E-Bird Data Set: Red-Winged Black Bird Prediction

Predicting the Sighting of a bird under Spark using Scala

CS6240
Team: Ayush Singh and Gautam Vashisht
Prof: Mirek Ridewald

CONTENTS

CS 6240: FINAL PROJECT REPORT

1. DESIGN DISCUSSION AND PSEUDO CODE

This project aims to predict the sighting of red winged black bird from the humongous set of other columns. Please find the below steps followed to build a machine learning model for prediction.

A. FEATURE ENGINEERING

We spent most of the time with analyzing and understanding the data. Given data contains many missing values ('?', 'X' or null). We replaced the missing value "?" and "null" with -1 and "X" with 1. There are few of the columns which needs to have more specific replacement for its missing values. "EFFORT_AREA_HA" contains some of the distances with negative values, distance can't be negative so we replace it with its mod value. We have added an additional column to provide weight to the "COUNT_TYPE" column having the value as P34. New column will have value '1' corresponding to P34 value else it will be '0'.

There are many sparse values given with parameters for nlcd, caus_prec and caus_snow, so we tried to run the model without these columns and there was not any change in accuracy. Therefore, we removed the list of above mentioned parameters for training our final model. Columns like Year and Country have constant value that won't provide any insight for prediction, so we removed them too. There are three duplicate columns given in data set that needs to be removed.

Count is given with the column to be predicted but as we are just interested in binary outcome, so we will replace the "Agelaius phoeniceus" column value with 1 for any value greater than 0, else 0.

2. MODEL SELECTION

For model selection, we came up with few of the suggestions like Logistic Regression, Support Vector Machines, Bagging and Random Forest Classifier to classify the output. Random Forest Classifier came as an opt choice for this

data set, as RFC works well with multi-dimensional dataset as it provides the ensemble model which decreases the variance of data by creating multiple decision trees.

- But since Radom Forest is best suited for parallelism
- Can handle large variance
- Parallel implementation in spark mllib is readily available

3. DATA PREPARATION FOR RANDOM FOREST CLASSIFIER MODEL

First of all, we have created the indexed variables for all the string variables using StringIndexer class and converted the remaining set of columns as Double data type. RFC requires one vector column for features which we have created through VectorAssembler class and the other is label which we cast to double type after dropping all rows with no label values.
Except last step, all test data goes through this processing as well.

4. TRAINING THE CLASSIFIER

We train the data on RFC model with numTrees as 50, depth as 20 and maximum number of bins as 1024. We started with 50 as number of trees and depth as 20 and ran the other model with different parameters. On increasing the numTrees beyond 50 doesn't lead to any increase in accuracy of prediction over test data, so we chose the optimal model with the above mentioned parameters.

2. MODEL PREDICTION

Model predicted the final output for unlabeled test data. Output of program will create the output file containing the Sampling_Event_Id, Prediction.

The steps above shows whole design structure of our project.

SPARK EXECUTION

We have implemented the Random Forest Classifier model using Spark and scala language. We are creating data frame for handling the data set in tabular format. Dataframes created under Spark are efficient to handle parallel execution. Random Forest Classifier model will divide the binary decision trees on random slave nodes and learn the data parallel.

We tried to run the program on m4 AWS instance.

AWS Instance	**Prediction on data Set**	**Time**	**Output**	**Test Error**
m4.2xlarge with 16 nodes	On split test set from labeled data.	6.20 hour and still running	Aborted the program as it was exceeding expected time limit.	NA
M4.2xlarge with 11 nodes	On unlabeled test set	3.48 hour and still running	Aborted the program as it was	NA

			exceeding expected time limit.	
r4.2xlarge	On split test set from labeled data.	2.28 hour	Successful	**0.12615123606398448**
r4.2xlarge	On unlabeled test set	1.18 hour	Successful	NA

We ran on r4.2x large as it provides the optimization for memory involving the processing of data involving huge computation and memory. Also cost wise, it was economically preferable too than m4 instance when compared with time of execution * number of workers.

SUMMARY

We got the final accuracy of 87.4% on whole labeled data. We split the 95% data as training and rest 5% as validation data. We got the best accuracy with 50 trees and 20 as the depth for random forest. We ran the final file on r4.2xlarge instance with 5 nodes and it returned the predicted output in 1 hour 18 minutes.