Logistic regression in pyspark

May 9, 2017

1 Logistic Regression in Apache pyspark

1.1 Introduction:

Apache Spark is becoming a fast and efficient way to analyse large amounts of data. In our project we are using the python api of spark to analyse the HR analytics data obtained from Kaggle. This dataset has more than a million observations and the objective of our project would be to predict whether an employee will leave the company or not. It is a binary classification problem and independent features include satifaction levels, last evaluation, number of projects completed, average monthly hours dedicated, time spent, last appraisals etc.

In the below report the logistic regression model will be used to predict whether the employee will leave or not. The following report and codes will help in better understanding the steps and reasoning behind the algorithm.

1.2 Motivation:

These days companies find it difficult to find how long can an employee stay in their company. Any sudden leave or resignation can cause losses to the companies, the losses may have a less monetary impact but the impact on overall productivity of the company gets hampered as they lose in terms of time. Also finding alternate and reliable human resources can sometimes be a problem. If we are able to find out how likely is an employee about to leave then we can take necessary steps so as to minimize the losses. These steps will be decided by the company, but some of the common examples can be trying to retain the employee or start looking for new ones in advance.

This advance knowledge can be of great help for the company. Since we have data being collected all around, we must find ways to best utilize it. Therefore we collected a sample of past behaviour of employees and try to predict the likelihood of leaving.

1.3 Design:

The design of our model can be broken down into following steps:

1. Loading the required libraries. 2. Since we are using spark, we will have to initialize and create a spark session which will be responsible for managing and maintaining the resources to be allocated for the processes. 3. Loading the dataset and doing the required pre-processing and data conversions so that the data can be used in the model building process. 4. Splitting into train and test and building the model on train. 5. Model evaluations based on suitable metrics and finding how the model performs on unseen data.

1.3.1 Step 1: Loading libraries

```
In [12]: from pyspark.sql import SQLContext#For loading the csv files as dataframes
         sqlContext = SQLContext(sc)
         from pyspark.ml.tuning import TrainValidationSplit#For doing train test split
         from pyspark.ml.classification import LogisticRegression#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 OneHotEncoder, StringIndexer, VectorAssembler#Data conve
         from pyspark.ml.evaluation import BinaryClassificationEvaluator#Model evaluator functio
```

1.3.2 Step 2: Initializing the spark session

```
In [13]: if __name__ == "__main__":
             spark = SparkSession \
                 .builder \
                 .appName("LR") \
                 .getOrCreate()
```

1.3.3 Step 3: Loading data and pre-processing

```
In [14]: df = sqlContext.read.format('com.databricks.spark.csv').options(header='true', infersch
         cols=df.columns#getting column names of the data

In [15]: df.head(5)#Viewing the first 5 rows of the data get some idea about the data

Out[15]: [Row(satisfaction_level=0.38, last_evaluation=0.53, number_project=2, average_montly_ho
          Row(satisfaction_level=0.8, last_evaluation=0.86, number_project=5, average_montly_hou
          Row(satisfaction_level=0.11, last_evaluation=0.88, number_project=7, average_montly_ho
          Row(satisfaction_level=0.72, last_evaluation=0.87, number_project=5, average_montly_ho
          Row(satisfaction_level=0.37, last_evaluation=0.52, number_project=2, average_montly_ho

In [16]: cols#Viewing the column names of the data

Out[16]: ['satisfaction_level',
          'last_evaluation',
          'number_project',
          'average_montly_hours',
          'time_spend_company',
          'Work_accident',
          'left',
          'promotion_last_5years',
          'sales',
          'salary']

In [17]: #Identifying the categorical variables so that they can be encoded as numeric to be int
         catcols=["sales","salary"]
```

In the below steps we will encode the categorical columns into numeric. We will create extra columns which will represent the encoded features.

We will also define a flow so that the data can be inserted into a pipeline and the necessary pre-processing steps will be performed. The final result would be vector of features representing the indenpendent variables and the label vector which will be the output variable.

```
In [18]: stages=[]
         for c in catcols:
             strIndexer=StringIndexer(inputCol=c, outputCol=c+"Index")
             encoder=OneHotEncoder(inputCol=c+"Index", outputCol=c+"classVec")
             stages = stages + [strIndexer,encoder]

         label_idx=StringIndexer(inputCol="left",outputCol="label")
         stages = stages+[label_idx]

         numcols=["satisfaction_level","last_evaluation","number_project","average_montly_hours"
                  "Work_accident","promotion_last_5years"]

         assem_ip=map(lambda c: c+ "classVec", catcols) + numcols
         assembler=VectorAssembler(inputCols=assem_ip,outputCol="features")
         stages=stages+[assembler]

In [19]: #Doing all the process of stages in a pipeline
         pipeline=Pipeline(stages=stages)

         pl_ml=pipeline.fit(df)
         df=pl_ml.transform(df)

         sel_cols=["label","features"]+cols

         df=df.select(sel_cols)

In [20]: df.head()#Viewing the first row of the transformed data

Out[20]: Row(label=1.0, features=SparseVector(18, {0: 1.0, 9: 1.0, 11: 0.38, 12: 0.53, 13: 2.0,
```

1.3.4 Step 4: Splitting into train and test and model building process

```
In [21]: train, test = df.randomSplit([0.75, 0.25], seed=141)#Splitting into train and test
         lr = LogisticRegression(maxIter=100, regParam=0.3)#buiding a linear regression model
         # Fit the model on the train data
         lrModel = lr.fit(train)
```

1.3.5 Step 5: Model evaluation process

In this step we will test and evaluate the model. The best practice is to test its performance on unseen data. Since we have the test data which is completely unknown to the model, we will test the performance on this data. We will use a binary classification evaluator which will define the performance of the model. The performance of the logistic regression model can be derived by

the ROC curve. The value of this metric ranges from 0.5 to 1.0 and high values indicate that our model is doing a good job in dircriminating between the two categories which comprise our label.

```
In [30]: predictions = lrModel.transform(test)#Predicting on test data
         evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")#Calling the
         evaluator.evaluate(predictions)#Getting models performance

Out[30]: 0.8052813342286241
```

```
In [33]: # Print the coefficients and intercept for logistic regression
         print("Coefficients: %s" % str(lrModel.coefficients))
         print("Intercept: %s" % str(lrModel.intercept))

Coefficients: [0.0241288152874,0.0311345364118,0.0515342912541,-0.0259221737696,0.00071948554659
Intercept: -0.880250209332
```

1.3.6 Challenges Faced:

These were some of the the challenges which were faced during this project

1. Spark installation: Getting spark up and running can sometimes have issues, so to get a hassle free installation we installed it on a ubuntu machine. 2. Defining the problem: Since this was an open problem we had to find a problem and come up with its solution. 3. Getting data into the format which spark ML libraries can process: Spark requires data to be provided in a specific format and hence finding the correct format and assembling the data in vector format was also one of the challenges. 4. Model tuning and evaluation: Tuning the model to get good results was also one of the challenge. We had to increase the iterations count to get good results. Als evaluating it on test data and getting good results was also one of the challenge.

2 Conclusion:

We were successfully able to get good accuracy in predicting whether an employee will leave or not. The objective of the project was served and this model will be very helpful for the HR departments of the companies to get a productivity boost of the entire organization. They would be well aware about the employees behaviour and necesaary steps would have been taken in advance.