Biyanta Shah
Sravani Beeram

CS6240: MR-PROJECT

- ## Prediction Model and parameters explored

We chose Random-Forests for training the model. These are the reasons why we felt it is a good choice.

- Random-Forests train a set of decision trees separately and thus can train multiple trees in parallel
- Training more trees reduces the likelihood of over-fitting because it reduces variance (by averaging)
- Easier to tune since performance increases monotonically as we increase the number of trees

Training process

The randomness is injected into the training process in the following ways:

- Bootstrapping the data (creating a random sample of the original data set on each iteration to get a different training set)
- Considering different random subsets of features to split at each node

Prediction process

For this problem, we needed to predict whether the bird was sighted or not, hence divide the output into two classes: 1.0 and 0.0 where 1.0 is a positive that the bird was sighted and 0.0 is when the bird wasn't sighted for a particular Event ID

This tells us that we needed to use Classification technique for prediction

- Classification is done by majority votes. Each tree's prediction is counted as a vote for one class. The class which receives the most votes, is the class predicted for that label

Parameters explored

1. Max Depth: Increasing the depth leads to over-fitting but due to averaging multiple trees, variance will be reduced, thus giving a better accuracy
2. Number of trees: Increasing the number of trees will decrease variance in predictions improving the model's accuracy, however more number of trees implies greater time for training

   Taking these two into consideration, we have tuned the trees to have a good balance between the depth and number of trees, so that we can get a good accuracy in a reasonable amount of time.

- ## Algorithm

For Training

- Load the labeled data into an RDD and extract the features required for training the model
- Creating a LabeledPoint structure where the label is the red winged blackbird and the features are the ones extracted. Filter out features which have no observations and create a sparse matrix of features with the remaining ones.
- Split the data into two parts, one for training and the other for testing
- Take the training data chunk and train a model using that data

Data is partitioned and trained as follows:

The input data is sampled into bins; the number of bins depends on the user defined parameters

The data is partitioned among all the workers. Each node/feature that needs to be split has its Gini index calculated. The Gini score for each feature is then shuffled on to the worker and the best split is decided based on the score. These splits are sent back to the master for creation of the trees

The depth till which the trees are grown and the number of trees created, depend on the parameters numTrees and maxDepth

- Predict the error by using that model on the test data

For Prediction

- Load the previously trained model
- Load the unlabeled data and extract the exact same features which were used in training
- For each test record run the trained model, each tree will predict a class and the class which receives the majority of the votes, will be the prediction output for that record

- **Pre-processing**

We performed pre-processing in order to extract the features, we thought would be useful to predict whether the red-winged blackbird was observed or not.

Here is a list of features we extracted from the columns of labeled data.

We were given a list of **core covariates** which were suggested by domain experts as the most important for species, for predicting accuracy and hence we included most of them. We haven't included the min and max temperatures as we felt that average temperature is a good enough measure for the both of them.

1. Bcr
2. Bailey_Ecoregion
3. Caus_prec
4. Caus_snow
5. Caus_temp_avg
6. Elev_gt (1 km* 1km approx.)
7. Housing density
8. Housing_percent_vacant
9. Pop00_SQMI

We were given a **checklist** file which had 16 columns describing the sampling events and then the other, close to 900 columns were regarding the species.
We have not considered any species other than the Red-winged bird (needed for prediction), since we lacked domain expertise for the relation between species. The other sampling event columns whose significance is covered by other columns are not included in the feature list. Example: Country, State and County are covered by Latitude and Longitude.

1. Latitude
2. Longitude
3. Year
4. Month

5. Time
6. Effort_hrs
7. Effort_distance_km
8. Effor_area_ha
9. Number_observers

One more list, **extended covariates**, contains the information about the habitats of the species. The attributes we chose from this file were relating to the water habitats

1. Dist_from_flowing_fresh
2. Dist_in_flowing_fresh
3. Dist_from_standing_fresh
4. Dist_in_standing_fresh
5. Dist_from_wet_veg_fresh
6. Dist_in_wet_veg_fresh
7. Dist_from_flowing_brackish
8. Dist_in_flowing_brackish
9. Dist_from_standing_brackish
10. Dist_in_standing_brackish
11. Dist_from_wet_veg_brackish
12. Dist_in_wet_veg_brackish

- **Accuracy for the parameters we tuned on**

These are not all the parameters we tuned, but a subset of them

Number of machines	Number of trees	Depth	Accuracy(%)
11	10	10	78.46
11	15	15	81.02
11	35	15	81.16
11	25	16	81.65
20	25	17	81.77
11	20	17	82.32
20	20	17	82.32
10	15	18	84.47
20	**15**	**18**	**84.51**

The maximum accuracy that we got was for 15 trees with the max depth of 18, on 20 m4.large machines.

From the above table these are the conclusions we formed:

- For a very small number of trees and depth, the accuracy goes haywire. This is because the complexities of the data are not captured very well
- Increasing the depth, increases the accuracy. This would be due to over-fitting the model, then when we average the trees, we decrease the variance

- **Running times for model training and prediction**

Number of machines	Number of trees	Max Depth of the tree	Model Training Time	Prediction Time
11 m4.large machines (workers)	15	18	38 minutes	12 minutes
20 m4.large machines (workers)	15	18	13 minutes	11 minutes

From the recorded results, this is what we can conclude:

- For model training, if we increase the number of machines, on the same number of trees and same maximum depth, the time from 11 machines to 20 machines (of type m4.large) does drop by a considerable amount. Thus we achieve a good speed-up
- For prediction, the increase in the number of machines for the same parameters, affects the prediction time by a very small amount, not resulting in a good speed up

- **MLlib - Random Forest Implementation**

Training by MLlib using Random Forests

1. **Splitting of a node**

We use Gini Index to split a node in two or more sub-nodes. The way Gini Index works is, if we select two items from a data set at random then they must be of the same class and probability for this is 1 if the data set is pure (containing only one class).
Each split will be chosen greedily by selecting the best split from the set of possible splits in order to maximize the information gain. The best split is selected by computing the Gini score for each feature and the feature having the maximum Gini score is the feature on which the node is split.

2. **Conditions to stop recursive tree construction**

There are 3 possible reasons when recursive tree construction is stopped at a node:
- The max depth specified is reached
- None of the split candidates have information gain which is greater than the minimum value set
- When the number of data points in a partition is below a minimum threshold.

3. **Partitioning of data by MLlib**

As observed from the syslog files, MLlib by default partitions this 8 GB labeled data into 123 partitions.
While running the model training on trees with a higher max depth, it would need to recursively go on for a longer time, thus elongating the overall training time. To reduce this time, we manually partitioned the data into 250 partitions and then went on with the training. The partitioning of the data did occur in 250 partitions and that was confirmed by the syslog files produced.

4. Parallelization by MLlib

Before training, MLlib samples the input data, and puts them in a certain number of bins.
To parallelize training, MLlib's implementation of decision trees partitions the input data across workers. During training, each node that needs to be split is put onto a queue. At each iteration, some number of nodes are pulled from the queue, depending on the memory. Then, Gini score is collected across the partitioned dataset. The Gini score for each feature is shuffled and sent back to the worker and the best split is decided. Finally, these best splits are sent back to the master, which grows the tree by creating more nodes, adding them into the queue as needed. MLlib trains multiple trees at once by adding nodes from all of them to the queue.

5. Pseudo code for Random Forest Training

S is the training set, F are the set of features and N is the number of trees in the forest

```
function RandomForest(S , F)
    H ← ∅
    for i ∈ 1, . . . , N do
            S(i) ← A bootstrap sample from S
            hi ← RandomizedTreeLearn(S(i) ,F)
            H ← H ∪ {hi}
     end for
     Return H
 end function


 function RandomizedTreeLearn(S, F)
      At each node:
            F ← very small subset of F
            Split on best feature in f
     Return The learned tree
  End function
```

Prediction by MLlib using Random Forests

The model which is trained by Random Forests is used for prediction. Each test record is run against this model. Trees that have been created will predict a class for that test record. The votes from all the trees are aggregated and the class having the majority of votes wins. This class becomes the label for the particular test record.
Below is the pseudo code for the same

F are the set of features and N is the number of trees

```
function predictByVoting(F)
        votes <- Map<Int, Double>
        for i ∈ 1 … N do
                prediction <- tree(i).predict(F)
                votes <- votes + tree(i).weight

        return (class with max votes)
```