Retain your customer!

Naveen Setia
Feb 12 · 7 min read

Leverage datascience to detect unhappy customers before they leave.

Representation image for customer churn

A. PROJECT OVERVIEW

I got an opportunity to work on the customer churn problem which is the final project of the “Udacity Data Science Nanodegree program”. We have been provided a virtual company called ‘Sparkify’ which is a music app and it offers paid and free listening service. The customer can switch between either service or they can cancel their subscription at any time.

This provided dataset contains two months of ‘Sparkify’ user behavior log. The log contains some basic information about the user as well as information about a single action.

The given dataset is large i.e. 12GB, thus the typical tools for analysis and machine learning will not be useful here as it will not fit in the most of the local computer's memory. The ideal and safe way to perform such analysis is to do it using Big Data tools like Apache Spark.

A.1 Why Apache Spark?

- As mentioned above, provided dataset shared as part of this project is over 12GB so it is virtually impossible to load this into a typical development machine.
- The Spark distributed analysis capability makes it relatively easy to handle these large datasets and reduce the time it takes to analyze the data and train models.

A.2 How to save costs?

- As highlighted above, it is not possible to load over 12GB in a typical local development machine. So, we can use a subset of data to explore and build a model in a local development environment.
- Once it is fully tested it can be used in a cloud-based development (e.g. on Amazon AWS, Microsoft Azure, etc.) environment to run the script on full 12GB data.

A.3 What is the outcome of the project?

- As highlighted above, the business ask is to predict whether a customer will churn or not.
- From the machine learning perspective, it is a binary classification problem i.e. a binary outcome whether a customer will churn or not.

B. PROBLEM STATEMENT

The purpose of the project is to find the traits/characteristics/features of churned users from the behavioural data of these users, and take appropriate measures to retain the potential lost users as early as possible.

B.1 What is the customer/subscriber churn (aka Customer Attrition)?

It occurs when customers/subscribers stop doing business with a company/service. It is sometimes also known as customer attrition.

B.2 Why it is important?

- Earning money from a new customer is a result of a lengthy process which includes taking a customer through the entire sales funnel.
- Given we have already earned the trust and loyalty of the existing customer, it is much less expensive to retain an existing customer.
- As a business, we should be able to assess customer churn in a given period of time to avoid impedes in growth.

B.3 How should we define customer churn for this project?

I used the *Cancellation Confirmation* events to define your churn, which happens for both paid and free users.

C. METRICS

C.1 How will you decide whether the trained customer attrition model is good or not?

- Normally, for a binary classification problem, it is fine to use accuracy (i.e. no. of correctly predicted outcomes / total outcomes).
- However, in the current scenario, it is not a good measure given the provided data set is imbalanced i.e. only 22% of the records show up as positive for churn. In other words, where only a minority of cases are positive cases.
- In the case of imbalanced class, F1 is a good measure to evaluate the model.
- How to calculate the F1 measure:

$$F1 = 2 * precision * recall / (precision + recall)$$

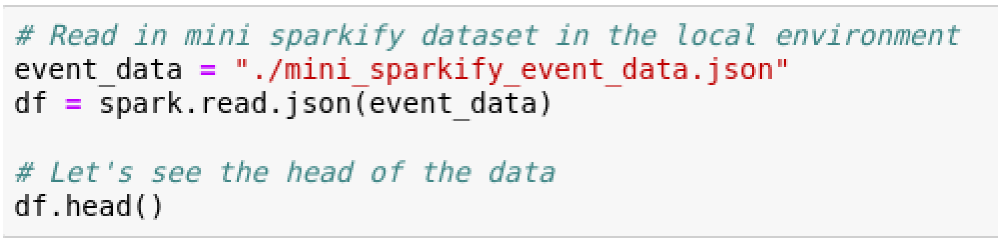
F1 Measure Formula

- **Precision** is the total number of actual positives out of the total number of positives identified (correctly and incorrectly).
- **The recall** is the total number of positives correctly identified out of all the true positives.

D. DATA EXPLORATION

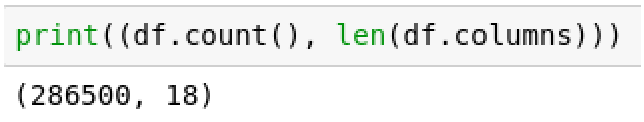
D.1 Explore a subset of the data

- As highlighted earlier, we are using a subset of the data for faster experimentation and cost perspective.
- It works very well as long as the subset holds true for the large dataset.
- Once the script is tested locally, we can use a cloud-based spark environment for analysis on the full data set.



```
# Read in mini sparkify dataset in the local environment
event_data = "./mini_sparkify_event_data.json"
df = spark.read.json(event_data)

# Let's see the head of the data
df.head()
```

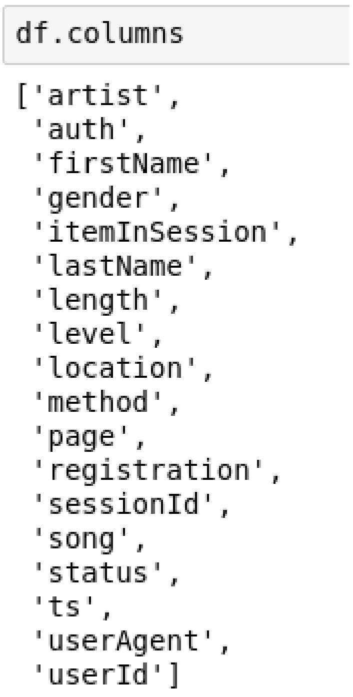
Load mini dataset into the spark session



```
print((df.count(), len(df.columns)))
```

```
(286500, 18)
```

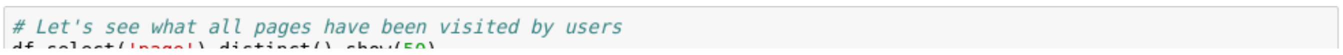
Check the number of records and columns in the dataset. As we can see, the subset of data has 2,86,500 records and 18 columns.



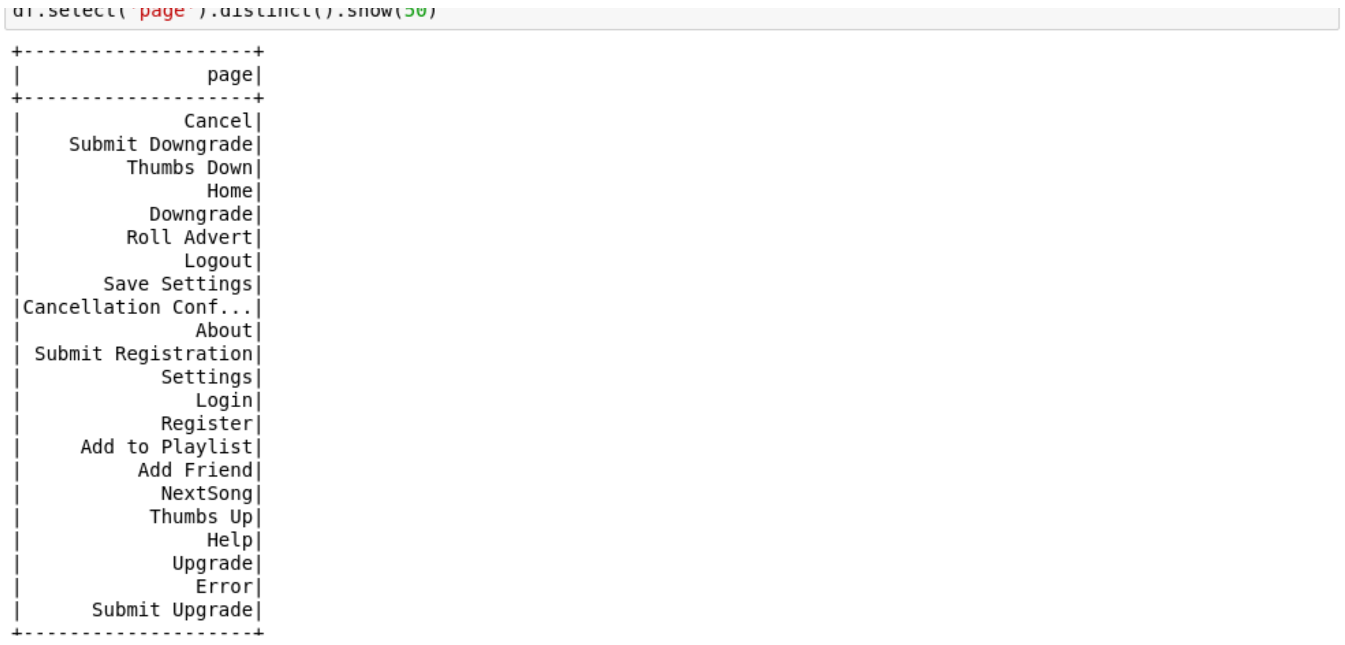
```
df.columns
```

```
['artist',
 'auth',
 'firstName',
 'gender',
 'itemInSession',
 'lastName',
 'length',
 'level',
 'location',
 'method',
 'page',
 'registration',
 'sessionId',
 'song',
 'status',
 'ts',
 'userAgent',
 'userId']
```

List of columns which are there in the dataset



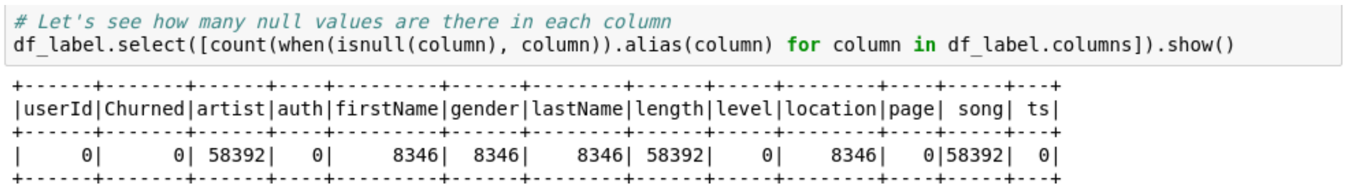
```
# Let's see what all pages have been visited by users
df.select('page').distinct().show(50)
```



```
df.select('page').distinct().show(30)
```

```
+--------------------+
|                page|
+--------------------+
|              Cancel|
|    Submit Downgrade|
|         Thumbs Down|
|                Home|
|           Downgrade|
|         Roll Advert|
|              Logout|
|       Save Settings|
|Cancellation Conf...|
|               About|
| Submit Registration|
|            Settings|
|               Login|
|            Register|
|     Add to Playlist|
|          Add Friend|
|            NextSong|
|           Thumbs Up|
|                Help|
|             Upgrade|
|               Error|
|      Submit Upgrade|
+--------------------+
```

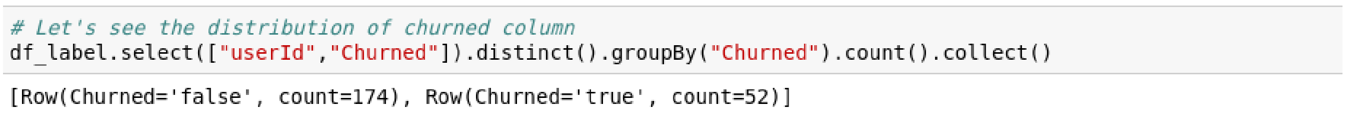
Distinct pages visited by users



```
# Let's see how many null values are there in each column
df_label.select([count(when(isnull(column), column)).alias(column) for column in df_label.columns]).show()
```

```
+------+-------+------+----+---------+------+--------+------+-----+--------+----+-----+---+
|userId|Churned|artist|auth|firstName|gender|lastName|length|level|location|page| song| ts|
+------+-------+------+----+---------+------+--------+------+-----+--------+----+-----+---+
|     0|      0| 58392|   0|     8346|  8346|    8346| 58392|    0|    8346|   0|58392|  0|
+------+-------+------+----+---------+------+--------+------+-----+--------+----+-----+---+
```

Checking missing value in fields



```
# Let's see the distribution of churned column
df_label.select(["userId","Churned"]).distinct().groupBy("Churned").count().collect()
```

```
[Row(Churned='false', count=174), Row(Churned='true', count=52)]
```

E. DATA PRE-PROCESSING

After analysis, I decided to create ***five*** features for model training:

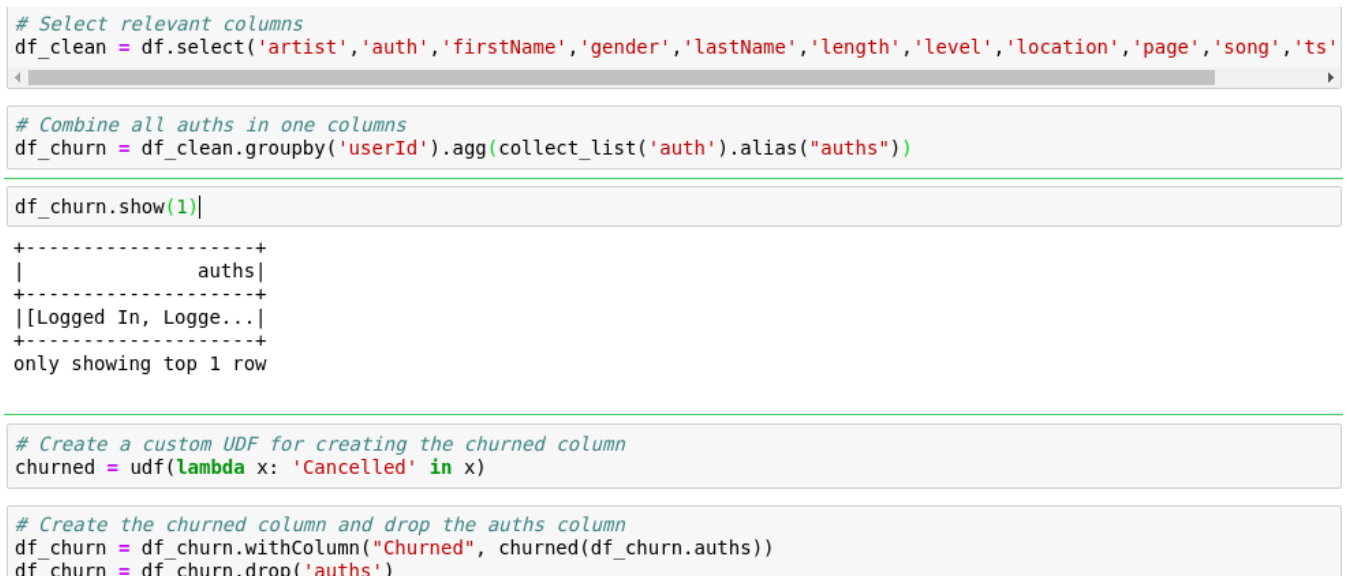
- `Thumbs Up:` The average number of 'thumbs up' given per song
- `Thumbs Down:` The average number of 'thumbs down' given per song
- `Songs Played:` The total number of songs played
- `Number of Days:` The number of days the user had been members of the music service
- `Songs Per Hour:` How many songs in an hour was played

Code for calculating features:

> *songs_played = df_label.where(col('song')!='null').groupby("userId").agg(count(col('song')).alias('SongsPlayed')).orderBy('userId')*
>
> *thumbs_up = df_label.where(df_label.page=='Thumbs Up').groupby("userId").agg(count(col('page')).alias('ThumbsUp')).orderBy('userId')*
>
> *thumbs_down = df_label.where(df_label.page=='Thumbs Down').groupby("userId").agg(count(col('page')).alias('ThumbsDown')).orderBy('userId')*
>
> *days = df_label.groupby('userId').agg(((max(col('ts')) — min(col('ts')))/86400000).alias("Days"))*
>
> *udf_songs_played_per_hour = udf(lambda songsPlayed, numberOfDays: float(songsPlayed)/float((numberOfDays*24)), FloatType())*
>
> *df_features = df_features.withColumn("SongsPerHour", udf_songs_played_per_hour(df_features.SongsPlayed, df_features.Days))*

Create the target variable i.e. 'Churn'

- Select the relevant columns into a cleaned data frame (i.e. df_clean)
- Created a user-defined function (UDF) i.e. churned to check if the Cancelled value exists in the auths column
- Once the churn column has been created, I joined it with a cleaned data frame (i.e. df clean) based on userId.



```
# Select relevant columns
df_clean = df.select('artist','auth','firstName','gender','lastName','length','level','location','page','song','ts'
```

```
# Combine all auths in one columns
df_churn = df_clean.groupby('userId').agg(collect_list('auth').alias("auths"))
```

```
df_churn.show(1)
```

```
+--------------------+
|               auths|
+--------------------+
|[Logged In, Logge...|
+--------------------+
only showing top 1 row
```

```
# Create a custom UDF for creating the churned column
churned = udf(lambda x: 'Cancelled' in x)
```

```
# Create the churned column and drop the auths column
df_churn = df_churn.withColumn("Churned", churned(df_churn.auths))
df_churn = df_churn.drop('auths')
df_churn = df_churn.drop('auth')
```

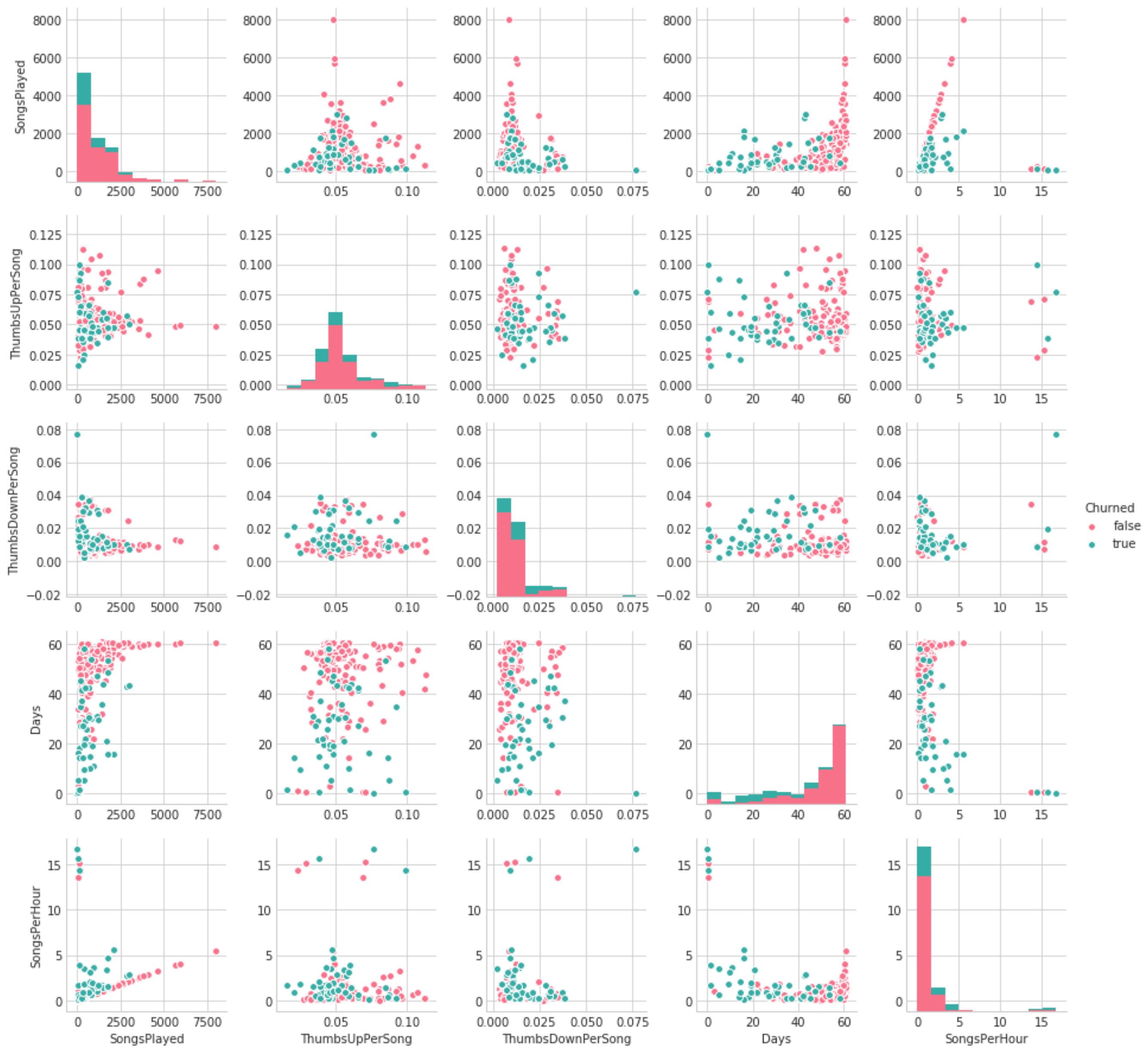
```
# Join the churn df with the clean df
df_label = df_churn.join(df_clean,'userId')
```

```
# Let's see the distribution of churned column
df_label.select(["userId","Churned"]).distinct().groupBy("Churned").count().collect()
```

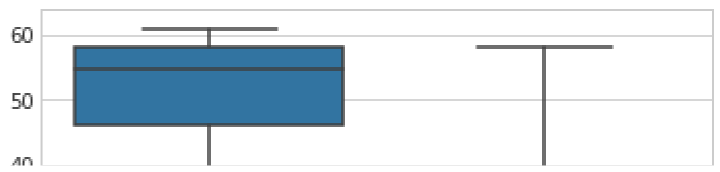
```
[Row(Churned='false', count=174), Row(Churned='true', count=52)]
```

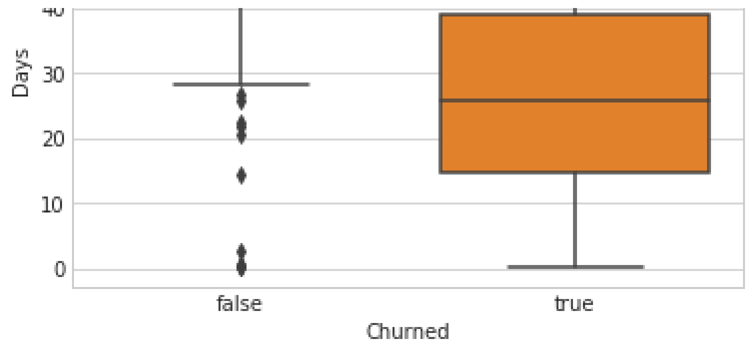
Distribution of Chuned

F. VISUALISATION



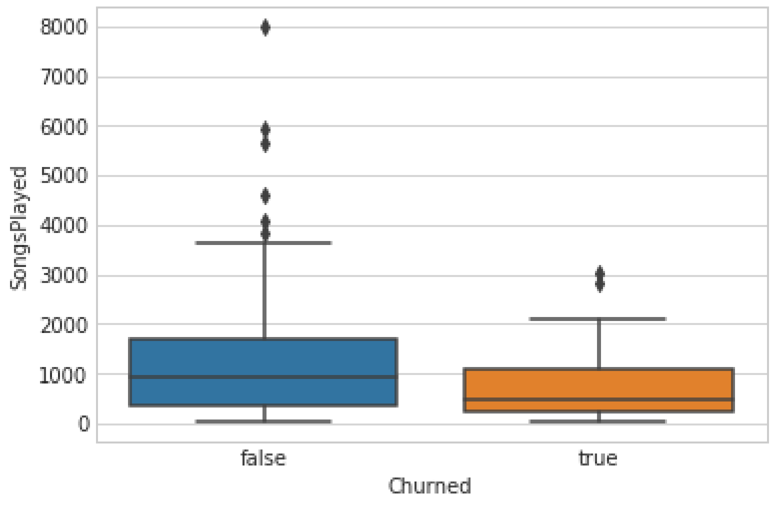
PairPlot of Important Features





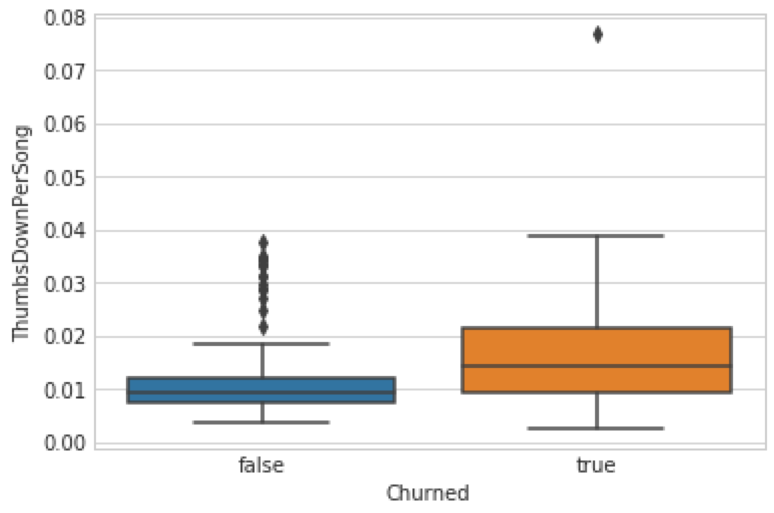
Box Blot for Days

It’s evident from the above graph that users who uses the service for a shorter time are more likely to churn compared to continuing users.



Box Plot for SongsPlayed

It’s evident from the above graph that users has played less songs is more likely to churn compared to use who has played more songs.



Box Plot for ThumbsDownPerSong

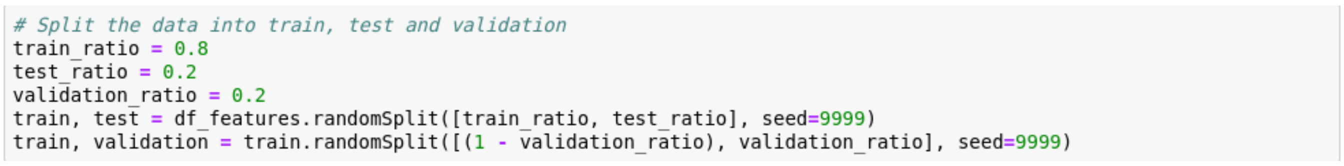
It’s evident from the above graph that song which has got more thumbs down is more likely to churn compared to user who has less thumbs down.

G. IMPLEMENTATION

Join all features for model training in a single vector and standardise it

> *assembler = VectorAssembler(inputCols=[“SongsPlayed”, “ThumbsUpPerSong”, “ThumbsDownPerSong”, “Days”, “SongsPerHour”], outputCol=”FeatureVector”)*
> *df_features = assembler.transform(df_features)*
>
> *scaler = StandardScaler(inputCol=”FeatureVector”, outputCol=”ScaledFeatures”, withStd=True)*
> *scaler_tranformer = scaler.fit(df_features)*
> *df_features = scaler_tranformer.transform(df_features)*

Split the data into train, test and validation



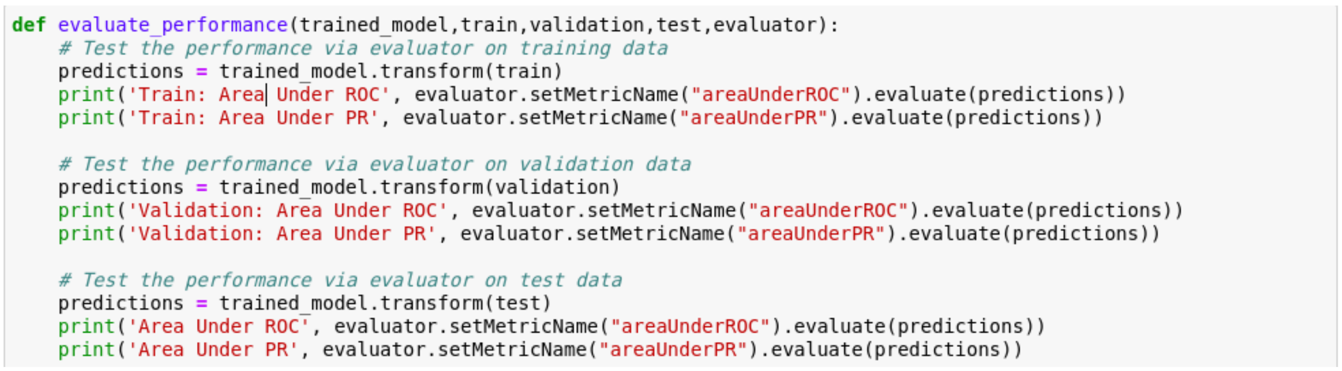
```python
# Split the data into train, test and validation
train_ratio = 0.8
test_ratio = 0.2
validation_ratio = 0.2
train, test = df_features.randomSplit([train_ratio, test_ratio], seed=9999)
train, validation = train.randomSplit([(1 - validation_ratio), validation_ratio], seed=9999)
```

Split the data into train, test and validation

Train the model

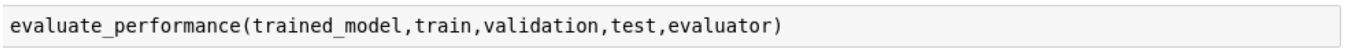
> *model = RandomForestClassifier(featuresCol = ‘FeatureVector’, labelCol = ‘label’, numTrees=100)*
>
> *trained_model = model.fit(train)*

Evaluate the Performance of the Model



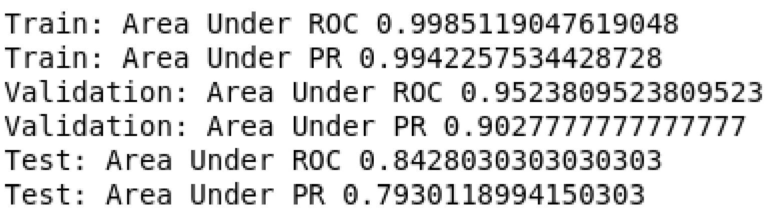
```python
def evaluate_performance(trained_model,train,validation,test,evaluator):
    # Test the performance via evaluator on training data
    predictions = trained_model.transform(train)
    print('Train: Area Under ROC', evaluator.setMetricName("areaUnderROC").evaluate(predictions))
    print('Train: Area Under PR', evaluator.setMetricName("areaUnderPR").evaluate(predictions))

    # Test the performance via evaluator on validation data
    predictions = trained_model.transform(validation)
    print('Validation: Area Under ROC', evaluator.setMetricName("areaUnderROC").evaluate(predictions))
    print('Validation: Area Under PR', evaluator.setMetricName("areaUnderPR").evaluate(predictions))

    # Test the performance via evaluator on test data
    predictions = trained_model.transform(test)
    print('Area Under ROC', evaluator.setMetricName("areaUnderROC").evaluate(predictions))
    print('Area Under PR', evaluator.setMetricName("areaUnderPR").evaluate(predictions))
```

Function to evaluate the performance



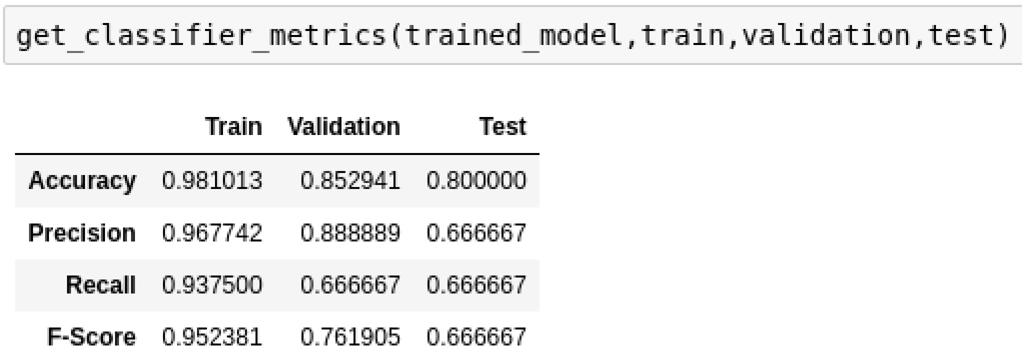
```
evaluate_performance(trained_model,train,validation,test,evaluator)
```

Function call to evaluate the performance



```
Train: Area Under ROC 0.9985119047619048
Train: Area Under PR 0.9942257534428728
Validation: Area Under ROC 0.9523809523809523
Validation: Area Under PR 0.9027777777777777
Test: Area Under ROC 0.8428030303030303
Test: Area Under PR 0.7930118994150303
```

Performance evaluation of the model on training, validation and test data



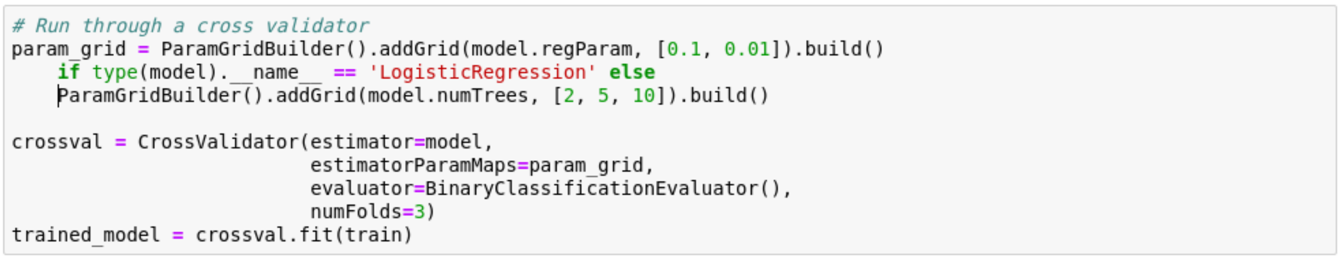
```
get_classifier_metrics(trained_model,train,validation,test)
```

	Train	Validation	Test
Accuracy	0.981013	0.852941	0.800000
Precision	0.967742	0.888889	0.666667
Recall	0.937500	0.666667	0.666667
F-Score	0.952381	0.761905	0.666667

Evaluation Metrics for Classifier

H. REFINEMENTS

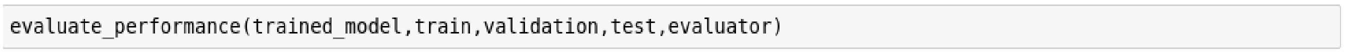
Model training using CrossValidator



```
# Run through a cross validator
param_grid = ParamGridBuilder().addGrid(model.regParam, [0.1, 0.01]).build()
    if type(model).__name__ == 'LogisticRegression' else
    ParamGridBuilder().addGrid(model.numTrees, [2, 5, 10]).build()

crossval = CrossValidator(estimator=model,
                          estimatorParamMaps=param_grid,
                          evaluator=BinaryClassificationEvaluator(),
                          numFolds=3)
trained_model = crossval.fit(train)
```

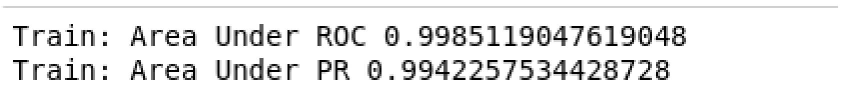
Function to evaluate the performance

I. RESULTS

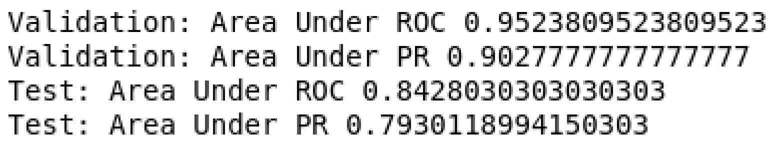


```
evaluate_performance(trained_model,train,validation,test,evaluator)
```

Function call to evaluate the performance

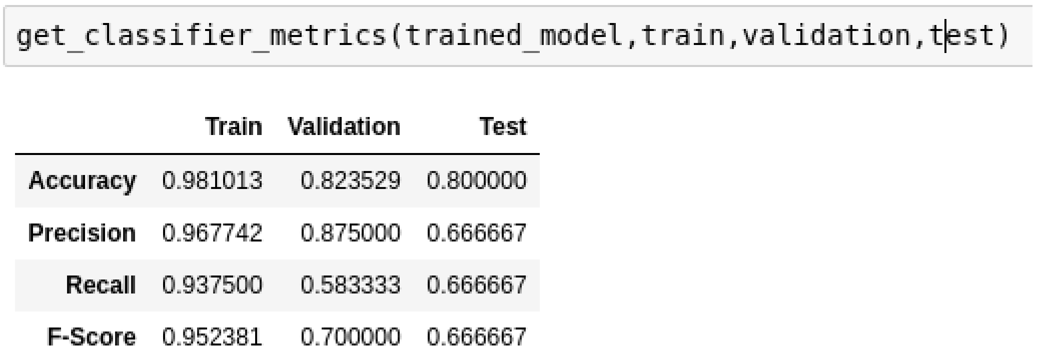


```
Train: Area Under ROC 0.9985119047619048
Train: Area Under PR 0.9942257534428728
```



```
Validation: Area Under ROC 0.9523809523809523
Validation: Area Under PR 0.9027777777777777
Test: Area Under ROC 0.8428030303030303
Test: Area Under PR 0.7930118994150303
```

Performance evaluation of the model on training, validation and test data using CrossValidator



```
get_classifier_metrics(trained_model,train,validation,test)
```

	Train	Validation	Test
Accuracy	0.981013	0.823529	0.800000
Precision	0.967742	0.875000	0.666667
Recall	0.937500	0.583333	0.666667
F-Score	0.952381	0.700000	0.666667

Evaluation Metrics for Classifier

J. JUSTIFICATION:

- I experimented with both LogisticRegression and RandomForestClassifier but RandomForestClassifier performed relatively better so I have shared the code for that.
- The difference in the performance of train and test data shows there is overfitting to some extent but it can be optimized with the help of more hyper parameter tuning.
- It seems we didn't gain much using CrossValidator. However, with more experimentation and hyper parameter tuning it can be perform better.

K. SCOPE OF IMPROVEMENT

- Current performance may be improved by using XGBoost and hyper parameter tuning.
- Adding more features e.g. Gender, days since last login, daily session time etc.

L. REFERENCE(S) / CREDIT(S):

- https://en.wikipedia.org/wiki/Customer_attrition
- https://spark.apache.org/docs/latest/ml-classification-regression.html#random-forest-classifier

- https://spark.apache.org/docs/latest/ml-classification-regression.html#binomial-logistic-regression