions separately. Therefor, by evenly distributing rdds to all machines and enabling mapping to happen in each individual partitions, the load balancing, scalability, speedup and running time of our program is optimal.

For bagging, we use the 8 models generated by image rotation and mirroring. We are using the majority vote strategy for getting the label of validation dataset, i.e. in each prediction, we need all of the 8 models to give their predictions and get the highest voted label as the label generated by bagging these models. Hence, we are partitioning the test data and persisting all the models in each machine.

Pre-processing Summarization

For data type pre-processing, data conforms into LabeledPoint data type to be utilized for Spark MLlib Random Forest model training input.
We also transform the data to get more labeled data for data diversity and potential higher accuracy. For data diversity pre-processing, as stated in pseudo-code, given 3-d dataset are transformed into 7 transformations:

- mirror by Y-axis
- rotate by 90 degrees, with or without mirroring
- rotate by 180 degrees, with or without mirroring
- rotate by 270 degrees, with or without mirroring

Parameter Settings

Local parameter tuning: 300 MB data

SNo.	NumTrees	MaxDepth	Accuracy
1	10	10	99.51%
2	10	20	99.59%
4	5	40	99.58%
3	10	40	99.63%

AWS parameter tuning: All Data

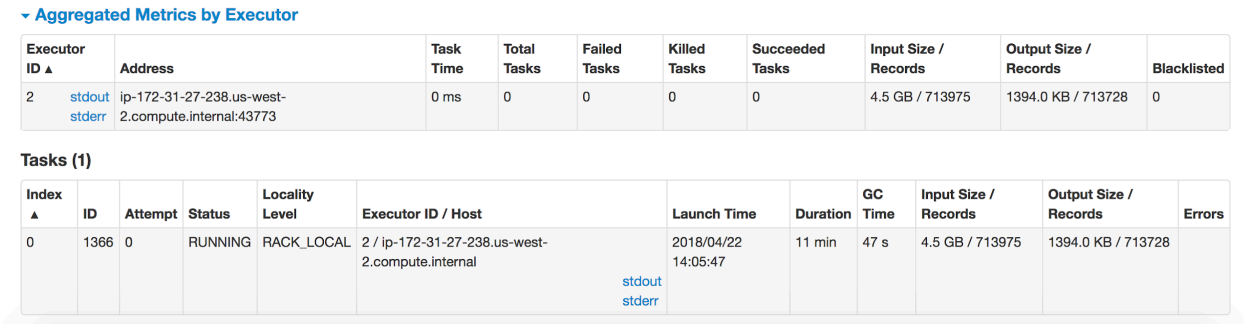
SNo.	NumTrees	MaxDepth	Accuracy
1	20	10	99.71%
2	10	10	99.73%

We find that the accuracy is higher with more number of trees and maximum depth. However, since according to the official document, and based on our own experiments, the training time increases roughly linearly in the number of trees. Considering the cost of training 8 models in our solutions, we decide to stick to the parameters of NumTrees: 10 and MaxDepth: 10.

Running Time and Speedup

Following time is based on training time for one model and prediction time for test dataset:

	Training Time (min)	Prediction Time (min)
6 m4.large	54.6	4.7
11 m4.large	22.4	4.6



▾ Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Input Size / Records	Output Size / Records	Blacklisted
2 stdout stderr	ip-172-31-27-238.us-west-2.compute.internal:43773	0 ms	0	0	0	0	4.5 GB / 713975	1394.0 KB / 713728	0

Tasks (1)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Output Size / Records	Errors
0	1366	0	RUNNING	RACK_LOCAL	2 / ip-172-31-27-238.us-west-2.compute.internal stdout stderr	2018/04/22 14:05:47	11 min	47 s	4.5 GB / 713975	1394.0 KB / 713728	

Our model training program has good scalability based on the evaluation of speedup. The speedup of the model training process is 2.44.

Since we want to get a single file in same rows with the test points for final prediction output, we make the partition for prediction program as one. Thus, the speedup of the prediction program is around 1.