Sparkify Churn Prediction

Overview

Many firms fear churn. Some customers will naturally stop using the service, but too many can harm businesses regardless of revenue streams (ad sales, subscriptions, or a mix of both). With this in mind, organizations must be able to predict churn by identifying at-risk customers to avert customer loss.

Machine learning models trained on historical data can detect churn signals and predict them.

In this project, The challenge is forecasting which Sparkify music streaming customers will leave. I'll utilize Spark to build a machine learning model.

Problem Introduction

Churn is a serious worry for many businesses. Some users will naturally cease utilizing the service, but if this number is too large, it can hurt businesses regardless of revenue sources (ad sales, subscriptions or a mix of both). With this in mind, the ability for businesses to predict churn by identifying at-risk clients is critical since it allows them to take steps to prevent the loss of customers, such as targeted offers or discounts.

Machine learning models built on historical data can help us understand churn signals and anticipate it before it happens.

Here the issue is predicting which Sparkify music streaming consumers would abandon the service. To accomplish this, I'll use Spark to create a machine learning model. Because there are only two possible outcomes, 'churn' and 'not churn,' we can use supervised learning to solve this binary classification problem.

Strategy to solve the problem

I will apply the following strategy to solve this problem:

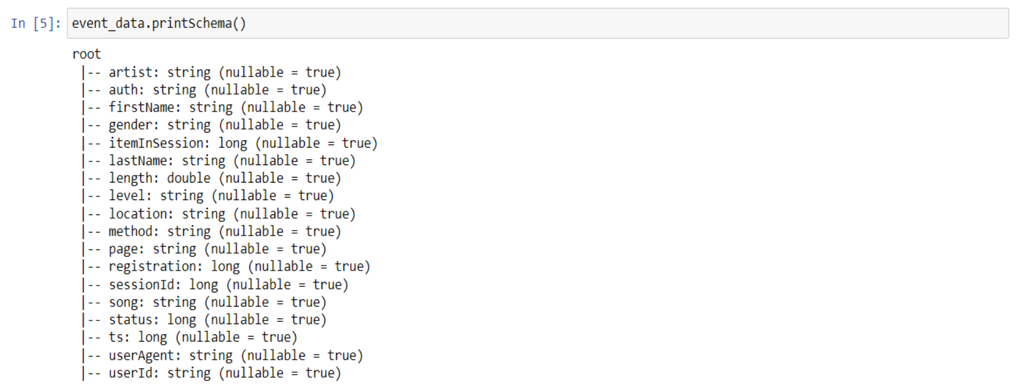
- Problem Understanding
- Data Preparation and Cleaning
- Data Exploration
- Feature Engineering
- Building Model
- Model Tuning

Metrics

The classification distribution is uneven since only 23% of users have the result churn. I picked the F1 Score measure, which is the harmonic mean of precision and recall, to account for this.

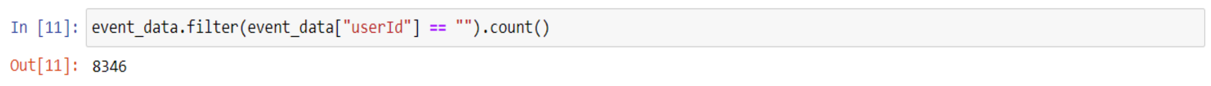
Data Preparation and Cleaning

Our data provided in .json file format with the following schema:

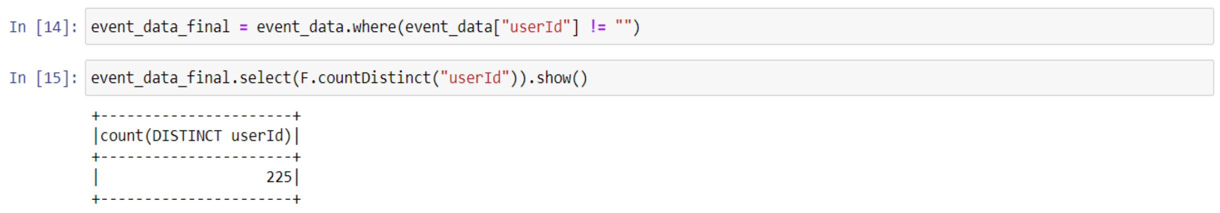


```
In [5]: event_data.printSchema()

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)
```

There is no null values on sessionId/userId columns, hence the only cleaning is required is to remove rows that has empty string in userId column



```
In [11]: event_data.filter(event_data["userId"] == "").count()

Out[11]: 8346
```



```
In [14]: event_data_final = event_data.where(event_data["userId"] != "")

In [15]: event_data_final.select(F.countDistinct("userId")).show()

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

Data Exploration(EDA)

I look at the data with some visuals before using more advanced analysis tools. This method helps me to check whether there are any visual links between the user interactions.

I started with checking values of page column (the available events user can perform through sparkify application)

```
In [18]: event_data_final.groupby("page").count().show(truncate=False)
```

```
+-------------------------+------+
|page                     |count |
+-------------------------+------+
|Cancel                   |52    |
|Submit Downgrade         |63    |
|Thumbs Down              |2546  |
|Home                     |10082 |
|Downgrade                |2055  |
|Roll Advert              |3933  |
|Logout                   |3226  |
|Save Settings            |310   |
|Cancellation Confirmation|52    |
|About                    |495   |
|Settings                 |1514  |
|Add to Playlist          |6526  |
|Add Friend               |4277  |
|NextSong                 |228108|
|Thumbs Up                |12551 |
|Help                     |1454  |
|Upgrade                  |499   |
|Error                    |252   |
|Submit Upgrade           |159   |
+-------------------------+------+
```

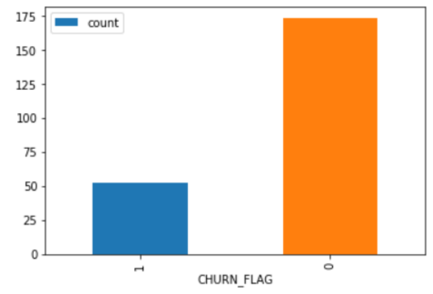
I defined a churn flag as users having “Cancellation Confirmation” transaction.

```
In [20]: churn_user = spark.sql("""
                     SELECT
                         USERID,
                         MAX(CHURN) AS CHURN_FLAG,
                         MAX(DOWNGRADE_SUBMIT) AS DOWNGRADE_SUBMIT
                     FROM
                         (
                         SELECT
                             USERID,
                             CASE WHEN PAGE = 'Cancellation Confirmation' THEN 1 ELSE 0 END AS CHURN,
                             CASE WHEN PAGE = 'Submit Downgrade' THEN 1 ELSE 0 END AS DOWNGRADE_SUBMIT
                         FROM
                             event_data_final
                         )
                     GROUP BY
                         USERID
         """)

         churn_user.createOrReplaceTempView("churn_user")
```

Then checking count of users for each group

```
In [27]: churn_df.select("userId", "CHURN_FLAG").distinct().groupby("CHURN_FLAG").count().show()
```

```
+----------+-----+
|CHURN_FLAG|count|
+----------+-----+
|         1|   52|
|         0|  173|
+----------+-----+
In [28]: churn_df.select("userId", "CHURN_FLAG").distinct().groupby("CHURN_FLAG").count().toPandas().plot.bar(x='CHURN_FLAG', y='count')
         plt.show()
```



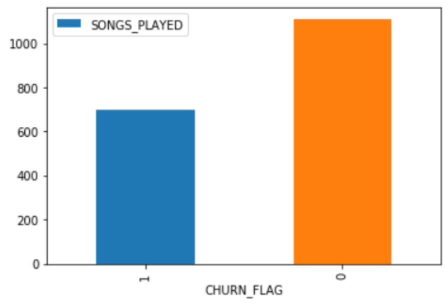
```
52 users out of 225 have churn flag (23%)
```

Of the 225 users present in the dataset, 52 ended up churning i.e. There is 23% of users have churn flag.

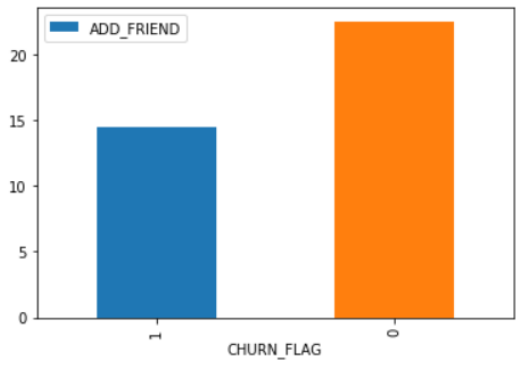
Then I will check some of features that may affect users to churn

1- Songs Played

   After checking count of songs played for each group, I found that non-churn users are more frequently play songs

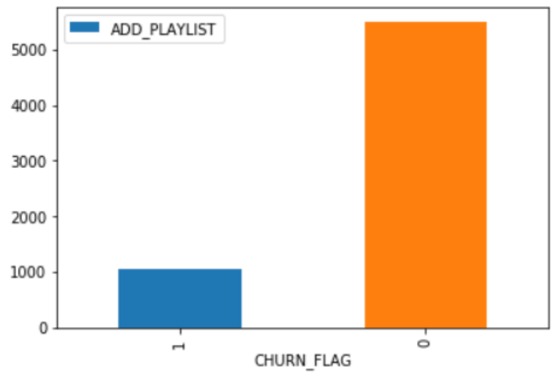


2- Add Friend
   After checking count of "Add Friend" event for each group, I found that non-churn users are more frequently add friends



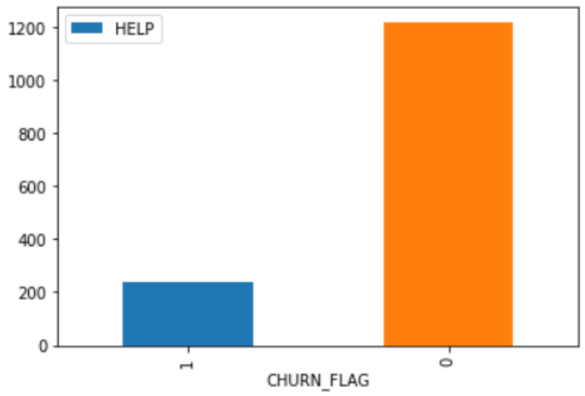
3- Add to Playlist

After checking count of "Add to Playlist" event for each group, I found that non-churn users are more frequently add songs to playlists



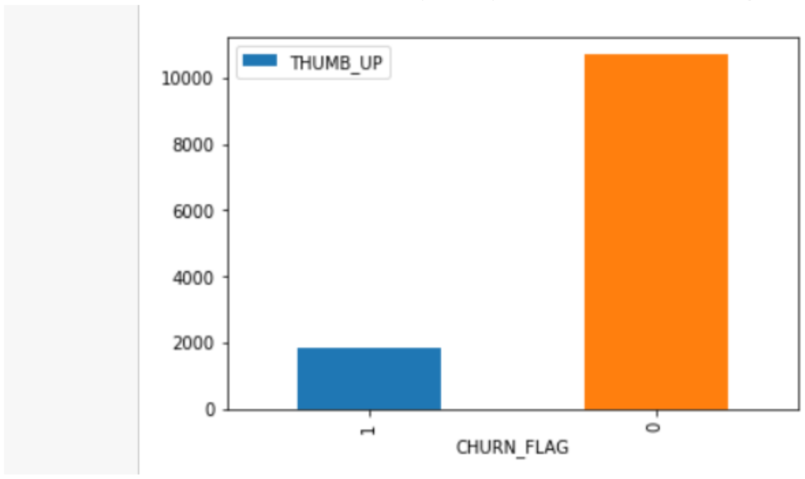
4- Visiting Help Page

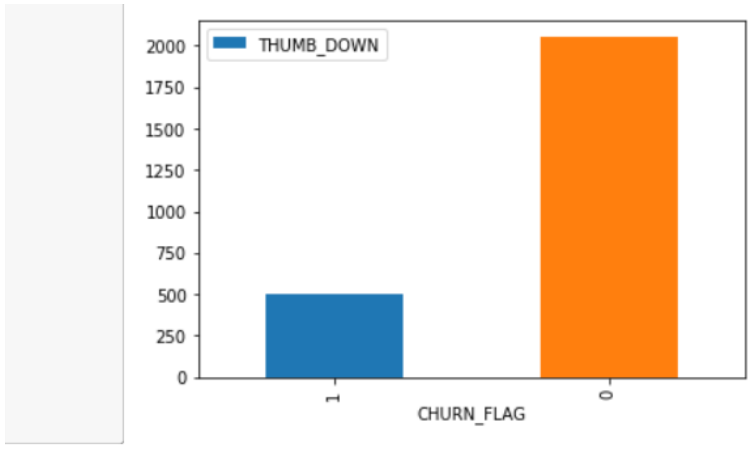
After checking count of "Help" event for each group, I found that non-churn users are more frequently visit Help page



5- Thumbs Up and Down

After checking count of "Thumbs Up" and "Thumbs Down" events for each group, I found that non-churn users are more frequently likes and dislike songs





Modelling

After exploring data and filtering interested features that seems to differentiate the two groups of churn, I will train some supervised machine-learning model and evaluate them.

Not all machine learning algorithms are supported by the Apache Spark framework. This aspect must be factored into my modeling method.

Modeling steps:

1- Split the event data log to train and validation sets (train = 90%, validation = 10%)
2- Build Pipeline: Create label and features vector and scaling the features
3- Train and predict: Not all machine learning algorithms are supported by the Apache Spark framework. This aspect must be factored into my modeling method. I implement three basic algorithms without any parameter tuning as a first step:
    - Random forest
    - Logistic Regression
    - Gradient Boosting
    - Decision Trees
4- Because the sample is highly skewed between those who churned and those who did not, I use the F1 score as the indicator for a successful prediction.

```
In [46]: assembler = VectorAssembler(inputCols=["LENGTH", "SONGS_PLAYED", "TIME_FROM_REGISTERATION", "ADD_FRIEND", "ADD_PLAYLIST", "HELP",
         features_df = assembler.transform(features_df)
```

```
In [47]: scaler = MinMaxScaler(inputCol="FeaturesVect", outputCol="FeaturesStandardScaler")
         scalerModel = scaler.fit(features_df)
         features_df = scalerModel.transform(features_df)
```

```
In [48]: features_df.take(1)

Out[48]: [Row(USERID='100010', CHURN_FLAG=0, LENGTH=66940.89735000003, SONGS_PLAYED=275, TIME_FROM_REGISTERATION=4807612000, ADD_FRIEND=
         4, ADD_PLAYLIST=7, HELP=2, THUMB_UP=17, THUMB_DOWN=5, FeaturesVect=DenseVector([66940.8974, 275.0, 4807612000.0, 4.0, 7.0, 2.0,
         17.0, 5.0]), FeaturesStandardScaler=DenseVector([0.0333, 0.034, 0.2161, 0.028, 0.0292, 0.0435, 0.0389, 0.0667]))]
```

Result

After training each model on the training dataset we tried to evaluate these model using accuracy and f1 score metrics.

```
In [66]: lr =  LogisticRegression(labelCol='label', featuresCol='features')
         rf =  RandomForestClassifier(labelCol='label', featuresCol='features')
         gbt =  GBTClassifier(labelCol='label', featuresCol='features')
         dt = DecisionTreeClassifier(labelCol='label', featuresCol='features')

         lr_trained = lr.fit(rest)
         rf_trained = rf.fit(rest)
         gbt_trained = gbt.fit(rest)
         dt_trained = dt.fit(rest)

         lr_accuracy, lr_f1 = model_score(lr_trained,validation)
         rf_accuracy, rf_f1 = model_score(rf_trained,validation)
         gbt_accuracy, gbt_f1 = model_score(gbt_trained,validation)
         dt_accuracy, dt_f1 = model_score(dt_trained, validation)

         print("logistic regression : accuracy({}), f1score({})".format(lr_accuracy, lr_f1))
         print("random forest : accuracy({}), f1score({})".format(rf_accuracy, rf_f1))
         print("gradient boosting : accuracy({}), f1score({})".format(gbt_accuracy, gbt_f1))
         print("decision tree : accuracy({}), f1score({})".format(dt_accuracy, dt_f1))

         logistic regression : accuracy(0.625), f1score(0.53125)
         random forest : accuracy(0.75), f1score(0.7227272727272727)
         gradient boosting : accuracy(0.6875), f1score(0.6346618357487923)
         decision tree : accuracy(0.6875), f1score(0.6346618357487923)
```

Hyperparameter tuning

I chose Random forest classifier as it has highest f1 score and trying to tune its parameter using cross validator as the following:

```
In [71]: rf_new =  RandomForestClassifier(labelCol='label', featuresCol='features')

         paramGrid = ParamGridBuilder() \
             .addGrid(rf_new.maxDepth,[3, 5, 7]) \
             .build()


         crossval = CrossValidator(estimator=rf_new,
                                   estimatorParamMaps=paramGrid,
                                   evaluator=MulticlassClassificationEvaluator(),
                                   numFolds=3)

         model = crossval.fit(rest)
         model.avgMetrics

Out[71]: [0.7300636533474125, 0.7581814844263598, 0.7488547397867442]
```

I tried multiple value for maxDepth parameter and best performance was for value 5

Conclusion

We deployed Spark machine learning models to estimate churn for Sparkify's users using previous log data.

Reflection:

With 75% and 0.72 f1 score, we developed a model that can detect consumers at danger of churning. This is carried out by To acquire a feel for the data, I first did some data exploration. After then, I started to add features. I trained four distinct types of models once I had a dataset. I used several hyperparameters and cross-validation to fine-tune the Random Forest classifier, which performed the best.

The model evaluation results were acceptable but not ideal, therefore we need to make some changes to get better outcomes.

Improvements:

- Because the dataset in question has an imbalanced problem, resampling approaches could help to improve its performance. Working with more data could also enhance performance and allow for a more thorough search of the hyper-parameter space, making this model even more reliable and helpful. For example, the model could be run every day (or week, depending on computer capability), and targeted messages or discounts may be delivered to those who were indicated as likely to churn.
- To avoid wasting money, A/B testing should be used to determine which action to take. This ensures rigor and statistical significance.
- As the user base and consumer behavior evolve, it is critical that this model be re-trained on a regular basis to avoid making decisions based on obsolete data.
- It would be interesting to duplicate this study on a bigger chunk of the log data if Sparkify had access to a cluster (for example, AWS or IBM Cloud).