Decision Tree in Pyspark

May 11, 2017

1 Introduction:

The objective of this report would be to find whether 2 records in the registry belong to a single person or not. It would be a classification problem where the label would be whether it is a match or not.

We have obtained the dataset from epidemiological cancer registry of the German state of North Rhine-Westphalia (Epidemiologisches Krebsregister NRW, http://www.krebsregister.nrw.de). This dataset has more than 5 million records and each record has phonetic equality metrics for 2 persons which would be used to find duplicates.

A decision tree classifier will be used for this project which will help us in predicting whether a record has a dupicate entry or not and can it be linked with any other record.

2 Motivation:

Data cleansing is a very important task where the data quality is of utmost importance. At times due to duplications we are not able to get exact figures regarding the analysis to be done. This is one of the challenges faced by disease registries, medical research networks, national databases etc. Record linkage provides methods for minimizing synonym and homonym errors thereby improving data quality.

We will utilize the power of apache spark and machine learning to find insights and help in solving the above problem. The solution to this problem can thereby improve data quality across various domains and hence help a lot of research projects and initiatives taken by institutions and governments.

3 Design:

The design of this project would be done in the following steps:

1. Understanding and defining the problem statement. 2. Loading the required packages in pyspark. 3. Creating a spark session and loading the data. 4. Doing some exploratory analysis. 5. Data restructuring and converting it to a defined format for model building. 6. Model building process. 7. Predictions and Evaluation process. 8. Conclusion and inferences.

3.1 Step 1: Problem Statement

Predicting whether a record can be linked with any other entry in the data or not.

3.2 Step 2: Loading the required packages in pyspark.

In the below code chunk we will load all the packages required in our further analysis.

```
In [1]: from pyspark.sql import SQLContext#To load the csv files as dataframes
        sqlContext = SQLContext(sc)
        from pyspark.ml.tuning import TrainValidationSplit#For train test split
        from pyspark.ml.classification import DecisionTreeClassifier#model builder function
        from pyspark.sql import SparkSession#to create spark session
        from pyspark.ml import Pipeline#Pipeline for creating a flow of processes to be done on
        from pyspark.ml.feature import  StringIndexer, VectorAssembler#Data converson functions
        from pyspark.ml.evaluation import BinaryClassificationEvaluator#Model evaluator function
        from pyspark.sql.types import StringType
        from pyspark.mllib.evaluation import MulticlassMetrics#For model evaluation
```

3.3 Step 3: Creating spark session and loading the data

```
In [2]: #Creating Spark session
        if __name__ == "__main__":
            spark = SparkSession \
                .builder \
                .appName("DecisionTree") \
                .getOrCreate()
```

```
In [3]: #We will be using spark sql context to load the csv file
        df = sqlContext.read.format('com.databricks.spark.csv')\
            .options(header='true', inferschema='true').load('/home/record_linkage.csv')

        #It was also found that our label feature was inferred as boolean hence we are explicitl
        #defining it as a character(string type) so that it can be integrated in the model
        df=df.withColumn("is_match", df.is_match.cast(StringType()))
```

3.4 Step 4: Exploratory analysis

Exploratory analysis will help in getting a deeper understanding of the dataset and also in making any changes to the data so that the model building process is a smooth flow.

```
In [4]: #Finding the number of rows in our dataset.
        df.count()
```

```
Out[4]: 5749132
```

```
In [5]: #Viewing the first 5 rows of our data.
        df.head(5)
```

```
Out[5]: [Row(id_1=37291, id_2=53113, cmp_fname_c1=3, cmp_fname_c2=3, cmp_lname_c1=1.0, cmp_lname
         Row(id_1=39086, id_2=47614, cmp_fname_c1=3, cmp_fname_c2=3, cmp_lname_c1=1.0, cmp_lname
         Row(id_1=70031, id_2=70237, cmp_fname_c1=3, cmp_fname_c2=3, cmp_lname_c1=1.0, cmp_lname
         Row(id_1=84795, id_2=97439, cmp_fname_c1=3, cmp_fname_c2=3, cmp_lname_c1=1.0, cmp_lname
         Row(id_1=36950, id_2=42116, cmp_fname_c1=3, cmp_fname_c2=3, cmp_lname_c1=1.0, cmp_lname
```

```
In [6]: #Removing the id columns as they are redundant in the model building process
        df=df.drop("id_1")
        df=df.drop("id_2")

        #getting column names of the data
        cols=df.columns
```

3.5 Step 5: Data restructuring and conversions

In the below code chunk we will structure the data according to the requirements of the spark machine learning model of decision tree. Spark takes data into vectors of features and labels, hence the below conversions has to be done.

```
In [7]: stages = [] # stages in our Pipeline
        # Convert label into label indices using the StringIndexer
        label_stringIdx = StringIndexer(inputCol = "is_match", outputCol = "label")
        stages += [label_stringIdx]

        ml_cols=cols[0:9]
        assembler = VectorAssembler(inputCols=ml_cols, outputCol="features")
        stages += [assembler]
```

In the below code chunk we will build a pipeline which will do all the structuring and conversions on the entire data.

```
In [8]: # Create a Pipeline.
        pipeline = Pipeline(stages=stages)
        # Run the feature transformations.
        #  - fit() computes feature statistics as needed.
        #  - transform() actually transforms the features.
        pipelineModel = pipeline.fit(df)
        df = pipelineModel.transform(df)

        # Keep relevant columns
        selectedcols = ["label", "features"] + cols
        df = df.select(selectedcols)
```

We can see in the below step that the data has been restructured and a vector has been created which can be used to build the model.

```
In [9]: df.head()
```

```
Out[9]: Row(label=1.0, features=DenseVector([3.0, 3.0, 1.0, 3.0, 1.0, 1.0, 1.0, 1.0, 0.0]), cmp_
```

3.6 Step 6: Model building

The model building step will comprise of splitting the data into train and test and building the model. We will use 75 percentage of data for training and 25 percentage for testing. For decision tree we will specify only one parameter for tuning called maxDepth. It will help in determining how deep the tree has to be grown. Deeper the trees, more is a chance of overfitting and hence the model may not perform well on the unseen data. We will therefore build a model with depth = 3.

```
In [10]: train, test = df.randomSplit([0.75, 0.25], seed=141)#Splitting into train and test

In [11]: # Create initial Decision Tree Model
         dt = DecisionTreeClassifier(labelCol="label", featuresCol="features", maxDepth=3)

         # Train model with Training Data
         dtModel = dt.fit(train)

In [12]: print "numNodes = ", dtModel.numNodes
         print "depth = ", dtModel.depth

numNodes =  15
depth =  3
```

3.7 Step 7: Predictions and model evauations

This step will involve predicting and testing the performance of model on unseen data. Since its a binary class problem we will have to use metrics such as accuracy, precision, recall.

```
In [13]: # Make predictions on test data using the Transformer.transform() method.
         predictions = dtModel.transform(test)
         # Evaluate model
         evaluator = BinaryClassificationEvaluator()
         evaluator.evaluate(predictions)

Out[13]: 0.9997059337157059
```

We can see that the accuracy which we are getting on unseen data is quite high. Also a point to note is that we are getting F1 score in the above step. Such a high F1-score states that the model performs equally well in predicting both the classes. And since it is a class imbalance problem where there are just 20900 observations belonging to class 1 and more than 5 million of class 0, the F1-score which we are getting is quite good. In the below step we will view all the other metrics and look into the confusion matrix as well.

```
In [14]: def print_metrics(predictions_and_labels):
             metrics = MulticlassMetrics(predictions_and_labels)
             print 'Precision of True ', metrics.precision(1)
             print 'Precision of False', metrics.precision(0)
             print 'Recall of True    ', metrics.recall(1)
             print 'Recall of False   ', metrics.recall(0)
             print 'F-1 Score         ', metrics.fMeasure()
             print 'Confusion Matrix\n', metrics.confusionMatrix().toArray()

         predictions_and_labels = predictions.select("prediction", "label").rdd \
         .map(lambda r: (float(r[0]), float(r[1])))

         print_metrics(predictions_and_labels)
```

```
Precision of True  0.995205753096
Precision of False 0.999801910034
Recall of True     0.946069122674
Recall of False    0.999983256966
F-1 Score          0.999785917545
Confusion Matrix


/home/meaww/spark-2.1.1-bin-hadoop2.7/python/pyspark/mllib/evaluation.py:262: UserWarning: Depre
  warnings.warn("Deprecated in 2.0.0. Use accuracy.")


[[  1.43340800e+06   2.40000000e+01]
 [  2.84000000e+02   4.98200000e+03]]
```

3.8 Step 8: Conclusion

We were succesfully able to find dupicates and record linkage was achieved with a good accuracy. This model will help in researches getting quality data and better and more accurate inferences can be drawn from our report.