CS6240 - Bird Predictor

PROJECT REPORT

Sudeep Kulkarni

Paulomi Mahidharia

23rd April, 2017

Classification Model

We chose our classification model to be **Random Forest** for the following reasons:

According to the data mining discussions in class, we wanted to use an ensemble of decision trees to train the labeled data. Therefore, our strategy was to use an ensemble algorithm provided by Spark ML that enabled us to that. Random Forest Classification Model offers several features that set it apart from other ensemble algorithms:

- There is no need for cross-validation or a separate test set to get an unbiased estimate of the test set error. It is estimated internally, during the run.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.
- It has methods for balancing error in class population unbalanced data sets.
- Generated forests can be saved for future use on other data.
- It offers an experimental method for detecting variable interactions.

MODEL PARAMETERS:

We tried the following parameter settings for generating optimal model:

NUMBER OF FEATURES	MAX TREE DEPTH	NUMBER OF TREES	RUNNING TIME
14	10	25	17 min
14	10	50	20 min
14	12	70	23 min

The following observations were made while tuning the parameters:

- **Max tree depth less or more than 12 for any number of trees impacted accuracy**. So we found 12 as a stable value for tree depth.
- The above figures are **subject to 14 chosen features** for prediction. For the same number of trees and same max depth, **if the number of features are increased, it increases the running time**.
- Max number of trees directly affected accuracy and running time in the following way:

  NUM OF TREES $\propto$ ACCURACY & RUNNING TIME

  (Increase in the number of trees "**upto certain point**" increased the accuracy. Increase in the number of trees "definitely" increased the running time)

FEATURE SELECTION:

- After understanding the e-bird data and performing some research for the species in consideration, we shortlisted the following Features:

Candidates for Categorical features: LOC_ID, BCR, BAILEY_ECOREGION and OMERNIK_L3_ECOREGION

Candidates for Continuous features: LATITUDE, LONGITUDE, POP00_SQMI, HOUSING_DENSITY, HOUSING_PERCENT_VACANT, ELEV_GT, ELEV_NED, CAUS_TEMP_AVG, CAUS_TEMP_MIN, CAUS_TEMP_MAX, CAUS_PREC, CAUS_SNOW, NLCD2001_FS_C11_7500_PLAND to NLCD2001_FS_C95_7500_PLAND, DIST_FROM_FLOWING_BRACKISH, DIST_IN_FLOWING_BRACKISH, DIST_FROM_STANDING_BRACKISH, DIST_IN_STANDING_BRACKISH, DIST_FROM_WET_VEG_BRACKISH, DIST_IN_WET_VEG_BRACKISH

- After some trial and error, we decided to go with the following features (The above model parameter predictions were subject to these 14 features):

LATITUDE, LONGITUDE, POP00_SQMI, HOUSING_DENSITY, HOUSING_PERCENT_VACANT, ELEV_GT, ELEV_NED, BCR, OMERNIK_L3_ECOREGION, CAUS_TEMP_AVG, CAUS_TEMP_MIN, CAUS_TEMP_MAX, CAUS_PREC, CAUS_SNOW.

Pseudo Code

TRAINING JOB:

```
class ModelTrainer{
	main(){
		//Instatntiate Spark Configuration and Session, and load raw labeled data
		sparkConfig = initialize spark configuration object
		 rawLabeledData = load labeled csv file with required columns

		// Clean the data - replace "X" with 1, "?" with 0 and convert Strings to Doubles
		data = preprocess data and select required features with labels from
			rawLabeledData
		labelIndexer = index the labels for use with ML pipeline and fit on data

		//create a DataFrame of LabeledPoints from the data by using (Label and
		//Vector[Features])
		labeledPoints = data.mapPartitions(iterator[Row] => iterator[LabeledPoint])

		//Split the data and train the 80% training data using
```

```
            //Random Forest Classification
            trainingData = 80% of labeledPoints
            testData = 20% of labeledPoints
            randomForestClassifier = define a new RandomForestClassifier instance with
                                     required # of trees, depth and impurity parameters.
            pipeline = new Pipeline instance with the following stages: labelIndexer ->
                       randomForestClassifier

            //Train the classifier on 80% training data
            model = pipeline.fit(trainingData)

            //Evaluate model by transforming on 20% test data
            predictions = model.transform(testData)

            //Evaluate and print accuracy
            evaluator = define an evaluator instance to check accuracy of predicted labels
            accuracy = evaluator.evaluate(predictions)
            print(accuracy)
      }
}
```

PREDICTION JOB:

```
class Predictor{
      main(){
            //Instatntiate Spark Configuration and Session, and load raw unlabeled data
            sparkConfig = initialize spark configuration object
            rawUnlabeledData = load unlabeled csv file with required columns

            // Clean the data - "?" with 0 and convert Strings to Doubles
            data = preprocess data and select required features with labels from
                  rawUnlabeledData

            //create a DataFrame of UnlabeledPoints from the data by using (Label and
            //Vector[Features])
            labeledPoints = data.mapPartitions(iterator[Row] => iterator[LabeledPoint])

            //Calculate predictions on unlabeled data using the model from training job
            model = load the Random Forest model saved in the previous job
            predictions = model.transform(data)

            //Print out predictions for each sampling event ID in CSV format
            printToCSVFile(predictions("SAMPLING_EVENT_ID", "SAW_AGELAIUS_PHOENICEUS"))
```

```
        }
}
```

Pre-processing

The following pre-processing steps were performed before training and prediction jobs:

- We uncompressed the labeled data to see the schema and the types of values in each column. We noticed that a few columns were duplicated, namely: LOC_ID and SAMPLING_EVENT_ID. So in our pre-processing step that **read data as a CSV DataFrame** using Spark SQL, we **removed the duplicate columns**.
- We also decided to **replace the occurrences of "X" in the Red-Winged Blackbird with 1**, since the reference document mentions that "X" means that the bird was observed, but the count was unknown. For our prediction program, we care about only observed/not observed and not the specific count.
- We also **replaced "?" value in considered features' columns with 0** to compensate for missing or unknown values during prediction.
- We **cast all the feature values that were Strings to Double**, as the LabeledPoint data types expects an array of Double values as its Vector Features value.

Accuracy

maxDepth	NumTrees	NumFeatures	RUNNING TIME (10 Workers)	ACCURACY (Approx.)
10	25	53 (Some candidates for Categorical and all candidates for Continuous features)	1 hour 03 min	70.0930%
10	25	14 (Shortlisted features)	17 min	75.9582%
10	50	14 (Shortlisted features)	20 min	76.0717%
12	70	14 (Shortlisted features)	23 min	78.6645%
14	70	14 (Shortlisted features)	28 min	70.0202%

Running time and Speedup

We were using the **map** transformation to transform the raw DataFrame to the desired DataFrame of LabeledPoints. With that, we noticed the running times of following magnitudes:

NUMBER OF FEATURES	MAX TREE DEPTH	NUMBER OF TREES	RUNNING TIME
14	10	40	1 hours 27 min
14	10	25	40 min
14	10	20	28min

After digging into the behavior of **map** vs **mapPartitions**, it became clear that map is inefficient as compared to mapPartitions for expensive transformations on big data, as the former will execute the expensive function once *for each record* as opposed to the latter which executes it once *per partition*. The difference in running time was dramatic, as illustrated in the table below:

NUMBER OF FEATURES	MAX TREE DEPTH	NUMBER OF TREES	RUNNING TIME
14	10	25	17 min
14	10	50	20 min
14	12	70	23 min