Sparkify

February 11, 2020

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 [1]: # Import Librariesimport os
        import re

        from matplotlib import pyplot as plt
        from pyspark.sql import SparkSession
        from pyspark.sql.functions import avg, col, concat, desc, explode, lit, min, max, split,
        from pyspark.sql.types import IntegerType, BooleanType, FloatType
        from pyspark.ml.feature import VectorAssembler, StandardScaler
        from pyspark.storagelevel import StorageLevel
        from pyspark.ml.classification import LogisticRegression, GBTClassifier
        from pyspark.ml.evaluation import BinaryClassificationEvaluator
        from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

        import importlib
        %matplotlib inline
```

```
In [2]: # Create a spark session
        spark = SparkSession.builder.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 [3]: # 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 preview of the data
df.head()
```

```
Out[3]: Row(artist='Martha Tilston', auth='Logged In', firstName='Colin', gender='M', itemInSess
```

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 [4]: print((df.count(), len(df.columns)))
```

```
(286500, 18)
```

```
In [5]: df.columns
```

```
Out[5]: ['artist',
         'auth',
         'firstName',
         'gender',
         'itemInSession',
         'lastName',
         'length',
         'level',
         'location',
         'method',
         'page',
         'registration',
         'sessionId',
         'song',
         'status',
         'ts',
         'userAgent',
         'userId']
```

In [6]:
```
# Let's see what all pages have been visited by users
df.select('page').distinct().show(50)
```

```
+--------------------+
|                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|
+--------------------+
```

In [7]:
```
# Select relevant columns
df_clean = df.select('artist','auth','firstName','gender','lastName','length','level','l
```

In [8]:
```
# Combine all auths in one column so that we can search for Cancelled i.e. our churn de
df_churn = df_clean.groupby('userId').agg(collect_list('auth').alias("auths"))
```

In [9]:
```
df_churn.show(1)
```

```
+------+--------------------+
|userId|               auths|
+------+--------------------+
|100010|[Logged In, Logge...|
+------+--------------------+
only showing top 1 row
```

```
In [10]: # Create a custom UDF for creating the churned column
         udf_churned = udf(lambda x: 'Cancelled' in x)

In [11]: # Create the churned column and drop the auths column
         df_churn = df_churn.withColumn("Churned", udf_churned(df_churn.auths))
         df_churn = df_churn.drop('auths')

In [12]: # Join the churn df with the clean df on the basis of user ID
         df_label = df_churn.join(df_clean,'userId')

In [13]: # Show the first record of df_label
         df_label.show(1)
```

```
+------+-------+--------------------+---------+---------+------+---------+---------+-----+------
|userId|Churned|              artist|     auth|firstName|gender| lastName|   length|level|
+------+-------+--------------------+---------+---------+------+---------+---------+-----+------
|100010|  false|Sleeping With Sirens|Logged In| Darianna|     F|Carpenter|202.97098| free|Bridge
+------+-------+--------------------+---------+---------+------+---------+---------+-----+------
only showing top 1 row
```

```
In [14]: # Let's see the distribution of churned column
         df_label.select(["userId","Churned"]).distinct().groupBy("Churned").count().collect()

Out[14]: [Row(Churned='false', count=174), Row(Churned='true', count=52)]

In [15]: # 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_lab
```

```
+------+-------+------+----+---------+------+--------+------+-----+--------+----+-----+---+
|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|
+------+-------+------+----+---------+------+--------+------+-----+--------+----+-----+---+
```

```
In [16]: # Persist the dataframe both in memory and on disk
         df_label.persist(StorageLevel.MEMORY_AND_DISK)

Out[16]: DataFrame[userId: string, Churned: string, artist: string, auth: string, firstName: str
```

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 [17]: # Calculate Thumbs Up: Filter Thumbs Up page visits groups then group that data by user
         # aggregates the counts of the Thumbs Up page per user, give alias name to the column a
         thumbs_up = df_label.where(df_label.page=='Thumbs Up').groupby("userId").agg(count(col(
```

```
In [18]: # Calculate Thumbs Down: Filter Thumbs Down page visits groups then group that data by
         # aggregates the counts of the Thumbs Down page per user, give alias name to the column
         thumbs_down = df_label.where(df_label.page=='Thumbs Down').groupby("userId").agg(count(
```

```
In [19]: # Join both thumbs up and thumbs join using userID
         thumbs_up_and_down = thumbs_up.join(thumbs_down,'userId')
```

```
In [20]: # Calculate Songs Played by a user
         songs_played = df_label.where(col('song')!='null').groupby("userId").agg(count(col('son
```

```
In [21]: # Join songs played and all thumbs data to the df features
         df_features = df_churn.join(songs_played,'userId')
         df_features = df_features.join(thumbs_up_and_down,'userId')
```

```
In [22]: # Calculate number of days user has been using the service
         days = df_label.groupby('userId').agg(((max(col('ts')) - min(col('ts')))/86400000).alia
         df_features = df_features.join(days, "userId")
```

```
In [23]: # Check null count in key features. After running this cell it's clear that it's all ze
         df_features.select([count(when(isnull(column), column)).alias(column) for column in ["u
```

```
+------+-----------+--------+----------+----+
|userId|SongsPlayed|ThumbsUp|ThumbsDown|Days|
+------+-----------+--------+----------+----+
|     0|          0|       0|         0|   0|
+------+-----------+--------+----------+----+
```

```
In [24]: # This is to avoid the problem which can occur in full data set i.e.
         # a Spark UDF will return a column of NULLs if the input data type doesnt match the out
         udf_thumbs_up_per_song = udf(lambda thumbsUp, songsPlayed: float(thumbsUp)/float(songsP
         udf_thumbs_down_per_song = udf(lambda thumbsDown, songsPlayed: float(thumbsDown)/float(
         udf_songs_played_per_hour = udf(lambda songsPlayed, numberOfDays: float(songsPlayed)/fl
```

```
In [25]: # Add the Thumbs UpPerSong using UDF
         df_features = df_features.withColumn("ThumbsUpPerSong", udf_thumbs_up_per_song(df_featu
```

```
In [26]: # Add the Thumbs DownPerSong using UDF
         df_features = df_features.withColumn("ThumbsDownPerSong", udf_thumbs_down_per_song(df_f
```

```
In [27]: # Add the SongsPerHour feature using UDF
         df_features = df_features.withColumn("SongsPerHour", udf_songs_played_per_hour(df_featu

In [28]: # Let's see what's the average of SongsPerHour for both churned as well as non churned
         # It seems that user who got churned have played more songs compared to users who didn'
         # Do they get bored?? Or they didn't like the service after experimenting a lot?
         df_features.select("SongsPerHour", "Churned").groupby("Churned").agg(avg(col('SongsPerH
```

```
+-------+------------------+
|Churned| avg(SongsPerHour)|
+-------+------------------+
|  false|1.3289544926175187|
|   true| 2.415751484163264|
+-------+------------------+
```

```
In [29]: df_features.select([count(when(isnull(c), c)).alias(c) for c in ["SongsPlayed", "Thumbs
```

```
+-----------+---------------+-----------------+----+------------+
|SongsPlayed|ThumbsUpPerSong|ThumbsDownPerSong|Days|SongsPerHour|
+-----------+---------------+-----------------+----+------------+
|          0|              0|                0|   0|           0|
+-----------+---------------+-----------------+----+------------+
```

```
In [30]: assembler = VectorAssembler(inputCols=["SongsPlayed", "ThumbsUpPerSong", "ThumbsDownPer
         df_features = assembler.transform(df_features)

In [31]: scaler = StandardScaler(inputCol="FeatureVector", outputCol="ScaledFeatures", withStd=T
         scaler_tranformer = scaler.fit(df_features)
         df_features = scaler_tranformer.transform(df_features)

In [32]: df_features.show()
```

```
+------+-------+-----------+--------+----------+------------------+---------------+------------
|userId|Churned|SongsPlayed|ThumbsUp|ThumbsDown|              Days|ThumbsUpPerSong|ThumbsDownPer
+------+-------+-----------+--------+----------+------------------+---------------+------------
|100010|  false|        275|      17|         5| 44.21780092592593|    0.061818182|      0.01818
|200002|  false|        387|      21|         6|45.496805555555554|    0.054263566|      0.01550
|   124|  false|       4079|     171|        41|59.996944444444445|     0.04192204|      0.01005
|    51|   true|       2111|     100|        21|15.779398148148148|    0.047370914|      0.00994
|     7|  false|        150|       7|         1|50.784050925925925|    0.046666667|      0.00666
|    15|  false|       1914|      81|        14| 54.77318287037037|     0.04231975|     0.007314
|    54|   true|       2841|     163|        29| 42.79719907407407|    0.057374164|      0.01020
|   155|  false|        820|      58|         3| 25.82783564814815|     0.07073171|     0.003658
|100014|   true|        257|      17|         3|41.244363425925926|     0.06614786|     0.011673
|   132|  false|       1928|      96|        17| 50.49740740740741|    0.049792532|      0.00881
```

```
|   101|   true|       1797|      86|        16|15.861481481481482|     0.04785754|       0.00890
|    11|  false|        647|      40|         9|53.241585648148146|    0.061823804|       0.01391
|   138|  false|       2070|      95|        24| 56.07674768518518|     0.04589372|       0.01159
|300017|  false|       3632|     303|        28| 59.11390046296296|     0.08342511|      0.007709
|100021|   true|        230|      11|         5|45.457256944444445|    0.047826085|        0.0217
|    29|   true|       3028|     154|        22| 43.32092592592593|    0.050858654|      0.007265
|    69|  false|       1125|      72|         9| 50.98648148148148|          0.064|             0
|   112|  false|        215|       9|         3| 56.87869212962963|    0.041860465|       0.01395
|    42|  false|       3573|     166|        25| 60.08825231481482|     0.04645956|      0.006996
|    73|   true|        377|      14|         7| 21.52954861111111|    0.037135277|       0.01856
+------+-------+-----------+--------+----------+------------------+---------------+-------------
only showing top 20 rows
```

In [33]:
```python
# UDF for convert label or targeting variable to integer
convertToInt = udf(lambda x: 1 if x=="true" else 0, IntegerType())
```

In [34]:
```python
# Now copy the churned column to label column post converting to integer
df_features = df_features.withColumn('label', convertToInt(df_features.Churned))
```

In [35]:
```python
df_features.show()
```

```
+------+-------+-----------+--------+----------+------------------+---------------+-------------
|userId|Churned|SongsPlayed|ThumbsUp|ThumbsDown|              Days|ThumbsUpPerSong|ThumbsDownPer
+------+-------+-----------+--------+----------+------------------+---------------+-------------
|100010|  false|        275|      17|         5| 44.21780092592593|    0.061818182|       0.01818
|200002|  false|        387|      21|         6|45.496805555555554|    0.054263566|       0.01550
|   124|  false|       4079|     171|        41|59.996944444444445|     0.04192204|       0.01005
|    51|   true|       2111|     100|        21|15.779398148148148|    0.047370914|       0.00994
|     7|  false|        150|       7|         1|50.784050925925925|    0.046666667|       0.00666
|    15|  false|       1914|      81|        14| 54.77318287037037|     0.04231975|      0.007314
|    54|   true|       2841|     163|        29| 42.79719907407407|    0.057374164|       0.01020
|   155|  false|        820|      58|         3| 25.82783564814815|     0.07073171|      0.003658
|100014|   true|        257|      17|         3|41.244363425925926|     0.06614786|      0.011673
|   132|  false|       1928|      96|        17| 50.49740740740741|    0.049792532|       0.00881
|   101|   true|       1797|      86|        16|15.861481481481482|     0.04785754|       0.00890
|    11|  false|        647|      40|         9|53.241585648148146|    0.061823804|       0.01391
|   138|  false|       2070|      95|        24| 56.07674768518518|     0.04589372|       0.01159
|300017|  false|       3632|     303|        28| 59.11390046296296|     0.08342511|      0.007709
|100021|   true|        230|      11|         5|45.457256944444445|    0.047826085|        0.0217
|    29|   true|       3028|     154|        22| 43.32092592592593|    0.050858654|      0.007265
|    69|  false|       1125|      72|         9| 50.98648148148148|          0.064|             0
|   112|  false|        215|       9|         3| 56.87869212962963|    0.041860465|       0.01395
|    42|  false|       3573|     166|        25| 60.08825231481482|     0.04645956|      0.006996
|    73|   true|        377|      14|         7| 21.52954861111111|    0.037135277|       0.01856
+------+-------+-----------+--------+----------+------------------+---------------+-------------
only showing top 20 rows
```

```
In [36]: # 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=
```

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 [37]: # This is for internal so that we can play with multiple models
         relevant_module_class = 'pyspark.ml.classification'
         relevant_model_class = 'RandomForestClassifier' # LogisticRegression
         model_params = {'featuresCol': 'FeatureVector', 'labelCol': 'label', 'maxIter': 10} if
         relevant_module = importlib.import_module(relevant_module_class)
         relevant_model = getattr(relevant_module,relevant_model_class)
         model = relevant_model(**model_params)
```

```
In [38]: 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(pre
             print('Train: Area Under PR', evaluator.setMetricName("areaUnderPR").evaluate(predi

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

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

```
In [39]: from pyspark.ml.classification import RandomForestClassifier
         evaluator = BinaryClassificationEvaluator()
         model = RandomForestClassifier(featuresCol = 'FeatureVector', labelCol = 'label', numTr
```

```
In [40]: trained_model = model.fit(train)
```

```
In [41]: evaluate_performance(trained_model,train,validation,test,evaluator)
```

```
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
```

In [42]:
```
# Run through a cross validator
param_grid = ParamGridBuilder().addGrid(model.regParam, [0.1, 0.01]).build() if type(mo

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

In [43]:
```
evaluate_performance(trained_model,train,validation,test,evaluator)
```

```
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
```

In [44]:
```
trained_model.coefficients if relevant_model_class == 'LogisticRegression' else trained
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-44-577410108966> in <module>()
----> 1 trained_model.coefficients if relevant_model_class == 'LogisticRegression' else trai


AttributeError: 'CrossValidatorModel' object has no attribute 'featureImportances'
```

In [ ]:
```
# EXPERIMENTED BUT DIDN'T USE
#training_summary = trained_model.summary

# Get the receiver-operating characteristic as a dataframe and areaUnderROC.
# training_summary.roc.show()
#print("areaUnderROC: " + str(training_summary.areaUnderROC))

# objectiveHistory = training_summary.objectiveHistory
```

```
# print("objectiveHistory:")
# for objective in objectiveHistory:
#     print(objective)

#f_measure = training_summary.fMeasureByThreshold
#max_f_measure = f_measure.groupBy().max('F-Measure').select('max(F-Measure)').head()
#best_threshold = f_measure.where(f_measure['F-Measure'] == max_f_measure['max(F-Measure
#    .select('threshold').head()['threshold']

#f_measure.show()
#print(best_threshold)
#print(max_f_measure)

# model.setThreshold(best_threshold)

#pr = training_summary.pr
#pr.show()

#predictions.show()
#print(predictions.filter(predictions.label == predictions.prediction).count())
#print(predictions.count())
```

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.

7 Based on the coefficients, the features that contribute the most are:

Average number of thumbsdown per song played Number of songs played