Problem Statement

Use data mining to predict sightings of the Red-winged Blackbird in birding checklists

Methodology

We have used Spark MLlib for this project and in MLLib, we have used Random Forest Classification algorithm to train a model using labeled data set and to predict the sightings of Red-winged Blackbird in unlabeled data set. Random Forests are ensembles of decision trees. As we have used ensemble technique, our goal is to get a good bias while training specific decision tree and to reduce the risk of overfitting and achieve good variance by combining many of these decision trees. For model training and validation, we partitioned the full labeled data into training (70%) and validation (30%) and tried to get the best trained accuracy. To achieve this goal, we have analyzed the impact of many model parameters like number of trees, strategy for selecting subset of features, maximum depth of the tree, features categorization, number of bins for discretizing continuous features, amount of memory to be used for collecting sufficient statistics and parameters to cache the model. We have also analyzed the input data to select the relevant features only which can take us to better prediction. Once a good number of runs were performed varying the features, tuning MLlib parameters for training model and achieving satisfactory training accuracy (86%), we have finalized values of parameters like numTrees = 33, featureSubsetStrategy = "auto", impurity = "gini", maxDepth = 20, maxBins = 32 and number of features as 47. We used this configuration to train the final model on full labeled data set and to predict the unlabeled data set. The in-build capabilities in spark MLlib such as RDD, parallel iterative computation and tunable parameters provided helped in building scalable solution.

Design Details

- Pre-Processing & Feature Selection
- Model Training
- Prediction

Preprocessing & Feature Selection

We are converting input data to Labeled Point (Label with associated features represented as sparse Vector). We used sparse representation as it is more space efficient and elegantly handles missing values. Sparse Vector is created providing its size, index array and value array.

The following criteria were considered in the preprocessing of the data as part of converting the input data to Labeled Point Vector:

1. Header records are not considered
2. If the value of species Red-winged Blackbird (*Agelaius phoeniceus*) is X or greater than 0 for a record ,then label is considered as 1.0 else 0.0
3. Only a subset of features (count-47) from entire data set were considered in creation of labeled point that are as follows: index values for features where index starts from 0.
   (2, 3, 5, 11, 12, 13, 14, 16, 24, 25, 45, 53, 139, 955, 956, 957, 958, 959, 960, 962, 963, 964, 965, 966, 967,1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1078, 1079, 1080, 1081, 1082, 1083) Please refer SelectedFeatures.xls provided in deliverables for more details on these features.
   a. LOCID, YEAR, DATE, TIME, COUNTRY, STATE and COUNTY were not considered as they are of low importance, and LAT, LONG provides the location features required in training
   b. GROUPID, SAMPING EVENT ID, PRIMARY CHECKLIST FLAG were ignored
   c. Features from index 11-14,16 were impacting in sighting of a species and are considered
   d. Features from index 955- 960 related to temperature, elevation density, population per sq km were researched to be helpful in bird sightings.

e. Features from index 1000 to 1015 i.e. Percent of surrounding landscape that is habitat class for year 2011 were found to be important factor in bird habitats and were considered.
f. Features from index 1078 to 1083 were included as analyzed from Cornell University dissertation titled **supporting analysts before and after the learning algorithm**.
g. Similar Species to Agelaius phoeniceus are considered in features as well (Index- 24,25 ,45,53,139. The list was found from https://www.allaboutbirds.org/guide/Red-winged_Blackbird/id)

4. If any of the above selected features have a value ? or "" or X or empty, then they are not considered in creation of vector except similar species (24,25 45,53,139).
5. Features – MONTH, COUNT TYPE and count of similar species occurrence are converted to categorical feature before including them in the vector.
6. Species that were found to be similar (24,25 45,53,139) are processed following same rules as mentioned in bullet point 2.

PSEUDO CODE

Model Training Phase:

inputdata = readDataToRDD(inputFile)
labeledData = parse(inputdata) // parse each line and select only relevant features
model = RandomForest.trainClassificationModel(labeledData)
persist(model)

Prediction Phase

model = load(modelPath)
inputdata = readDataToRDD(inputFile)
unlabeledData = parse(inputdata) // parse each line and select only relevant data
predictions = model.predict(unlabeledData)
write(predictions)

Detailed pseudo code for model training and prediction by random forest classification given in next section.

Model Training

Random Forest Ensemble Algorithm

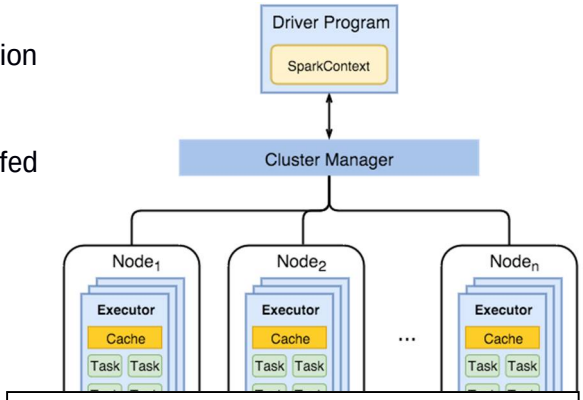
Random forests are ensembles of decision trees. It trains a set of decision trees separately, so that the training is done in parallel.

An RDD of Labeled points is created from the preprocessed data set and fed to Spark mllib.RandomForest classifier with various parameters

```
val splits = labeledData.randomSplit(Array(0.7, 0.3))
val (trainingData, testData) = (splits(0), splits(1))

// Train a RandomForest model.
// Empty categoricalFeaturesInfo indicates all features are continuous.
val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int]()
val numTrees = 25
val featureSubsetStrategy = "auto"
val impurity = "gini"
val maxDepth = 20
val maxBins = 32

val model = RandomForest.trainClassifier(trainingData, numClasses, categoricalFeaturesInfo,
  numTrees, featureSubsetStrategy, impurity, maxDepth, maxBins)
```



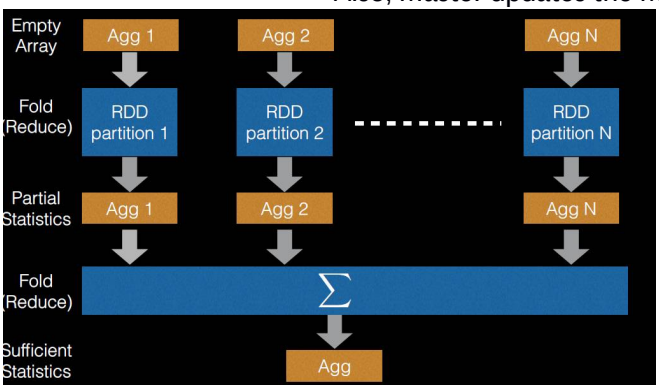
Major Components of Spark Code
(multiple tasks are combined in Stage which don't require data shuffling/repartitioning)

The data is partitioned by rows and it can be manually specifying number of partitions by doing **repartition(numberOfPartitions).**

Below are the detailed steps happening inside MLLIB RandomForest:

1. Spark MLlib constructs number of trees specified in the train Classifer parallelly. Below are detailed steps:
   a. Find the splits and the corresponding bins (interval between the splits) using a sample of the input data if the feature has continuous values after which each continuous feature becomes an ordered discretized feature with at most maxBins possible values. (at Master Node)
   b. Convert an input dataset (RDD spread across many workers) into its TreePoint representation (RDD), binning feature values in preparation for DecisionTree training.
   c. Convert the previous RDD into its BaggedPoint representation RDD (spread across many workers) and cache input RDD for speedup during multiple passes.
   d. At first, all the all data belong to the root nodes
   e. A queue is created at master node that is a FIFO queue of tree nodes to train ((treeIndex, node))
   f. Now, until the queue is empty,
      i. some number of nodes are pulled off the queue and a subset of candidate features are chosen for each node at master node
      ii. For a BaggedPoint RDD, given node pulled, and candidate features, worker with a RDD partition1, worker with RDDpartition2 ….workerN with RDD partitionN compute the best split among the candidate feature (At Worker Nodes)
      iii. For each node, the statistics for that node are aggregated to a worker via reduceByKey(). The designated worker chooses the best (feature, split) pair
      iv. The best split statistics from workers are sent back to master (see screenshot below where RDD on worker hold the partial statistics which are then sent to master)
      v. Master grows the tree by creating more nodes and adding them into the queue as needed

Also, Master updates the Model and passes the updated RandomForestmodel to workers



For each iteration of the while loop (queue.isNoEmpty())
For each tree node
For each feature
For each bin,
Stats are generated

set of stats(bestsplit) transferred is **(# of nodes)* (# of features)*(#bins/feature) for a Decision Tree in the forest**

(#bins/feature) - This can be controlled by maxBins parameter
There are suffleRead and shuffleWrite at end of each iteration and amount of this data increases with each iteration of loop as the trees build level by level with each iteration in a Forest. This is the major data communication overhead.

We utilize **SPARK UI on AWS instance** to understand logs and partitioning of input data and training model generated. Below is our understanding of the partition creation, comm. b/w mater, workers and data shuffle across the spark parallel execution (10 machines run, 30 trees in config):

Model Training Findings: The input dataset (labeled) is partitioned by rows and RDD is created that is partitioned across 7 worker machines as the input is converted to RDD[labeledPoint] representation. The partitioning can be improved using repartition to partition across all 10 workers and fully utilize all workers

Below is RDD[LabeledPoint] partition size at each worker

Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Succeeded Tasks	Input Size / Records	Shuffle Write Size / Records
1	ip-172-31-13-42.us-east-2.compute.internal:36639	1.3 min	1	0	1	64.1 MB / 287515	324.2 KB / 79430
2	ip-172-31-5-182.us-east-2.compute.internal:41879	1.3 min	1	0	1	64.1 MB / 283284	320.2 KB / 77832
5	ip-172-31-12-35.us-east-2.compute.internal:41702	22 s	1	0	1	18.6 MB / 74829	86.0 KB / 20398
6	ip-172-31-9-140.us-east-2.compute.internal:38596	1.3 min	1	0	1	64.1 MB / 271907	301.5 KB / 74401
7	ip-172-31-11-136.us-east-2.compute.internal:43752	1.2 min	1	0	1	64.1 MB / 277923	303.7 KB / 74354
8	ip-172-31-9-134.us-east-2.compute.internal:36648	1.0 min	1	0	1	64.1 MB / 225556	274.5 KB / 64249
9	ip-172-31-8-86.us-east-2.compute.internal:33760	1.3 min	1	0	1	64.0 MB / 280962	290.7 KB / 77832

Once, the **TrainClassifier is called on MLLIB RandomForest**, multiple trees are trained in parallel until the queue of nodes for all trees in Forest is empty (*See previous page for more details*). In each iteration, size of statistics sets (# of nodes)* (# of features)*(#bins/feature)) increases as size of nodes increases at each level. So, the data transferred from all workers nodes to driver increases. The same can be seen in the below screenshot i.e. at Stage 29 data transferred is 107 MB which increases to 125 MB at Stage 41 due to increase in nodes and at last it reduces to 117 MB at leaf nodes

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
50	mapPartitions at RandomForest.scala:521 +details	2017/04/22 20:33:38	18 s	7/7	863.4 MB			114.6 MB
49	collectAsMap at RandomForest.scala:550 +details	2017/04/22 20:33:01	5 s	80/80			117.7 MB	
48	mapPartitions at RandomForest.scala:521 +details	2017/04/22 20:32:46	15 s	7/7	863.4 MB			117.7 MB
47	collectAsMap at RandomForest.scala:550 +details	2017/04/22 20:32:13	4 s	80/80			120.1 MB	
46	mapPartitions at RandomForest.scala:521 +details	2017/04/22 20:31:56	17 s	7/7	863.4 MB			120.1 MB
45	collectAsMap at RandomForest.scala:550 +details	2017/04/22 20:31:21	4 s	80/80			120.0 MB	
44	mapPartitions at RandomForest.scala:521 +details	2017/04/22 20:31:07	14 s	7/7	863.4 MB			120.0 MB
43	collectAsMap at RandomForest.scala:550 +details	2017/04/22 20:30:36	4 s	80/80			122.6 MB	
42	mapPartitions at RandomForest.scala:521 +details	2017/04/22 20:30:23	13 s	7/7	863.4 MB			122.6 MB
41	collectAsMap at RandomForest.scala:550 +details	2017/04/22 20:29:56	4 s	80/80			123.5 MB	
40	mapPartitions at RandomForest.scala:521 +details	2017/04/22 20:29:43	12 s	7/7	863.4 MB			123.5 MB
39	collectAsMap at RandomForest.scala:550 +details	2017/04/22 20:29:17	4 s	80/80			128.1 MB	
38	mapPartitions at RandomForest.scala:521 +details	2017/04/22 20:29:04	13 s	7/7	863.4 MB			128.1 MB
37	collectAsMap at RandomForest.scala:550 +details	2017/04/22 20:28:44	4 s	80/80			130.2 MB	
36	mapPartitions at RandomForest.scala:521 +details	2017/04/22 20:28:33	11 s	7/7	863.4 MB			130.2 MB
35	collectAsMap at RandomForest.scala:550 +details	2017/04/22 20:28:12	4 s	80/80			135.4 MB	
34	mapPartitions at RandomForest.scala:521 +details	2017/04/22 20:28:00	12 s	7/7	863.4 MB			135.4 MB
33	collectAsMap at RandomForest.scala:550 +details	2017/04/22 20:27:44	5 s	80/80			141.3 MB	
32	mapPartitions at RandomForest.scala:521 +details	2017/04/22 20:27:31	12 s	7/7	863.4 MB			141.3 MB
31	collectAsMap at RandomForest.scala:550 +details	2017/04/22 20:27:17	6 s	80/80			148.3 MB	
30	mapPartitions at RandomForest.scala:521 +details	2017/04/22 20:27:02	15 s	7/7	863.4 MB			148.3 MB
29	collectAsMap at RandomForest.scala:550 +details	2017/04/22 20:26:50	5 s	80/80			107.5 MB	
28	mapPartitions at RandomForest.scala:521 +details	2017/04/22 20:26:38	12 s	7/7	863.4 MB			107.5 MB
27	collectAsMap at RandomForest.scala:550 +details	2017/04/22 20:26:32	2 s	80/80			75.8 MB	
26	mapPartitions at RandomForest.scala:521 +details	2017/04/22 20:26:22	10 s	7/7	863.4 MB			75.8 MB

Prediction /Validation

```
// Evaluate model on test instances and compute test error
val labelAndPreds = testData.map { point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
}
```

The input test data is partitioned as RDD's across various workers. The RandomForestModel (Forest of Decision Trees) from training Phase is broadcasted to all workers by driver. Each of the workers compute the prediction based on the model and emit the prediction for its data. Thus, there is no shuffling of prediction data after creating initial labeled point RDD. If we see this in detail then the pseudo code can look like below in MapReduce:

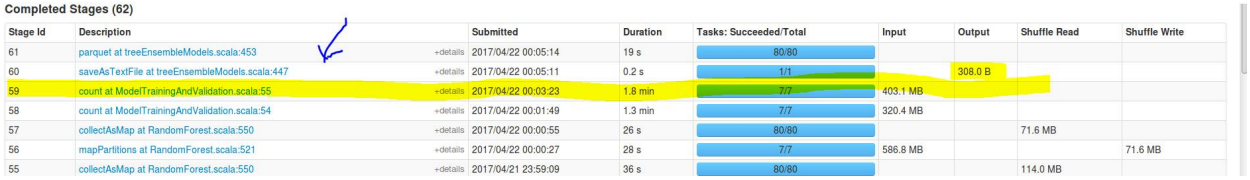
```
Mapper {
        Setup() {
                models = readRandomForestModel()
        }
        map(Record unlabeledRecord) {
                prediction = models.predict(unlabeledRecord)
                emit(unlabeledRecord, prediction)
        }
}
```

Validation Findings

- Prediction run using model trained on input data (70% labeled data) and test data (30% of Labeled Data) for validating accuracy of trained model. Here, prediction was run as part of same spark job that trains the model.

We see that the entire labeled data is read in input (100%) and when the test data RDD is lazy evaluated, 30% of this labeled data is used for prediction (Stage 59 in the screenshot below)

The amount of data read as input is 403 MB (test Data RDD), with trained model size (71.6 MB) and output (predictions) size is 308 Bytes as highlighted below:



Completed Stages (62)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
61	parquet at treeEnsembleModels.scala:453 +details	2017/04/22 00:05:14	19 s	80/80				
60	saveAsTextFile at treeEnsembleModels.scala:447 +details	2017/04/22 00:05:11	0.2 s	1/1		308.0 B		
59	count at ModelTrainingAndValidation.scala:55 +details	2017/04/22 00:03:23	1.8 min	7/7	403.1 MB			
58	count at ModelTrainingAndValidation.scala:54 +details	2017/04/22 00:01:49	1.3 min	7/7	320.4 MB			
57	collectAsMap at RandomForest.scala:550 +details	2017/04/22 00:00:55	26 s	80/80			71.6 MB	
56	mapPartitions at RandomForest.scala:521 +details	2017/04/22 00:00:27	28 s	7/7	586.8 MB			71.6 MB
55	collectAsMap at RandomForest.scala:550 +details	2017/04/21 23:59:09	36 s	80/80			114.0 MB	

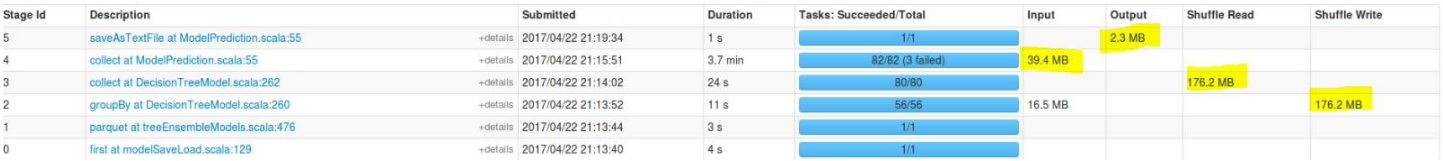
Prediction Findings:

- Prediction run using model trained on input data (100% labeled data) and test data (100% unlabeled data). Here, prediction was run as separate spark job using the model saved to disk by separate model training job

Stage 2 ShuffleMapTask writes RDD blocks (unlabeled data in sparse vector representation, SIZE = 176.2 MB) to local, and then the task in the next stage 3 fetches these blocks over the network, uses it and broadcasted trained model (39.4 MB) to perform prediction in stage 4. The prediction results written to disk in Stage 05 and output size is 2.3 MB

Model

Unlabeled Data



Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
5	saveAsTextFile at ModelPrediction.scala:55 +details	2017/04/22 21:19:34	1 s	1/1		2.3 MB		
4	collect at ModelPrediction.scala:55 +details	2017/04/22 21:15:51	3.7 min	82/82 (3 failed)	39.4 MB			
3	collect at DecisionTreeModel.scala:262 +details	2017/04/22 21:14:02	24 s	80/80			176.2 MB	
2	groupBy at DecisionTreeModel.scala:260 +details	2017/04/22 21:13:52	11 s	56/56	16.5 MB			176.2 MB
1	parquet at treeEnsembleModels.scala:476 +details	2017/04/22 21:13:44	3 s	1/1				
0	first at modelSaveLoad.scala:129 +details	2017/04/22 21:13:40	4 s	1/1				

Parameters Explored, Accuracy Results, Speedup Statistics

We tested the accuracy of the training model (model trained using 70% of labeled data and predicted with remaining 30% of labeled data) with different parameters. We found that below parameters impact the accuracy the most.

Parameters	Description & Impact	Value	Accuracy
numTrees	Number of trees in the forest. Increasing the number of trees decrease the variance in predictions and improves the model's test-time accuracy. So higher is better.	5	0.78456827
		10	0.814152437
		25	0.854160143
		50	0.865160261
maxDepth	Maximum depth of each tree in the forest. Increasing the depth of the tree reduces the bias but increase the variance. As variance can be reduced by averaging multiple trees in the forest we can go with the higher variance. So higher is better.	10	0.82034716
		20	0.864132463
		30	Failed
maxBins	Number of bins used when discretizing continuous features. Increasing maxBins allows us to consider more split candidates and make fine-grained split decisions which can increase the accuracy up to some level.	32	0.863907125
		64	0.864132463
features	We have spent most of the time in finding most relevant features which can improve the accuracy and come up with best possible combination. The more relevant the features are the better accuracy we will get.	Without similar species	0.853495617
		With Similar species	0.864160143
featureSubsetStrategy	Rather than selecting all the features while finding best split in making decision tree, we have chosen the "auto" option. This option takes the square root of # of features and randomly selects those many features only while finding best split. This option increases the independence between Decision trees in the forest and ultimately increases the accuracy.	"all"	0.856240238
		"auto"	0.865971042

Below are the run times and speed up metrics results on different configuration on AWS:

	Running Time (in seconds) (1 Master & 6 Worker)	Running Time (in seconds) (1 Master & 10 Worker)	% Reduction in Running Time
Model Training with Validation	1865	1244	33%
Model Training	1546	1307	15%
Prediction	511	372	27%

As per this table we can say that the solution scales well on increasing on machine. We found that as per the model parameter configuration, there is a lower limit that how many minimum number of machines you must need to train a model with specific parameter settings, e.g. you will get out of memory exception if you try to execute our program with 33 trees and 20 depth on 5 machines. As the random forest trains each tree in parallel, it will scale well on adding more machines. But here we have lot of communication is going on between Master and Client nodes and if you add high number of machines then normal requirement then you might get worst running time due to high communication cost. We got a case where running on 10 worker nodes gave the better running time then running on 20 worker nodes.

We have also analyzed the parameters as shown below which affects the performance of model training job.

Parameter	Description & Impact	Value	Running Time (s)
numTrees	Training time increases roughly linearly with number of trees	**30**	1865
		50	2381
maxDepth	Increases running time on increasing tree's max depth	**10**	705
		20	1712
featureSubsetStrategy	If the number of features to select best split is low then it will give better performance. So "auto" option gives good performance compare to "all" option	**auto**	1665
		all	5060
categoricalFeaturesInfo	Performance improves on properly designing categorical features	**with**	1712
		without	2046
maxBins	On increasing, it increases computation and communication cost so lower is better.	**32**	1712
		64	2046
useNodeIdCache	If this is set to true, the algorithm will avoid passing the current model (tree or trees) to executors on each iteration. So saves the communication cost in each iteration.	**TRUE**	1865
		FALSE	1244

BIBLIOGRAPHY

- OUTSIDE THE MACHINE LEARNING BLACKBOX: SUPPORTING ANALYSTS BEFORE AND AFTER THE LEARNING ALGORITHM A Dissertation Presented to the Faculty of the Graduate School of Cornell University in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy by Miles Arthur Munson May
- Improving Ensemble of Trees in MLlib Jianneng Li, Ashkon Soroudi, Zhiyuan Lin
- http://datastrophic.io/core-concepts-architecture-and-internals-of-apache-spark/
- https://github.com/apache/spark/blob/ef2f55b97f58fa06acb30e9e0172fb66fba383bc/mllib/src/main/scala/org/apache/spark/mllib/tree/RandomForest.scala
- https://databricks.com/blog/2014/09/29/scalable-decision-trees-in-mllib.html