Sparkify- Churn Prediction Using Spark

Customers are the crucial part of any business. Providing industry standard user experiences along with keeping cost optimal is itself challenging. In today's competitive world, it is important for businesses to move forward keeping their products relevant in order to satisfy their customers. One of the important metrics for subscription based services is member churn. Churn, the loss of customers, could have negative effect on the businesses. In order to forecast those customers who are most likely to churn and devise a business strategy to influence them not to leave.

In this Data Science Nanodegree Capstone Project, I explore how to perform churn prediction on a data set from a hypothetical business called "Sparkify", a subscription based music listening service. I have used pySpark, a technology for scalable data science that is widely used in industry today.

Problem statement:

Analysis of the users trend using the music streaming app 'Sparkify' to predict which users can stop using the app or churn.

Suggested Methodology:

CRISP-DM process is used for predicting churn. It involves business/data understanding, data preparation, modelling, evaluation, deployment.

Load and Clean Data:

- Created a spark session and specify the name of the app. I have named it 'Sparkify'.

```
In [2]: # create a Spark session
        spark = SparkSession \
            .builder \
            .appName("Sparkify") \
            .getOrCreate()
```

```
In [3]: spark
```

Out[3]: **SparkSession - in-memory**

SparkContext

[Spark UI]()

Version

`v2.4.3`

Master

`local[*]`

AppName

`Sparkify`

- Once the spark session is created, load the data using spark.

```
In [4]: df = spark.read.json("mini_sparkify_event_data.json")
```

```
In [5]: # Sample top 5 records
        df.show(5)
```

```
+---------------+---------+---------+------+-------------+--------+---------+-----+--------------------+------+--------+------
-------+---------+--------------------+------+-------------+--------------------+------+
|         artist|     auth|firstName|gender|itemInSession|lastName|   length|level|            location|method|    page| regis
tration|sessionId|                song|status|           ts|           userAgent|userId|
+---------------+---------+---------+------+-------------+--------+---------+-----+--------------------+------+--------+------
-------+---------+--------------------+------+-------------+--------------------+------+
|  Martha Tilston|Logged In|    Colin|     M|           50| Freeman|277.89016| paid|     Bakersfield, CA|   PUT|NextSong|153817
3362000|       29|           Rockpools|   200|1538352117000|Mozilla/5.0 (Wind...|    30|
|Five Iron Frenzy|Logged In|    Micah|     M|           79|    Long|236.09424| free|Boston-Cambridge-...|   PUT|NextSong|153833
1630000|        8|              Canada|   200|1538352180000|"Mozilla/5.0 (Win...|     9|
|    Adam Lambert|Logged In|    Colin|     M|           51| Freeman| 282.8273| paid|     Bakersfield, CA|   PUT|NextSong|153817
3362000|       29|   Time For Miracles|   200|1538352394000|Mozilla/5.0 (Wind...|    30|
|          Enigma|Logged In|    Micah|     M|           80|    Long|262.71302| free|Boston-Cambridge-...|   PUT|NextSong|153833
1630000|        8|Knocking On Forbi...|   200|1538352416000|"Mozilla/5.0 (Win...|     9|
|       Daft Punk|Logged In|    Colin|     M|           52| Freeman|223.60771| paid|     Bakersfield, CA|   PUT|NextSong|153817
3362000|       29|Harder Better Fas...|   200|1538352676000|Mozilla/5.0 (Wind...|    30|
+---------------+---------+---------+------+-------------+--------+---------+-----+--------------------+------+--------+------
-------+---------+--------------------+------+-------------+--------------------+------+
only showing top 5 rows
```

- Next step is to have a look at the columns in the data set and see how relevant they are for the analysis.

```
In [8]: df.printSchema()

root
 |-- artist: string (nullable = true)
 |-- auth: string (nullable = true)
 |-- firstName: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- itemInSession: long (nullable = true)
 |-- lastName: string (nullable = true)
 |-- length: double (nullable = true)
 |-- level: string (nullable = true)
 |-- location: string (nullable = true)
 |-- method: string (nullable = true)
 |-- page: string (nullable = true)
 |-- registration: long (nullable = true)
 |-- sessionId: long (nullable = true)
 |-- song: string (nullable = true)
 |-- status: long (nullable = true)
 |-- ts: long (nullable = true)
 |-- userAgent: string (nullable = true)
 |-- userId: string (nullable = true)
```

- For better explanation please follow below table.

Column name	Type	Description
Artist	Categorical	Artist name
Auth	Categorical	If user is logged in
First name	Categorical	User first name
Gender	Categorical	Gender information
Item In Session	Numerical	Number of items in a single session
Last name	Categorical	User last name
Length	Numerical	Length of a song
Level	Categorical	Paid member or free
Location	Categorical	Geographical location of a user
Method	Categorical	put/get http request
Page	Categorical	Event information
Registration	Numerical	Registration information
Status	Numerical	Web page codes such as 404 error
sessionId	Numerical	Identifier of current session
Song	Categorical	Song name
Ts	Numerical	Timestamp
userAgent	Categorical	Browser and device information
Userid	Numerical	Unique user id

- After dropping the rows the data set has 278,154 entries as can be seen in the figure below for total of 226 users.

In [12]:
```
df.where(df.userId == "").show(5)
df.where(df.userId == "").select("page").distinct().show()
df.select("userID").distinct().count()
```

```
+------+----------+---------+------+-------------+--------+------+-----+--------+------+-----+------------+---------+----+-----
-+-------------+---------+------+
|artist|      auth|firstName|gender|itemInSession|lastName|length|level|location|method| page|registration|sessionId|song|statu
s|           ts|userAgent|userId|
+------+----------+---------+------+-------------+--------+------+-----+--------+------+-----+------------+---------+----+-----
-+-------------+---------+------+
|  null|Logged Out|     null|  null|          100|    null|  null| free|    null|   GET| Home|        null|        8|null|   20
0|1538355745000|     null|      |
|  null|Logged Out|     null|  null|          101|    null|  null| free|    null|   GET| Help|        null|        8|null|   20
0|1538355807000|     null|      |
|  null|Logged Out|     null|  null|          102|    null|  null| free|    null|   GET| Home|        null|        8|null|   20
0|1538355841000|     null|      |
|  null|Logged Out|     null|  null|          103|    null|  null| free|    null|   PUT|Login|        null|        8|null|   30
7|1538355842000|     null|      |
|  null|Logged Out|     null|  null|            2|    null|  null| free|    null|   GET| Home|        null|      240|null|   20
0|1538356678000|     null|      |
+------+----------+---------+------+-------------+--------+------+-----+--------+------+-----+------------+---------+----+-----
-+-------------+---------+------+
only showing top 5 rows

+-------------------+
|               page|
+-------------------+
|               Home|
|              About|
|Submit Registration|
|              Login|
|           Register|
|               Help|
|              Error|
+-------------------+
```

Out[12]: 226

In [13]:
```
# Removing anonymous users
df = df.filter(df["userId"] != "")
df.count()
```

Out[13]: 278154

Exploring the Dataset:

Given the sample data set provided, we define "Churn" as the event that takes place when a user cancels their subscription, whether they are on the free or paid level of service.

```
# different types of authorisation
df.select("auth").distinct().show()
```

```
+---------+
|     auth|
+---------+
|Cancelled|
|Logged In|
+---------+
```

```
# different types of levels
df.select("level").distinct().show()
```

```
+-----+
|level|
+-----+
| free|
| paid|
+-----+
```

```
# number of unique users
df.agg(f.countDistinct('userId')).show()
```

```
+--------------------+
|count(DISTINCT userId)|
+--------------------+
|                 225|
+--------------------+
```

- Different types of authorisation, subscription level and unique users were found.
- Following are the findings on the churn dataset.

```
df.groupBy('churn').agg(f.count('userId')).show()
df.groupBy(['churn', 'gender']).agg(f.count('userId')).show()
```

```
+-----+-------------+
|churn|count(userId)|
+-----+-------------+
|    1|           52|
|    0|       278102|
+-----+-------------+

+-----+------+-------------+
|churn|gender|count(userId)|
+-----+------+-------------+
|    1|     F|           20|
|    0|     M|       123544|
|    1|     M|           32|
|    0|     F|       154558|
+-----+------+-------------+
```

Feature Engineering

I came up with following 7 features, to create our prediction model.

1. Rounded number of days from registration

```
+------+-------------+
|userId|days_since_reg|
+------+-------------+
|100010|      12587.0|
|200002|      23514.0|
|   125|        781.0|
|    51|      26450.0|
|   124|     477451.0|
|     7|       9208.0|
|    54|     303053.0|
|    15|      57096.0|
|   155|      12167.0|
|   132|     115510.0|
|   154|       1168.0|
|100014|      17884.0|
|   101|      99228.0|
|    11|      86704.0|
|   138|     106760.0|
|300017|     180125.0|
|    29|     131534.0|
|    69|      60971.0|
|100021|      15463.0|
|    42|     158175.0|
+------+-------------+
only showing top 20 rows
```

2. Number of songs played

```
+------+--------+
|userId|numSongs|
+------+--------+
|100010|     275|
|200002|     387|
|   125|       8|
|    51|    2111|
|   124|    4079|
|     7|     150|
|    54|    2841|
|    15|    1914|
|   155|     820|
|   132|    1928|
|   154|      84|
|100014|     257|
|   101|    1797|
|    11|     647|
|   138|    2070|
|300017|    3632|
|    29|    3028|
|    69|    1125|
|100021|     230|
|    42|    3573|
+------+--------+
only showing top 20 rows
```

3. Average Number of Songs per session

userID	avg_sess_songs
100010	39.285714285714285
200002	64.5
125	8.0
51	211.1
124	145.67857142857142
7	21.428571428571427
54	81.17142857142858
15	136.71428571428572
155	136.66666666666666
100014	42.833333333333336
132	120.5
154	28.0
101	179.7
11	40.4375
300017	59.540983606557376
138	138.0
29	89.05882352941177
69	125.0
100021	46.0
42	87.14634146341463

only showing top 20 rows

4. Total play time as per user

userId	totalPlayTime
100010	66940.89735000003
200002	94008.87593999993
125	2089.1131000000005
51	523275.8428000004
124	1012312.0927899999
7	38034.08710000002
54	711344.9195400011
15	477307.60581000015
155	198779.2919
132	483118.9038399997
154	20660.023910000007
100014	67703.47208000004
101	447464.0146699989
11	159669.96303999983
138	512449.8827599989
300017	897406.9802100015
29	754517.5625700009
69	286064.0256399999
100021	57633.17563999999
42	881792.9661300007

only showing top 20 rows

5. Average number of clicks per user

```
+------+------------------+
|userId|         avgClicks|
+------+------------------+
|200002|33.857142857142854|
|100010| 34.63636363636363|
|   125|              2.75|
|    51|164.26666666666668|
|   124|344.64285714285717|
|     7|15.461538461538462|
|    54|180.89473684210526|
|    15|162.71428571428572|
|   155|              66.8|
|   132|             144.0|
|100014|23.846153846153847|
|   154|             14.75|
|    11| 49.88235294117647|
|   101|119.38888888888889|
|   138|          154.3125|
|300017| 316.2857142857143|
|    29|211.94117647058823|
|100021| 24.53846153846154|
|    69|            83.875|
|    42|          266.0625|
+------+------------------+
only showing top 20 rows
```

6. Gender

```
+------+-----------+
|userId|gender_male|
+------+-----------+
|100010|          0|
|200002|          1|
|   125|          1|
|    51|          1|
|   124|          0|
+------+-----------+
only showing top 5 rows
```

7. Subscription Paid level

```
+------+----------+
|userId|paid_level|
+------+----------+
|100010|         0|
|200002|         0|
|   125|         0|
|    51|         1|
|   124|         1|
+------+----------+
only showing top 5 rows
```

Modelling

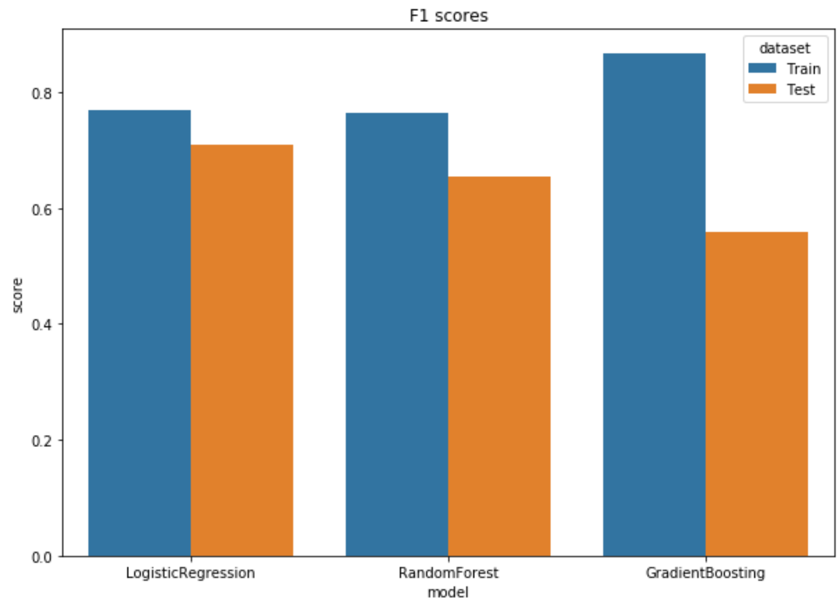
After creating the features, I have aggregated all of them at userId level. Then, I have splitted the dataset into training and validation set.

- I split the data into 2 parts : 70% for training and 30% for validation
- Then I used the Cross-Validator method because it helps us try out various hyper-parameters and automatically select the best ones.
- I have trained 3 models : Logistic Regression, Gradient Boosting Trees and Random Forest Classifier.

Evaluation

Among the 3 models, following are the detailed metrics report for different classifier model.

model	f1_train	f1_test	precision_train	precision_test	recall_train	recall_test	accuracy_train	accuracy_test
LogisticRegression	0.768818	0.709235	0.753084	0.753785	0.81592	0.776316	0.81592	0.776316
RandomForest	0.765804	0.654135	0.86291	0.58	0.835821	0.75	0.835821	0.75
GradientBoostedTree	0.866467	0.559649	0.883641	0.541596	0.885572	0.578947	0.885572	0.578947



As per F1 score evaluation metric, LogisticRegression is the optimal model to predict churn because it has the highest F1 score.

Improvements

Improvements to the above results could be possible by applying e.g. the following approaches:

- Creating more number of features
- Training on a larger dataset, by deploying on cloud (AWS, IBM cloud)
- Try more models, e.g. Deep Learning models
- Try to find an additional data source which could be helpful for the considered task