MINI-PROJECT REPORT

MACHINE LEARNING
IN BIG DATA USING PYSPARK IN HADOOP PLATFORM

Submitted By:
Jeeva Mary Loui
&
Leno Mathew

Metro College of Technology
August-04 2019

TABLE OF CONTENTS

Project part-1: Linear Regression Model: Google Travel Review Rating

Introduction

The dataset for the regression project was taken from the UCI database. The dataset contains 5456 data points based on Google reviews given to European tourist attractions in 24 categories. Google user rating ranges from 1 to 5 and average user rating per category is calculated.

The target value is the average ratings on monuments and how it gets affected by all other average ratings. As most tourists visit Europe to see the great monuments all other parameters will influence the entire experience of a tourist.

Data Source	UCI Machine Learning Repository https://archive.ics.uci.edu/ml/datasets/Tarvel+Review+Ratings
Variables Type	Numerical
No. of Observations	5456
No. of Variables	25
File type	CSV
Analysis Type	Linear Regression
Data Set Characteristics	Multivariate

Project Objectives

- To understand the factors that influences the overall experience of tourists who visit Europe and which factors are strongly correlated
- By analyzing strongly correlated fields government and private agencies can identify the key areas to invest so as to maximize customer experience and industry income

Following attributes are used to rate the tourist attractions:

1. Unique user id

2. Average ratings on churches
3. Average ratings on resorts
4. Average ratings on beaches
5. Average ratings on parks
6. Average ratings on theatres
7. Average ratings on museums
8. Average ratings on malls
9. Average ratings on zoo
10. Average ratings on restaurants
11. Average ratings on pubs/bars
12. Average ratings on local services
13. Average ratings on burger/pizza shops
14. Average ratings on hotels/other lodgings
15. Average ratings on juice bars
16. Average ratings on art galleries
17. Average ratings on dance clubs
18. Average ratings on swimming pools
19. Average ratings on gyms
20. Average ratings on bakeries
21. Average ratings on beauty & spas
22. Average ratings on cafes
23. Average ratings on view points
24. Average ratings on gardens
25. Average ratings on monuments

Implementation Details

Exploratory Data Analysis for Linear Regression:

Step 1: Importing Libraries

```
>>> import pyspark
>>> from pyspark.sql import SparkSession
>>> SpSession = SparkSession \
...     .builder \
...     .master("local") \
...     .appName("py_spark") \
...     .getOrCreate()
>>> SpContext = SpSession.sparkContext
```

Step 2: Importing the Dataset

```
>>> SpContext = SpSession.sparkContext
>>> GRData = SpContext.textFile("proj_jeeva_leno/google_review.csv")
>>> GRData.cache()
google_review_ratings.csv MapPartitionsRDD[1] at textFile at NativeMethodAccesso
>>> GRData.take(5)
```

Step 3: Data Cleanup

```
>>> def CleanupData(inputStr) :
...     attList=inputStr.split(",")
...
...     #Create a row with cleaned up and converted data
...     values= Row(    AVERAGE_RATINGS_CHURCHES=float(attList[0]),\
...                     AVERAGE_RATINGS_RESORTS=float(attList[1]), \
...                     AVERAGE_RATINGS_BEACHES=float(attList[2]), \
...                     AVERAGE_RATINGS_PARKS=float(attList[3]), \
...                     AVERAGE_RATINGS_THEATRES=float(attList[4]), \
...                     AVERAGE_RATINGS_MUSEUMS=float(attList[5]), \
...                     AVERAGE_RATINGS_MALLS=float(attList[6]), \
...                     AVERAGE_RATINGS_ZOO=float(attList[7]), \
...                     AVERAGE_RATINGS_RESTAURANTS=float(attList[8]), \
...                     AVERAGE_RATINGS_PUBS_BARS=float(attList[9]), \
...                     AVERAGE_RATINGS_LOCALSERVICES=float(attList[10]), \
...                     AVERAGE_RATINGS_BURGER_PIZZASHOPS=float(attList[11]), \
...                     AVERAGE_RATINGS_HOTELS_OTHER_LODGINGS=float(attList[12]), \
...                     AVERAGE_RATINGS_JUICE_BARS=float(attList[13]), \
...                     AVERAGE_RATINGS_ART_GALLERIES=float(attList[14]), \
...                     AVERAGE_RATINGS_DANCE_CLUBS=float(attList[15]), \
...                     AVERAGE_RATINGS_SWIMMINGPOOLS=float(attList[16]), \
...                     AVERAGE_RATINGS_GYMS=float(attList[17]), \
...                     AVERAGE_RATINGS_BAKERIES=float(attList[18]), \
...                     AVERAGE_RATINGS_BEAUTY_SPAS=float(attList[19]), \
...                     AVERAGE_RATINGS_CAFES=float(attList[20]), \
...                     AVERAGE_RATINGS_VIEWPOINTS=float(attList[21]), \
...                     AVERAGE_RATINGS_GARDENS=float(attList[22]), \
...                     AVERAGE_RATINGS_MONUMENTS=float(attList[23]), \
...                 )
...
...     return values
...
```

Step 4: Run Map for cleanup

```
>>> dataLines = GRData.filter(lambda x: "AVERAGE_RATINGS_MONUMENTS" not in x)
>>> dataLines.count()
5456
>>> GRMap = dataLines.map(CleanupData)
>>> GRMap.cache()
PythonRDD[4] at RDD at PythonRDD.scala:48
>>> GRMap.take(5)
19/08/03 20:03:19 WARN Executor: 1 block locks were not released by TID = 3:
[rdd_4_0]
[Row(AVERAGE_RATINGS_ART_GALLERIES=1.74, AVERAGE_RATINGS_BAKERIES=0.5, AVERAGE_RATINGS_BEACHES=3.6299999999
, AVERAGE_RATINGS_BURGER_PIZZASHOPS=1.6899999999999999, AVERAGE_RATINGS_CAFES=0.0, AVERAGE_RATINGS_CHURCHES=
99999999999997, AVERAGE_RATINGS_GARDENS=0.0, AVERAGE_RATINGS_GYMS=0.0, AVERAGE_RATINGS_HOTELS_OTHER_LODGINGS
AVERAGE_RATINGS_LOCALSERVICES=1.73, AVERAGE_RATINGS_MALLS=5.0, AVERAGE_RATINGS_MONUMENTS=0.0, AVERAGE_RATING
RATINGS_PARKS=3.6299999999999999, AVERAGE_RATINGS_PUBS_BARS=2.6400000000000001, AVERAGE_RATINGS_RESORTS=0.0
```

Step 5: Create a data frame

```
>>> GRDf = SpSession.createDataFrame(GRMap)
19/08/03 20:08:37 WARN Executor: 1 block locks were not released by TID = 4:
[rdd_4_0]
>>> GRDf.describe().show()
19/08/03 20:08:52 WARN Utils: Truncated the string representation of a plan since it was too large.
+-------+-----------------------------+-----------------------+-----------------------+-----------
|summary|AVERAGE_RATINGS_ART_GALLERIES|AVERAGE_RATINGS_BAKERIES|AVERAGE_RATINGS_BEACHES|AVERAGE_RATI
+-------+-----------------------------+-----------------------+-----------------------+-----------
|  count|                         5456|                   5456|                   5456|
|   mean|          2.2065725806451684|     0.9698112170087951|     2.4893310117302074|         1.6
| stddev|          1.7159606559930898|     1.2039715446781865|     1.2478154592661745|          1.
|    min|                          0.0|                    0.0|                    0.0|
|    max|                          5.0|                    5.0|                    5.0|
+-------+-----------------------------+-----------------------+-----------------------+-----------
```

Step 6: Find correlation between predictors and target

```
>>> for i in GRDf.columns:
...     if not( isinstance(GRDf.select(i).take(1)[0][0], unicode)) :
...         print( "Correlation to Average Ratings of Monuments for ", i, GRDf.stat.corr('AVERAGE_RATINGS_MONUMENTS',i))
...
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_ART_GALLERIES', -0.16376140300703801)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_BAKERIES', 0.089490440350135078)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_BEACHES', 0.11768226383988874)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_BEAUTY_SPAS', 0.14601436158147188)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_BURGER_PIZZASHOPS', -0.21090979853113551)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_CAFES', 0.34890494850280729)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_CHURCHES', 0.40701106870572629)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_DANCE_CLUBS', 0.050890603193548684)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_GARDENS', 0.45619162768654814)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_GYMS', 0.14052653466300014)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_HOTELS_OTHER_LODGINGS', -0.1573264530326916)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_JUICE_BARS', -0.29087506279305242)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_LOCALSERVICES', -0.12811043430965952)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_MALLS', -0.22297058102643605)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_MONUMENTS', 1.0)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_MUSEUMS', -0.080938659317529721)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_PARKS', 0.1736020850667512)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_PUBS_BARS', -0.21286727230491906)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_RESORTS', 0.07752955339936142)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_RESTAURANTS', -0.26843783590853998)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_SWIMMINGPOOLS', 0.13098144432621509)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_THEATRES', 0.12822728352533824)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_VIEWPOINTS', 0.47217675402249809)
('Correlation to Average Ratings of Monuments for ', 'AVERAGE_RATINGS_ZOO', -0.16800767000443295)
```

From the above results, its clear that all the independent variables are poorly correlated with the target variable. However, the most important correlations are average ratings on churches, cafes and view points and gardens

Step 7: Transform to a Data Frame for input to Machine Learning

```
>>> from pyspark.ml.linalg import Vectors
>>> def transformToLabeledPoint(row) :
...     lp = ( row["AVERAGE_RATINGS_MONUMENTS"], Vectors.dense([row["AVERAGE_RATINGS_CHURCHES"],\
...                          row["AVERAGE_RATINGS_RESORTS"], \
...                          row["AVERAGE_RATINGS_BEACHES"], \
...                          row["AVERAGE_RATINGS_PARKS"], \
...                          row["AVERAGE_RATINGS_THEATRES"], \
...                          row["AVERAGE_RATINGS_MUSEUMS"], \
...                          row["AVERAGE_RATINGS_MALLS"], \
...                          row["AVERAGE_RATINGS_ZOO"], \
...                          row["AVERAGE_RATINGS_RESTAURANTS"], \
...                          row["AVERAGE_RATINGS_PUBS_BARS"], \
...                          row["AVERAGE_RATINGS_LOCALSERVICES"], \
...                          row["AVERAGE_RATINGS_BURGER_PIZZASHOPS"], \
...                          row["AVERAGE_RATINGS_HOTELS_OTHER_LODGINGS"], \
...                          row["AVERAGE_RATINGS_JUICE_BARS"], \
...                          row["AVERAGE_RATINGS_ART_GALLERIES"], \
...                          row["AVERAGE_RATINGS_DANCE_CLUBS"], \
...                          row["AVERAGE_RATINGS_SWIMMINGPOOLS"], \
...                          row["AVERAGE_RATINGS_GYMS"], \
...                          row["AVERAGE_RATINGS_BAKERIES"], \
...                          row["AVERAGE_RATINGS_BEAUTY_SPAS"], \
...                          row["AVERAGE_RATINGS_CAFES"], \
...                          row["AVERAGE_RATINGS_VIEWPOINTS"], \
...                         row["AVERAGE_RATINGS_GARDENS"]]))
...     return lp
...

>>> GRLp = GRMap.map(transformToLabeledPoint)
>>> GRDF = SpSession.createDataFrame(GRLp,["label", "features"])
>>> GRDF.select("label","features").show(10)
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  0.0|[0.0,0.0,3.63,3.6...|
|  0.0|[0.0,0.0,3.63,3.6...|
|  0.0|[0.0,0.0,3.63,3.6...|
|  0.0|[0.0,0.5,3.63,3.6...|
|  0.0|[0.0,0.0,3.63,3.6...|
|  0.0|[0.0,0.0,3.63,3.6...|
|  0.0|[0.0,5.0,3.63,3.6...|
|  0.0|[0.0,5.0,3.63,3.6...|
|  0.0|[0.0,5.0,3.64,3.6...|
|  0.0|[0.0,5.0,3.64,3.6...|
+-----+--------------------+
only showing top 10 rows
```

Step 8: Perform Machine Learning

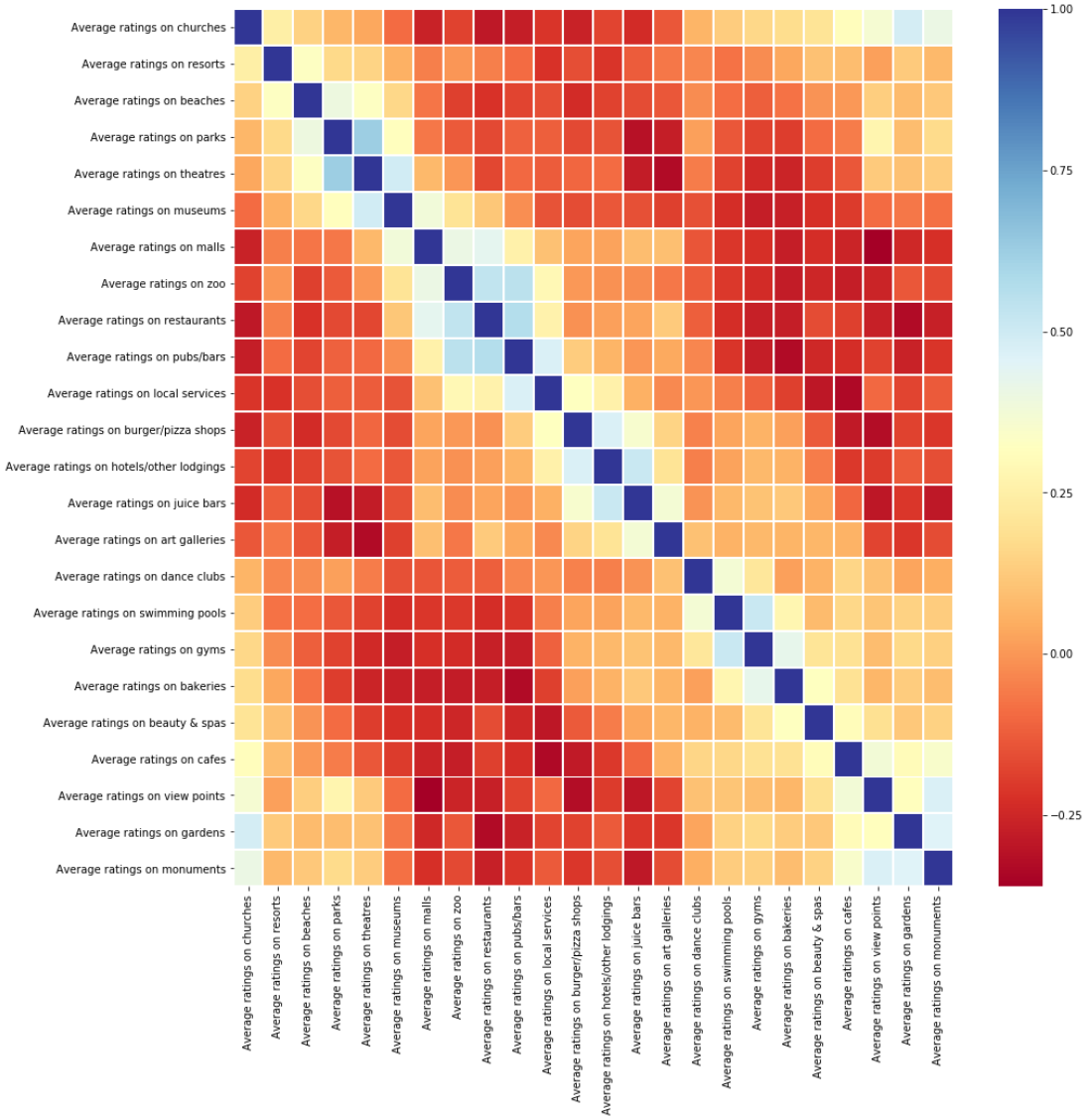
```
>>> (trainingData, testData) = GRDF.randomSplit([0.75, 0.25])
>>> trainingData.count()
4043
>>> testData.count()
1413
>>> from pyspark.ml.regression import LinearRegression
>>> lr = LinearRegression(maxIter=10)
>>> lrModel = lr.fit(trainingData)

>>> print("Coefficients: " + str(lrModel.coefficients))
Coefficients: [0.181920818316,-0.0152975276048,0.0123993896306,0.0139877217491,0.0
>>> print("Intercept: " + str(lrModel.intercept))
Intercept: 0.353155804784
>>> predictions = lrModel.transform(testData)
>>> predictions.select("prediction","label","features").show()
+--------------------+-----+--------------------+
|          prediction|label|            features|
+--------------------+-----+--------------------+
| 0.16358303724249149|  0.0|[0.0,0.0,1.42,1.4...|
| 0.16318375942981186|  0.0|[0.0,0.0,1.45,1.4...|
|-0.06465239139999085|  0.0|[0.0,0.0,1.51,1.4...|
|  0.1636097956487186|  0.0|[0.0,0.0,1.53,1.4...|
|  0.6250212498828356|  0.0|[0.0,0.0,3.63,3.6...|
|  0.6153655481115259|  0.0|[0.0,0.0,3.63,3.6...|
| 0.15190341494960513|  0.0|[0.0,0.51,1.41,1....|
|-0.07214603859922275|  0.0|[0.0,0.51,1.51,1....|
|  0.4360348429175349|  0.0|[0.0,0.51,5.0,3.6...|
| 0.40559660719875923|  0.0|[0.0,0.54,0.54,3....|
| 0.01798762794445191|  0.0|[0.0,0.54,1.41,1....|
|  0.5607694229206082|  0.0|[0.0,1.41,1.39,1....|
| 0.09039754368397768|  0.0|[0.0,1.41,1.4,1.4...|
|   0.564304440504259|  0.0|[0.0,1.42,1.41,1....|
|-0.04071125381798529|  0.0|[0.0,1.43,1.41,1....|
| 0.08273849647320275|  0.0|[0.0,1.43,1.43,1....|
|  0.6032738591831013|  0.0|[0.0,1.44,1.42,1....|
|  0.3767540334901607|  0.0|[0.0,1.52,1.41,1....|
| -0.0175371614561598|  0.0|[0.0,1.52,1.41,1....|
|  0.5492710979817308|  0.0|[0.0,5.0,3.63,3.6...|
+--------------------+-----+--------------------+
only showing top 20 rows
```

Step 9: R2 Score and Mean Square Error

```
>>> from pyspark.ml.evaluation import RegressionEvaluator
>>> evaluator = RegressionEvaluator(predictionCol="prediction", \
...                  labelCol="label",metricName="r2")
>>> evaluator.evaluate(predictions)
0.35692281778689083
>>> evaluator = RegressionEvaluator(predictionCol="prediction", \
...                  labelCol="label",metricName="mse")
>>> evaluator.evaluate(predictions)
1.07964291275
>>> print("Mean Square Error: " + str(evaluator.evaluate(predictions)))
Mean Square Error: 1.07964291275
```

Step 25: Heat map for the data



RESULTS AND DISCUSSIONS

- The correlation between the independent variable and the target variable is very weak
- A positive experience in one of the attractions can improve the overall travel experience of the travellers
- R2 value is low – 0.36, indicating that the model won't be of practical use
- Google Ratings of zero should be revaluated as ratings starts from 1

RECOMMENDATION

A direct influence of one parameter on the average rating of monuments seems to be low. Ratings of several attractions is given as zero while the minimum Google rating is 1, hence the data should be revaluated for better results

Project part -2: Build a Customer Churn Predictor Using Decision Tree Classification Method

INTRODUCTION

Customer churn, when a customer ends their relationship with a business, is one of the most basic factors in determining the revenue of a business. For a business it is important to know which of the customers are loyal and which are at risk of churning, and also its important to determine the factors that affect these decisions from a customer perspective. This code pattern explains how to build a machine learning model and use it to predict whether a customer is at risk of churning

Data Source	Kaggle.com https://www.kaggle.com/zagarsuren/telecom-churn-ibm-watson
No. of Observations	7043
No. of Variables	22
File type	CSV
Data Set Characteristics	Multivariate

Project Objectives

- Find the factors which are correlated to churn of customer in the telco company
- Run the decision tree classifier to build a model to predict the churn of customer.
- The report analyzes and gives findings about the Accuracy, F1 score,

Following attributes are used in the data:

1. ‘Customer_ID’ : Unique identification given to each customer.
2. 'gender': Sex of Customer.
3. 'SeniorCitizen': Age classification.
4. 'Partner': Whether customer has a partner in sharing connection or not.
5. 'Dependents': Whether customer has a dependant or not.
6. 'tenure'. Customer connection duration.
7. 'PhoneService': Whether customer use a phone service or not.
8. 'MultipleLines': Whether customer uses multiple lines for connection or not.

9. 'InternetService': Whether customer uses internet service or not.
10. 'OnlineSecurity': Whether customer uses online security or not.
11. 'OnlineBackup': Whether customer uses online backup or not.
12. 'DeviceProtection': Whether customer uses online backup facility or not.
13. 'TechSupport': Whether customer uses tech support or not.
14. 'StreamingTV': Whether customer stream TV or not.
15. 'StreamingMovies': Whether customer stream movies or not.
16. 'Contract': Whether customer is in a contract or not.
17. 'PaperlessBilling': Whether customer use paperless billing or not.
18. 'PaymentMethod':Payment mode of customer.
19. 'MonthlyCharges':Monthly charge of customer.
20. 'TotalCharges':Total Charges of Customer.
21. 'Churn':The target or Churn.

Detailed Description of the Project:

Step 1: Create a Spark Session in Hadoop Platform

```
>>> #Create a Spark Session
... SpSession = SparkSession \
...     .builder \
...     .master("local") \
...     .appName("py_spark") \
...     .getOrCreate()
>>>
>>> SpContext = SpSession.sparkContext
```

Step 2: Data Analysis and Create RDD

```
>>> # Infer the schema, and register the DataFrame as a table.
... churndf = SpSession.createDataFrame(churnMap)
19/08/05 02:02:07 WARN Executor: 1 block locks were not released by TID = 5:
[rdd_1_0]
>>> churndf.cache()
DataFrame[c_id: double, churn: string, contract: double, dependants: double, device_protection: double, gender: d
ultiple_lines: double, online_backup: double, online_security: double, paperlessbilling: double, partner: double,
zen: double, streaming TV: double, streaming movies: double, tech support: double, tenure: double, total charges:
```

```
>>> #Load the CSV file into a RDD
... dataLines = SpContext.textFile("leno_jeeva/classification.csv",)
>>> dataLines.cache()
leno_jeeva/classification.csv MapPartitionsRDD[1] at textFile at NativeMethodAccessorImpl.java:0
>>> dataLines.count()
7042
>>> dataLines.take(10)
```

Step3: Create Dataframe from RDD

```
>>> from pyspark.sql import Row
>>> #Create a Data Frame from the data
... parts = dataLines.map(lambda l: l.split(","))
>>> parts.take(5)
[[u'3962', u'1', u'0', u'0', u'0', u'34', u'1', u'0', u'0', u'2', u'0', u'2', u'0', u'0', u'0', u'1'
9', u'2', u'1', u'0', u'0', u'2', u'2', u'0', u'0', u'0', u'0', u'0', u'1', u'3', u'53.85', u'157',
', u'0', u'2', u'2', u'0', u'0', u'1', u'0', u'0', u'42.3', u'1400', u'No'], [u'6511', u'0', u'0', u
u'0', u'1', u'2', u'70.7', u'925', u'Yes'], [u'6551', u'0', u'0', u'0', u'0', u'8', u'1', u'2', u'1'
4', u'Yes']]
>>> churnMap = parts.map(lambda p: Row(c_id=float(p[0]), \
...                                    gender=float(p[1]), \
...                                    seniorcitizen=float(p[2]), \
...                                    partner=float(p[3]), \
...                                    dependants=float(p[4]), \
...                                    tenure=float(p[5]), \
...                                    phone_service=float(p[6]), \
...                                    multiple_lines=float(p[7]), \
...                                    internetservice=float(p[8]), \
...                                    online_security=float(p[9]), \
...                                    online_backup=float(p[10]), \
...                                    device_protection=float(p[11]), \
...                                    tech_support=float(p[12]), \
...                                    streaming_movies=float(p[13]), \
...                                    streaming_TV=float(p[14]), \
...                                    contract=float(p[15]), \
...                                    paperlessbilling=float(p[16]), \
...                                    payment_method=float(p[17]), \
...                                    monthly_charges=float(p[18]), \
...                                    total_charges=float(p[19]), \
...                                    churn=p[20]))
>>> churnMap.take(5)
>>> # Infer the schema, and register the DataFrame as a table.
... churndf = SpSession.createDataFrame(churnMap)
19/08/05 02:02:07 WARN Executor: 1 block locks were not released by TID = 5:
[rdd_1_0]
>>> churndf.cache()
```

Step 4: Label And Index the target using String Indexer

```
>>> from pyspark.ml.feature import StringIndexer
>>> stringIndexer = StringIndexer(inputCol="churn", outputCol="ind_churn")
>>> si_model = stringIndexer.fit(churndf)
[Stage 5:>                                                          (0 + 2) / 2]
>>> churnDf = si_model.transform(churndf)
>>>
>>> churnDf.select("churn","ind_churn").distinct().show()

+-----+---------+
|churn|ind_churn|
+-----+---------+
|   No|      0.0|
|  Yes|      1.0|
+-----+---------+

>>> churnDf.cache()
```

Step 5: Run Correlation Analysis to study the correlation of each variable to target

```
>>> #Find correlation between predictors and target
... for i in churnDf.columns:
...     if not(isinstance(churnDf.select(i).take(1)[0][0], unicode)) :
...         print("Correlation to Churn for ", i, \
...                     churnDf.stat.corr('ind_churn',i))
...
19/08/05 02:21:35 WARN Executor: 1 block locks were not released by TID = 215:
[rdd_28_0]
('Correlation to Churn for ', 'c_id', -0.017370854166055653)
19/08/05 02:21:35 WARN Executor: 1 block locks were not released by TID = 218:
[rdd_28_0]
19/08/05 02:21:36 WARN Executor: 1 block locks were not released by TID = 219:
[rdd_28_0]
('Correlation to Churn for ', 'contract', -0.39681281878276314)
19/08/05 02:21:36 WARN Executor: 1 block locks were not released by TID = 222:
[rdd_28_0]
('Correlation to Churn for ', 'dependants', -0.16428642349317318)
19/08/05 02:21:36 WARN Executor: 1 block locks were not released by TID = 225:
[rdd_28_0]
('Correlation to Churn for ', 'device_protection', -0.17823953505159018)
19/08/05 02:21:36 WARN Executor: 1 block locks were not released by TID = 228:
[rdd_28_0]
('Correlation to Churn for ', 'gender', -0.008699111952878081)
19/08/05 02:21:37 WARN Executor: 1 block locks were not released by TID = 231:
[rdd_28_0]
('Correlation to Churn for ', 'internetservice', -0.047398302300455095)
19/08/05 02:21:37 WARN Executor: 1 block locks were not released by TID = 234:
[rdd_28_0]
('Correlation to Churn for ', 'monthly_charges', 0.19328082483699671)
19/08/05 02:21:37 WARN Executor: 1 block locks were not released by TID = 237:
[rdd_28_0]
```

```
('Correlation to Churn for ', 'multiple_lines', 0.038043225272184726)
19/08/05 02:21:37 WARN Executor: 1 block locks were not released by TID = 240:
[rdd_28_0]
('Correlation to Churn for ', 'online_backup', -0.19544540326377796)
19/08/05 02:21:37 WARN Executor: 1 block locks were not released by TID = 243:
[rdd_28_0]
('Correlation to Churn for ', 'online_security', -0.28941218929307733)
19/08/05 02:21:37 WARN Executor: 1 block locks were not released by TID = 246:
[rdd_28_0]
('Correlation to Churn for ', 'paperlessbilling', 0.19191045823360778)
19/08/05 02:21:38 WARN Executor: 1 block locks were not released by TID = 249:
[rdd_28_0]
('Correlation to Churn for ', 'partner', -0.15037453354265298)
19/08/05 02:21:38 WARN Executor: 1 block locks were not released by TID = 252:
[rdd_28_0]
('Correlation to Churn for ', 'payment_method', 0.10709997701388023)
19/08/05 02:21:38 WARN Executor: 1 block locks were not released by TID = 255:
[rdd_28_0]
('Correlation to Churn for ', 'phone_service', 0.011689368449764582)
```

```
('Correlation to Churn for ', 'seniorcitizen', 0.15085772416175788)
19/08/05 02:21:38 WARN Executor: 1 block locks were not released by TID = 261:
[rdd_28_0]
('Correlation to Churn for ', 'streaming_TV', -0.038591774378182314)
19/08/05 02:21:39 WARN Executor: 1 block locks were not released by TID = 264:
[rdd_28_0]
('Correlation to Churn for ', 'streaming_movies', -0.03668019839212304)
19/08/05 02:21:39 WARN Executor: 1 block locks were not released by TID = 267:
[rdd_28_0]
('Correlation to Churn for ', 'tech_support', -0.28259500389894859)
19/08/05 02:21:39 WARN Executor: 1 block locks were not released by TID = 270:
[rdd_28_0]
('Correlation to Churn for ', 'tenure', -0.35238756744940508)
19/08/05 02:21:39 WARN Executor: 1 block locks were not released by TID = 273:
[rdd_28_0]
('Correlation to Churn for ', 'total_charges', 0.014445490546075227)
19/08/05 02:21:39 WARN Executor: 1 block locks were not released by TID = 276:
[rdd_28_0]
('Correlation to Churn for ', 'ind_churn', 1.0)
```

Step 6: Select variable with strong correlation score and transform rows to labeled points.

The features contract, tenure, tech_support, online_security, paperlessbilling, online_backup, monthly_charges, device_protection have correlation with target churn higher than 0.17 and the rest of features are eliminated.

```
>>> from pyspark.ml.linalg import Vectors
>>> def transformToLabeledPoint(row) :
...     lp = ( row["churn"], row["ind_churn"], \
...                 Vectors.dense([row["contract"],\
...                         row["tenure"], \
...                         row["tech_support"], \
...                         row["online_security"], \
...                         row["paperlessbilling"], \
...                         row["online_backup"], \
...                         row["monthly_charges"], \
...                         row["device_protection"]]))
...     return lp
...
...
>>> churnLp = churnDf.rdd.map(transformToLabeledPoint)
>>> churnLpDf = SpSession.createDataFrame(churnLp,["churn","label", "features"])
>>> churnLpDf.select("churn","label","features").show(10)
+-----+-----+--------------------+
|churn|label|            features|
+-----+-----+--------------------+
|   No|  0.0|[1.0,34.0,0.0,2.0...|
|  Yes|  1.0|[0.0,2.0,0.0,2.0,...|
|   No|  0.0|[1.0,45.0,2.0,2.0...|
|  Yes|  1.0|[0.0,2.0,0.0,0.0,...|
|  Yes|  1.0|[0.0,8.0,0.0,0.0,...|
|   No|  0.0|[0.0,22.0,0.0,0.0...|
|   No|  0.0|[0.0,10.0,0.0,2.0...|
|  Yes|  1.0|[0.0,28.0,2.0,0.0...|
|   No|  0.0|[1.0,62.0,0.0,2.0...|
|   No|  0.0|[0.0,13.0,0.0,2.0...|
+-----+-----+--------------------+
only showing top 10 rows

>>> churnLpDf.cache()
```

Step 7: Split dataframe to train and test to run machine learning model.

The dataframe is split to train and test using random split with a ratio of ¾ and ¼

```
>>> #Split into training and testing data
... (trainingData, testData) = churnLpDf.randomSplit([0.75, 0.25])
>>> trainingData.count()
5262
>>> testData.count()

1780
>>> testData.show()
+-----+-----+--------------------+
|churn|label|            features|
+-----+-----+--------------------+
|   No|  0.0|[0.0,1.0,0.0,0.0,...|
|   No|  0.0|[0.0,1.0,0.0,0.0,...|
|   No|  0.0|[0.0,1.0,0.0,0.0,...|
|   No|  0.0|[0.0,1.0,0.0,0.0,...|
|   No|  0.0|[0.0,1.0,0.0,0.0,...|
|   No|  0.0|[0.0,1.0,0.0,0.0,...|
|   No|  0.0|[0.0,1.0,0.0,0.0,...|
|   No|  0.0|[0.0,1.0,0.0,0.0,...|
|   No|  0.0|[0.0,1.0,0.0,0.0,...|
|   No|  0.0|[0.0,1.0,0.0,0.0,...|
|   No|  0.0|[0.0,1.0,0.0,0.0,...|
|   No|  0.0|[0.0,1.0,0.0,0.0,...|
|   No|  0.0|[0.0,1.0,0.0,2.0,...|
|   No|  0.0|[0.0,1.0,1.0,1.0,...|
|   No|  0.0|[0.0,1.0,1.0,1.0,...|
|   No|  0.0|[0.0,1.0,1.0,1.0,...|
|   No|  0.0|[0.0,1.0,1.0,1.0,...|
|   No|  0.0|[0.0,1.0,1.0,1.0,...|
|   No|  0.0|[0.0,1.0,1.0,1.0,...|
|   No|  0.0|[0.0,1.0,1.0,1.0,...|
+-----+-----+--------------------+
only showing top 20 rows
```

Step 8: Run the Decision Tree Classification Model

The decision tree classification model is build with the train set with number of nodes equal to seven and depth of Two.
The predictions are made with the trained model. And the predictions are displayed.

```
>>> from pyspark.ml.classification import DecisionTreeClassifier
>>> dtClassifer = DecisionTreeClassifier(maxDepth=2, labelCol="label",\
...                 featuresCol="features")
>>> dtModel = dtClassifer.fit(trainingData)
>>>
>>> dtModel.numNodes
7
>>> dtModel.depth
2
>>> #Predict on the test data
... predictions = dtModel.transform(testData)
>>> predictions.select("prediction","churn","label").show()
+----------+-----+-----+
|prediction|churn|label|
+----------+-----+-----+
|       1.0|   No|  0.0|
|       1.0|   No|  0.0|
|       1.0|   No|  0.0|
|       1.0|   No|  0.0|
|       1.0|   No|  0.0|
|       1.0|   No|  0.0|
|       1.0|   No|  0.0|
|       1.0|   No|  0.0|
|       1.0|   No|  0.0|
|       1.0|   No|  0.0|
|       1.0|   No|  0.0|
|       1.0|   No|  0.0|
|       1.0|   No|  0.0|
|       0.0|   No|  0.0|
|       0.0|   No|  0.0|
|       0.0|   No|  0.0|
|       0.0|   No|  0.0|
|       0.0|   No|  0.0|
|       0.0|   No|  0.0|
|       0.0|   No|  0.0|
|       0.0|   No|  0.0|
+----------+-----+-----+
only showing top 20 rows
```

Step 9: Classification Evaluation using metrics and create confusion matrix

From the pyspark evaluation library MulticlassificationEvaluator is run to see how successful the predictor model is.

```
>>> from pyspark.ml.evaluation import MulticlassClassificationEvaluator
>>> evaluator = MulticlassClassificationEvaluator(predictionCol="prediction", \
...                     labelCol="label",metricName="accuracy")
>>> print("Accuracy: " + str(evaluator.evaluate(predictions)))
Accuracy: 0.724719101124
>>>
>>> evaluator = MulticlassClassificationEvaluator(predictionCol="prediction", \
...                     labelCol="label",metricName="weightedPrecision")
>>> print("Precision: " + str(evaluator.evaluate(predictions)))
Precision: 0.777513323081
>>> evaluator = MulticlassClassificationEvaluator(predictionCol="prediction", \
...                     labelCol="label",metricName="weightedRecall")
>>> print("Recall: " + str(evaluator.evaluate(predictions)))
Recall: 0.724719101124
>>> evaluator = MulticlassClassificationEvaluator(predictionCol="prediction", \
...                     labelCol="label",metricName="f1")
>>> print("F1: " + str(evaluator.evaluate(predictions)))
F1: 0.739526502005
>>>
>>> #Draw a confusion matrix
... predictions.groupBy("label","prediction").count().show()
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  1.0|       1.0|  323|
|  0.0|       1.0|  360|
|  1.0|       0.0|  130|
|  0.0|       0.0|  967|
+-----+----------+-----+
```

CONCLUSION

The model to predict the churn of customer using decision tree classifier is built successfully with an accuracy of .72, Precision of .77, Recall of .72 and F1 score of .72.
The scores say the model can be employed with 72% accuracy. And thus, churn of customer can be predicted with this model.