Sparkify

May 27, 2021

1 Sparkify Project Workspace

This workspace contains a tiny subset (128MB) of the full dataset available (12GB). Feel free to use this workspace to build your project, or to explore a smaller subset with Spark before deploying your cluster on the cloud. Instructions for setting up your Spark cluster is included in the last lesson of the Extracurricular Spark Course content.

You can follow the steps below to guide your data analysis and model building portion of this project.

In [112]:
```
# import libraries
from pyspark.sql import SparkSession, SQLContext
from pyspark.sql.functions import avg, col, concat, desc, explode, lit, min, max, spli
from pyspark.sql.types import IntegerType
from pyspark.ml.classification import LogisticRegression, RandomForestClassifier, GBTC
from pyspark.ml.feature import StandardScaler,RegexTokenizer, StringIndexer, CountVect
from pyspark.ml import Pipeline
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import BinaryClassificationEvaluator , MulticlassClassifica

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import numpy as np
import pandas as pd
```

In [113]:
```
# create a Spark session
spark = SparkSession.builder.master("local").appName("Capstone_Project").getOrCreate()
```

2 Load and Clean Dataset

In this workspace, the mini-dataset file is `mini_sparkify_event_data.json`. Load and clean the dataset, checking for invalid or missing data - for example, records without userids or sessionids.

In [114]:
```
# load data into spark DataFrame

mydata = spark.read.json("./mini_sparkify_event_data.json")
```

```
          mydata.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)
```

2.1 Clean Data

```
In [115]: #Clean Dataset

          # temp view of the data frame

          mydata.createOrReplaceTempView('data_tbl')
```

```
In [116]: # check if there are nulls in sessionId column

          spark.sql("""
                      SELECT COUNT(userId) as UserId
                      FROM data_tbl
                      WHERE sessionId IS NULL
                    """).show()
```

```
+------+
|UserId|
+------+
|     0|
+------+
```

In [117]:
```
# check if there are empty sessionIds

spark.sql("""
            SELECT COUNT(userId) as UserId
            FROM data_tbl
            WHERE sessionId == ''
          """).show()
```

```
+------+
|UserId|
+------+
|     0|
+------+
```

In [118]:
```
# check if there are nulls in userId column

spark.sql("""
            SELECT COUNT(userId) as UserId
            FROM data_tbl
            WHERE userId IS NULL
          """).show()
```

```
+------+
|UserId|
+------+
|     0|
+------+
```

In [119]:
```
# check if there are empty UserIDs

spark.sql("""
            SELECT COUNT(userId) as UserId
            FROM data_tbl
            WHERE userId == ''
          """).show()
```

```
+------+
|UserId|
+------+
|  8346|
+------+
```

In [120]:
```
# remove the invalid user IDs from the dataset
```

```
mydata = spark.sql("""
                        SELECT *
                        FROM data_tbl
                        WHERE userId != ''
                    """)
```

```
In [121]: # temporary view of the data frame

mydata.createOrReplaceTempView('data_tbl')
```

3 Exploratory Data Analysis

When you're working with the full dataset, perform EDA by loading a small subset of the data and doing basic manipulations within Spark. In this workspace, you are already provided a small subset of data you can explore.

3.0.1 Define Churn

Once you've done some preliminary analysis, create a column `Churn` to use as the label for your model. I suggest using the `Cancellation Confirmation` events to define your churn, which happen for both paid and free users. As a bonus task, you can also look into the `Downgrade` events.

3.0.2 Explore Data

Once you've defined churn, perform some exploratory data analysis to observe the behavior for users who stayed vs users who churned. You can start by exploring aggregates on these two groups of users, observing how much of a specific action they experienced per a certain time unit or number of songs played.

```
In [122]: page = mydata.select("page").dropDuplicates().show()
```

```
+--------------------+
|                page|
+--------------------+
|              Cancel|
|    Submit Downgrade|
|         Thumbs Down|
|                Home|
|           Downgrade|
|         Roll Advert|
|              Logout|
|       Save Settings|
|Cancellation Conf...|
|               About|
|            Settings|
|     Add to Playlist|
|          Add Friend|
|            NextSong|
```

```
|           Thumbs Up|
|                Help|
|             Upgrade|
|               Error|
|      Submit Upgrade|
+--------------------+
```

In [123]:
```
# create churn user list

mydata = spark.sql("""
                        SELECT *,
                               CASE
                                    WHEN page == 'Cancellation Confirmation' THEN 1
                                    ELSE 0 END as Churned
                        FROM data_tbl
                    """)

mydata.createOrReplaceTempView('data_tbl')

Churned = spark.sql("""
                                SELECT DISTINCT userID
                                FROM data_tbl
                                WHERE Churned = 1
                            """).toPandas().values

Churned = [user[0] for user in Churned]
```

In [124]:
```
#show churned and non-churned user in dataset

spark.sql("""
          SELECT
              Churned,
              count(distinct userId)
            FROM
                data_tbl
            GROUP BY
                Churned
            """)
```

Out[124]:
```
DataFrame[Churned: int, count(DISTINCT userId): bigint]
```

In [125]:
```
#create churn table

churn = spark.sql("""
          SELECT
              distinct userId,
              Churned
```

```
                    FROM
                        data_tbl

                    """)
churn.createOrReplaceTempView('churn')
```

In [126]:
```
# show  churn in gender

spark.sql("""
          SELECT distinct
              gender,
              Churned,
              count(distinct userId) as DistinctUsers
            FROM
                data_tbl
            GROUP BY
                gender,Churned
            order by Churned desc
            """)
```

Out[126]: DataFrame[gender: string, Churned: int, DistinctUsers: bigint]

4 Feature Engineering

Once you've familiarized yourself with the data, build out the features you find promising to train your model on. To work with the full dataset, you can follow the following steps. - Write a script to extract the necessary features from the smaller subset of data - Ensure that your script is scalable, using the best practices discussed in Lesson 3 - Try your script on the full data set, debugging your script if necessary

If you are working in the classroom workspace, you can just extract features based on the small subset of data contained here. Be sure to transfer over this work to the larger dataset when you work on your Spark cluster.

In [127]:
```
#gender feature " replace str by int values"


gender = mydata.dropDuplicates(['userId']).sort('userId').select(['userId','gender'])
gender = gender.replace(['F','M'], ['1', '0'], 'gender')
gender = gender.withColumn('gender', gender.gender.cast("int"))

gender.createOrReplaceTempView('gender')
```

In [128]:
```
#number of songs played per user

songs = mydata.where(mydata.song!='null').groupby('userId')
songs= songs.agg(count(mydata.song).alias('Played_Songs')).orderBy('userId')
songs = songs.select(['userId','Played_Songs'])

songs.createOrReplaceTempView('songs')
```

```
In [129]: # number of listened singers per user

          listened_singers_per_user = mydata.dropDuplicates(['userId','artist']).groupby('userId
          listened_singers_per_user = listened_singers_per_user.agg(count(mydata.artist).alias('
          listened_singers_per_user = listened_singers_per_user.select(['userId','Listened_Singe

          listened_singers_per_user.createOrReplaceTempView('listened_singers_per_user')

In [130]: #thumbs_Down

          thumbs_Down = mydata.where(mydata.page=='Thumbs Down').groupby(['userId'])
          thumbs_Down = thumbs_Down.agg(count(col('page')).alias('thumbs_down')).orderBy('userId
          thumbs_Down = thumbs_Down.select(['userId','thumbs_down'])

          thumbs_Down.createOrReplaceTempView('thumbs_Down')

In [131]: #thumbs_Up

          thumbs_Up = mydata.where(mydata.page=='Thumbs Up').groupby(['userId'])
          thumbs_Up = thumbs_Up.agg(count(col('page')).alias('thumbs_Up')).orderBy('userId')
          thumbs_Up = thumbs_Up.select(['userId','thumbs_Up'])

          thumbs_Up.createOrReplaceTempView('thumbs_Up')
```

5 Modeling

Split the full dataset into train, test, and validation sets. Test out several of the machine learning methods you learned. Evaluate the accuracy of the various models, tuning parameters as necessary. Determine your winning model based on test accuracy and report results on the validation set. Since the churned users are a fairly small subset, I suggest using F1 score as the metric to optimize.

```
In [132]: # join features

          Data = churn.dropDuplicates(['userId']).sort('userId').select(['userId','Churned'])
          for selected_features in [ gender, songs, listened_singers_per_user, thumbs_Up, thumbs
              Data = Data.join(selected_features,'userId')

In [133]: # convert data type into float
          for selected_features in Data.columns[1:]:
              Data = Data.withColumn(selected_features,Data[selected_features].cast('float'))

In [134]: Data.dtypes

Out[134]: [('userId', 'string'),
           ('Churned', 'float'),
           ('gender', 'float'),
           ('Played_Songs', 'float'),
```

```
 ('Listened_Singers', 'float'),
 ('thumbs_Up', 'float'),
 ('thumbs_down', 'float')]
```

In [135]:
```
# split our data into train and test sets

train_set, test_set = Data.randomSplit([0.8, 0.2])
```

In [136]:
```
assembler = VectorAssembler(inputCols=Data.columns[2:],outputCol='featuresassemble')
scaler = StandardScaler(inputCol="featuresassemble", outputCol="features")
indexer = StringIndexer(inputCol="Churned", outputCol="label")
stringIndexer = StringIndexer(inputCol="label", outputCol="indexed")
RandomForestClassifier = RandomForestClassifier(numTrees=3, maxDepth=2, labelCol="inde
LogisticRegression =  LogisticRegression(maxIter=100, regParam=0.0, elasticNetParam=0)
```

In [137]:
```
LogisticRegression_pipeline = Pipeline(stages=[assembler, scaler, indexer, LogisticReg

paramGrid_LogisticRegression = ParamGridBuilder().addGrid(LogisticRegression.regParam,

CrossValidator_LogisticRegression = CrossValidator(estimator=LogisticRegression_pipeli
                                    evaluator=BinaryClassificationEvaluator(),numFolds

CrossValidator_LogisticRegression_Model = CrossValidator_LogisticRegression.fit(train_

CrossValidator_LogisticRegression_Model.avgMetrics
```

Out[137]:
```
[0.6658790392910175, 0.7057689613270375, 0.7096003789515586]
```

6 Final Steps

Clean up your code, adding comments and renaming variables to make the code easier to read and maintain. Refer to the Spark Project Overview page and Data Scientist Capstone Project Rubric to make sure you are including all components of the capstone project and meet all expectations. Remember, this includes thorough documentation in a README file in a Github repository, as well as a web app or blog post.

In [ ]: