PROJECT REPORT

CSP 554 Big Data Technologies

December 7th, 2022

Customer Churn Prediction on Sparkify Dataset

Team Members

Jasleen Bhatia - A20495420

Kranti Mahadev - A20473187

Nandish Bhagat - A20490179

Taranpreet Bhatia - A20460031

INDEX

I. ABSTRACT

Sparkify is a music streaming dataset. Customers may switch from a free version to a premium membership or paid subscriptions, or vice versa as they interact with the music service. As we know, customer churn occurs when a customer chooses to terminate or degrade their membership to a certain plan. The aim of this project is to anticipate the customer churn on the real-world resemblance music streaming dataset.

II. OVERVIEW

1.1 Objective

The project mainly focuses on predicting customer churn (detect if the specific customer will cancel the service) on the Sparkify Dataset (mini dataset – 128 Mb & full dataset – 12 Gb). The dataset is composed of numerous user events in the audio streaming service provider like Spotify and Pandora. At first, we will analyze the mini dataset using Spark Dataframe, Spark SQL & Spark ML API locally using Pyspark and then deploy the spark cluster on AWS to run the model on mini dataset.If time permits, we will do the same process on full dataset. Moreover, we will use Tableau to visualize the insights from the dataset.

1.2 What we seek to Address

To achieve the aim of this project, we will focus and investigate customers churn by gathering insights from the dataset -

- Distribution of each feature
- Distribution of customers by gender
- Churn by location/states
- Visualization of free vs paid customers
- Predicting customer churn.

III. LITERATURE REVIEW

Apache Spark has become the most popular tool for analyzing large datasets. We have demonstrated the use of Spark for scalable data manipulation and machine learning. We have used the user log data from a fictitious music streaming company, Sparkify, to predict which customers are at risk to churn.

We analyzed a mini dataset (128 MB) and built classification models using Spark Dataframe, Spark SQL, and Spark ML APIs in local mode through the python interface API, PySpark. Then we deployed a Spark cluster on AWS to run the models. For models, we tried Logistic Regression, Random Forest and Gradient Boosted Trees.
Apache Spark is an open source distributed processing system used for big data workloads. It utilizes in-memory caching, and optimized query execution for fast analytic queries against data of any size. It provides development APIs in Java, Scala, Python and R, and supports code reuse across multiple workloads—batch processing, interactive queries, real-time analytics, machine learning, and graph processing.

The reason we used Spark, is because it is capable of handling several petabytes of data at a time. It has an extensive set of developer libraries and APIs and supports languages such as Java, Python, R, and Scala; its flexibility makes it well-suited for a range of use cases. Typical use cases include:
Stream Processing: The data arrives in a steady stream, often from multiple sources simultaneously. Streams of data can be processed in real time using Spark.
Machine Learning: Spark's ability to store data in memory and rapidly run repeated queries makes it a good choice for training machine learning algorithms. Running broadly similar queries again and again, at scale, significantly reduces the time required to go through a set of possible solutions in order to find the most efficient algorithms.
Interactive Analytics: The interactive query process requires systems like Spark that are able to respond to questions asked for data exploration, and adapt quickly.
Data Integration: Spark is widely used to reduce the cost and time required to extract, transform and load data from different systems, clean and standardize it.

For visualization of results, we have used Tableau. All the graphs and visual representations of data are done easily with the help of Tableau.

In the future, we can use H2O AutoML, when using a very large dataset. It is the process of automating algorithm selection, feature generation, hyperparameter tuning, iterative modeling, and model assessment. AutoML tools make it easy to train and evaluate machine learning models. Automating the repetitive data science tasks allows people to focus on the data and the business problems they are trying to solve. H2O AutoML selects the best model automatically and there is no need to run three different models as we did.

IV. DATA PROCESSING

2.1 Dataset Description

- Mini Dataset – 128 MB with 286K records with 225 unique customers
- Large Dataset – 12 GB with 26M records with 22k unique customers

We are now utilizing the small dataset to address this problem since our job is to determine if a given client will terminate the services or not and to understand how this will influence.

2.2 Issues in data and changes made

Sparkify Dataset:

- There are a lot of missing values and column label mismatch in the dataset.
- Data containing the “Userid”, whose “FirstName” has a lot of missing values.
- There are a lot of missing values for artist, pages, length, songs etc.
- Date Time formats modification.
- Store a copy of the original and transformed datasets in multiple cloud platforms and systems for backup in case of failure of an individual system.
- **Changes made:**
    - Renamed columns among all files to common labels
    - Artist value used to populate missing FirstName and LastName values.
    - Formatted date time and other feature values in all files to have a single standard format.
    - Merged a few features to one which was used for analysis.
    - Season data has been added to each row based on the month of the trip.

2.3 Tools

- Programming Language - Python
- Applications/Framework - Jupyter Notebook and Apache Spark
- Libraries -
- Development - GitHub
- Visualization Tool - Tableau

V. PROJECT OUTLINE

- Download data to local directory
- Upload data and python script to an S3 bucket
- Create a cluster that uses Spark and JupyterHub applications
- Process the data
- Analyze the data
- Visualization reports
- Visualize the data on Tableau

VI. METHOD

The initial step was to download and gather the information from the official website. After that, the data files were moved from the local computer to an S3 bucket on AWS. To get the S3 bucket operating, we used the identical procedures as earlier assignments to create a key-pair and an EMR cluster. Once the data was placed onto S3, it was simple for us to undertake analysis and produce engaging visualizations.

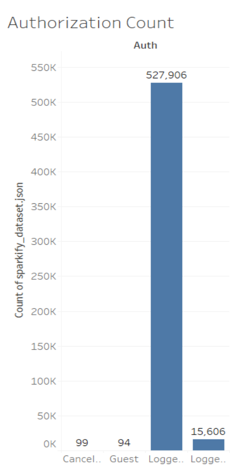
Once the data was in the S3 Bucket, we used JupyterHub to set up a PySpark environment so that we could analyze the data. In PySpark, we were able to create efficient code that produced some worthwhile data insights. Before we could utilize the data for any studies, it had to be prepped and cleaned which we already did.

VII. EXPLORATORY DATA ANALYTICS

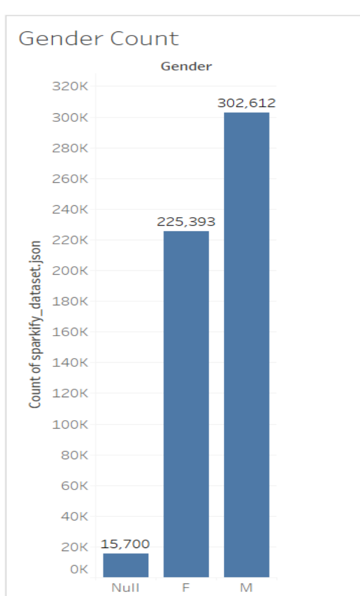
Our prediction model's goal is to identify which clients are likely to leave and which are not. Thus, the issue is fundamentally one of binary categorization. Churned vs. Engaged are the classes.If engaged consumers are classified as churning ones, the company could take activities that mislead the customer and possibly cause them to churn the service. The proper classification of clients who are churning is also crucial. Therefore, both client classes should be accurately classified by our classifier.

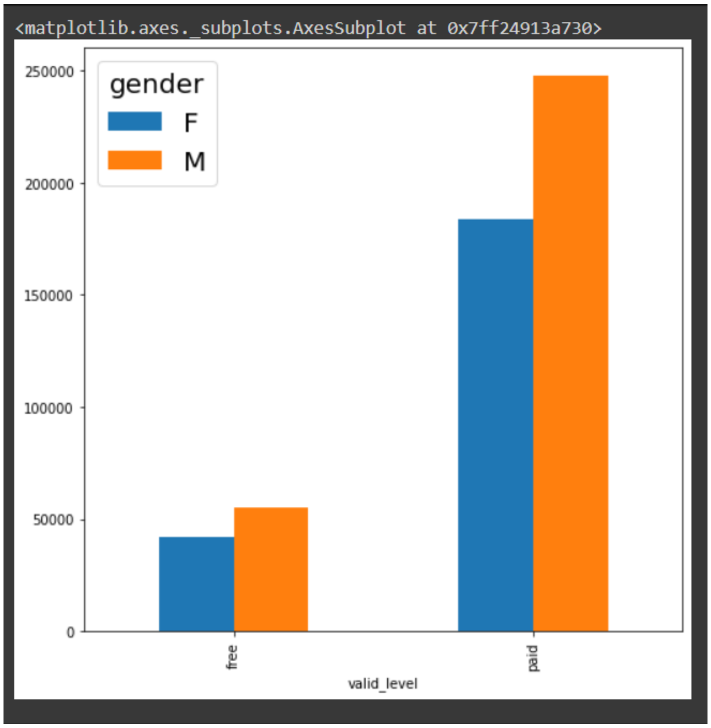
Visualizations

- **Distribution by Authorization Count**

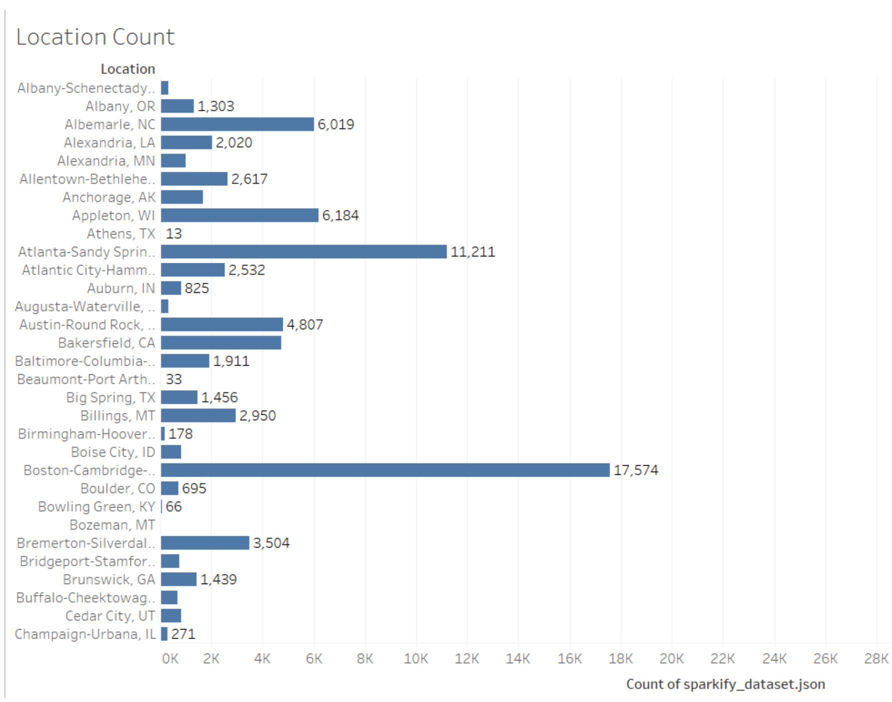


- **Distribution of customers by gender**

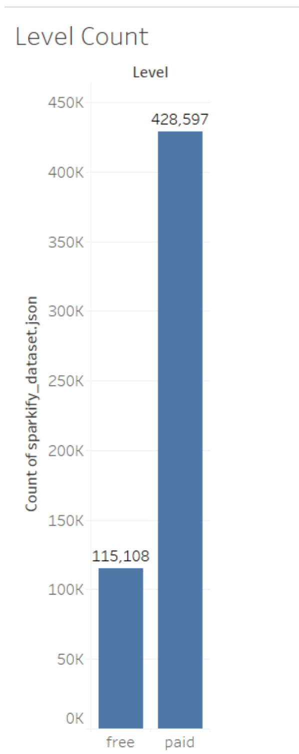




- **Churn by location/states**



- **Visualization of free vs paid customers**



VIII. DATA CLEANING

Data cleaning is a critically important step in any machine learning project. Data cleaning refers to identifying and correcting errors in the dataset that may negatively impact a predictive model. There are various steps involved in data cleaning, which include, removal of unwanted observations, fixing structural errors, managing unwanted outliers and handling missing data.

- Creating a function to clean the data by removing null values and processing time, date etc. columns

```
## Function to Clean the Data
def datacleaning(df):
  for field in df.schema.fields:
        if field.dataType==StringType():
            df = df.withColumn(field.name,regexp_replace(field.name,'[^a-zA-Z0-9\,\-]',''))

  df1 = df.withColumn('interaction_time', from_unixtime(col('ts').cast(LongType())/1000).cast(TimestampType()))
  df2 = df1.withColumn('month', month(col('interaction_time')))
  #df2
  df3 = df2.withColumn('date', from_unixtime(col('ts')/1000).cast(DateType()))
  df4 = df3.withColumn('userId', col('userId').cast(LongType()))
  df5 = df4.filter(col('userId').isNotNull())
  #print(df5)
  df6 = df5.filter(col('auth')!='LoggedOut')
  df7 = df6.withColumn('location', split(col('location'),',').getItem(1))
  return df7
```

- Function to add the label column for binary classification
- Function to get the number of days from the registration of the user
- Function to get the subscription level of the user
- Function to calculate average item length for each user

```
[19] ## Function to label the data
     def labelling(df):
         labelling = df.withColumn('label',when((col('page').isin(['Cancellation Confirmation','Cancel'])) | (col('auth')=='Cancelled'),1 ).otherwise(0)).groupby('userId')
         df = df.join(labelling, on='userId')
         #df
         return df

[20] ## Function to check the Registred days
     def registered_days(df):
         lastused =  df.groupBy('userId').agg(max('ts').alias('last_interaction'))
         df = lastused.join(df, on='userId').withColumn('registered_days', ((col('last_interaction')-col('registration'))/86400000).cast(IntegerType()))

         return df

     ## Function to check the Subscription Level
     def latest_level(df):
         lastlevel = df.orderBy('ts', ascending=False).groupBy('userId').agg(first('level').alias('valid_level'))
         df = df.drop('level')
         #df
         df = df.join(lastlevel, on='userId')
         return df

[22] def avglen(df):
         averagelength = df.groupBy('userId').avg('length').withColumnRenamed('avg(length)', 'length')
         df = df.drop('length')
         #print(df)
         df = df.join(averagelength, on='userId')
         return df
```

- Function to build a pipeline with indexer and assembler

```
[23] ## Building a Pipline
     def pipeline(num_cols):
         gender =StringIndexer(inputCol='gender', outputCol='gender_index')
         location =StringIndexer(inputCol='location', outputCol='location_index')
         assembler= VectorAssembler(inputCols=num_cols, outputCol='features')
         pipeline= Pipeline(stages=[gender, location, assembler])

         return pipeline
```

- Function to get the monthly and daily averages for each session of each user
- Function for calculate monthly and daily averages for each item of each user

```
[27] def sessionduration(df):
         daily = df.groupby('userId','date','sessionId').agg(max('ts').alias('session_end'), min('ts').alias('session_start')).withColumn('session_duration_sec', (col('session_end')-col('session_star

         monthly = df.groupby('userId','month','sessionId').agg(max('ts').alias('session_end'), min('ts').alias('session_start')).withColumn('session_duration_sec', (col('session_end')-col('session_

         return daily.join(monthly, on='userId')
```

```
def item_aggregates(df):
    daily2 = df.groupby('userId','date').agg(max('itemInSession')).groupBy('userId').avg('max(itemInSession)').withColumnRenamed('avg(max(itemInSession))', 'avg_daily_items')
    #daily2
    monthly3 = df.groupby('userId','month').agg(max('itemInSession')).groupBy('userId').avg('max(itemInSession)').withColumnRenamed('avg(max(itemInSession))', 'avg_monthly_items')

    return daily2.join(monthly3, on='userId')
```

- Function to calculate the daily and monthly averages for each user for each page except the ones that include “Cancel”

```
def pageevent_dailyormonthly(df):

    # Function to calculate the daily averages for each user except the ones that include cancel
    pg_daily_df,exp_dict = daily_pg(df)
    # Function to calculate the monthly averages for each user except the ones that include cancel
    pg_monthly_df = monthly_pg(df,exp_dict)

    return pg_daily_df.join(pg_monthly_df, on='userId')
```

```
def daily_pg(df):
    listOfDistinctPages =[row.page for row in df.select('page').distinct().collect()]
    listOfDistinctPages.remove('Cancel')
    listOfDistinctPages.remove('CancellationConfirmation')
    daily_page_event_df = df.groupby('userId','date').pivot('page').count()
    exp_dict={}
    for page in listOfDistinctPages:
        exp_dict.update({page:'mean'})
    daily_page_event_df = daily_page_event_df.join(daily_page_event_df.groupBy('userId').agg(exp_dict).fillna(0), on='userId')

    for page in listOfDistinctPages:
        daily_page_event_df =daily_page_event_df.drop(page)
        daily_page_event_df= daily_page_event_df.withColumnRenamed('avg({})'.format(page), 'avg_daily_{}'.format(page))

    pg_daily_df = daily_page_event_df.drop('Cancel','CancellationConfirmation','date').drop_duplicates()
    return pg_daily_df,exp_dict
```

```
[31] def monthly_pg(df,exp_dict):
         listOfDistinctPages =[row.page for row in df.select('page').distinct().collect()]
         listOfDistinctPages.remove('Cancel')
         listOfDistinctPages.remove('CancellationConfirmation')
         # exp_dict={}
         monthly_page_event_df = df.groupby('userId','month').pivot('page').count()

         monthly_page_event_df = monthly_page_event_df.join(monthly_page_event_df.groupBy('userId').agg(exp_dict).fillna(0), on='userId')
         for page in listOfDistinctPages:
             monthly_page_event_df = monthly_page_event_df.drop(page)
             monthly_page_event_df = monthly_page_event_df.withColumnRenamed('avg({})'.format(page), 'avg_monthly_{}'.format(page))

         pg_monthly_df = monthly_page_event_df.drop('Cancel','CancellationConfirmation','month').drop_duplicates()
         return pg_monthly_df
```

- Function to join all aggregate data frames & main data frame for original columns

```python
def mergefeature(df, df_sess_duration, df_item, df_page):

    alljoined =df_sess_duration.join(df_item, on='userId').join(df_page, on='userId')
    #drop the following features
    df = df.drop('auth', 'level','length','userAgent','month','date','interaction_time','registration', 'ts','song','page','itemInSession','sessionId','artist','firstName','lastName','method'
    finaljoined = alljoined.join(df, on='userId')

    finaljoined2 = finaljoined.drop_duplicates()
    features = finaljoined2.drop('userId')

    return features
```

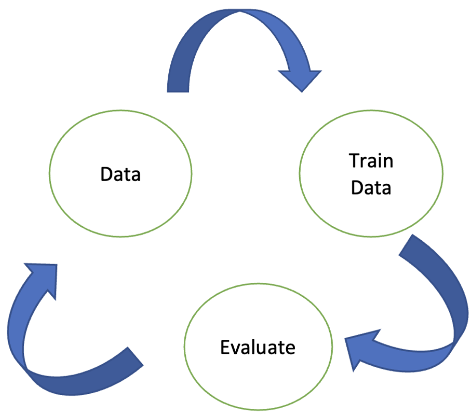
- Final dataframe after data cleaning and processing

```python
df.show()
```

```
+--------------------------+----------------------------+------------------+-----------------+--------------------+------------------+-------------------+------------------+---------------+-
|avg_daily_session_duration|avg_monthly_session_duration|   avg_daily_items|avg_monthly_items|avg_daily_RollAdvert|avg_daily_Settings|avg_daily_Downgrade|avg_daily_NextSong|avg_daily_Error|
+--------------------------+----------------------------+------------------+-----------------+--------------------+------------------+-------------------+------------------+---------------+-
|                    6254.0|                      6254.0|              24.0|             24.0|                 1.0|               0.0|                0.0|              24.0|            0.0|
|                   13246.5|                     16262.0|             70.75|            124.5|   6.333333333333333|               0.0|                0.0|             54.75|            0.0|
|         13753.21052631579|            14771.694444444445| 76.94736842105263|            278.0|                5.75|               1.5| 1.4285714285714286| 62.611111111111114|            1.0|
|          6101.766666666666|            7921.972222222222| 37.46666666666667|            108.0|                 3.5|               1.0|                0.0|              30.8|            0.0|
|         21386.954545454544|            25398.67948717949| 138.4090909090909|            379.5|   2.142857142857143|               2.0|                1.6|  99.5909090909091|            1.0|1.
|                   44981.5|                     90144.0|             404.5|            435.0|                 0.0|               1.0|                1.0|             180.0|            1.0|
|         15128.689655172413|            16899.851648351647| 98.89655172413794|            357.0|                 3.0|1.4285714285714286|               2.25| 60.464285714285715|            1.0|
|         3833.6666666666665|                     3507.75|19.666666666666668|             25.5|                 1.0|               0.0|                0.0|              17.0|            0.0|
|                   14242.0|                     14242.0|              76.0|             76.0|                 5.0|               0.0|                0.0|              62.0|            0.0|
|          4769.357142857143|            8450.988888888889|33.714285714285715|            154.5|   4.833333333333333|               1.0|                1.0| 31.076923076923077|            0.0|
|         12856.333333333334|                     19435.5| 67.66666666666667|            124.0|                 0.0|               1.0|                0.0|              43.0|            0.0|
|         13401.458333333332|            18736.005555555555|              99.1|            312.0|                 1.5|1.1666666666666667|              1.375|              73.2|            1.0|
|         11771.833333333334|            13089.076923076924| 86.58333333333333|            366.0|   3.888888888888889|               2.5| 2.6666666666666665| 57.666666666666664|            0.0|
|         10934.466666666665|            12644.20652173913| 74.74285714285715|            211.5|  3.8421052631578947|1.0833333333333333|              1.125|  54.48571428571429|           1.25|
|                    4986.0|                      5380.0| 29.22222222222222|             92.0|                 1.4|               1.0|                2.0| 22.333333333333332|            0.0|
|         17807.555555555555|            22176.303571428572|118.38888888888889|            276.5|                 1.6|1.1666666666666667|  2.142857142857143|  76.05555555555556|            1.0|1.
|                 24261.875|            39529.066666666666|155.58333333333334|            426.5|                 2.5|1.8571428571428572|                2.0|            103.25|            2.0|
|         12015.083333333332|            10858.57142857143|              95.0|            273.0|   4.333333333333333|               1.5|                2.0|            78.625|            1.0|
|                   11778.9|                   14780.625|              84.0|            136.0|                 3.0|               1.0|                2.0|              48.4|            0.0|
|          20681.85714285714|            24167.333333333336|105.28571428571429|            258.5|  1.3333333333333333|               2.0| 1.3333333333333333|  81.57142857142857|            2.0|
+--------------------------+----------------------------+------------------+-----------------+--------------------+------------------+-------------------+------------------+---------------+-
only showing top 20 rows
```

IX. DATA MODELLING

Data Modeling in software engineering is the process of simplifying the diagram or data model of a software system by applying certain formal techniques. Here, we used Gradient boosting, a technique used in creating models for prediction. The technique is mostly used in regression and classification procedures. Prediction models are often presented as decision trees for choosing the best prediction. Gradient boosting presents model building in stages, just like other boosting methods, while allowing the generalization and optimization of differentiable loss functions.



- Function to train the model on training data on 3 different model namely Logistic Regression, Random Forest and Gradient Boosting and predicting on test dataset

```
from pyspark.ml.classification import LogisticRegression, RandomForestClassifier, GBTClassifier
# Main Function
# Function to Predict
# We are using 3 models i.e. Logistic Regression , Random Forest and Gradient Boosting
def predictionfunction(train, test, model):
    #logistic Regression Model
    if model == 'Logistic-Regression':
        ml = LogisticRegression()
    #Random Forest Model
    elif model == 'Random-Forest':
        ml = RandomForestClassifier()
    # Gradient Boosting Model
    elif model == 'Gradient-Boost':
        ml = GBTClassifier()
    else:
        return "Model not valid"

    clf = ml.fit(train)
    results = clf.transform(test)
    modeleval(results)
    return clf, results
```

- Function to evaluate the model performance in terms of F1 score

```
[26] ## Function to evaluate the Model
     from sklearn import metrics
     def modeleval(results):
         f1_score= MulticlassClassificationEvaluator(metricName='f1')
         f1_score2= f1_score.evaluate(results.select(col('label'), col('prediction')))
         #modelsumm=metrics.classification_report(test,results.select(col('label'), col('prediction')))
         #print("The summary of the model", modelsumm)
         print('The F1 score on the test set is {:.2%}'.format(f1_score2))
```

X. MODEL EVALUATION

We use several assessment criteria to measure the prediction outcomes of these diverse strategies in handling the trip prediction problem in order to evaluate their performance. Given that churned users make up a much smaller fraction than engaged users, we choose to utilize the measures listed below to assess the model performance and choose the best model.

For Logistic Regression

- F1 Score = 85.29%

For Gradient Boosting Classifier

- F1 Score = 88.55%

For Random Forest Classifier

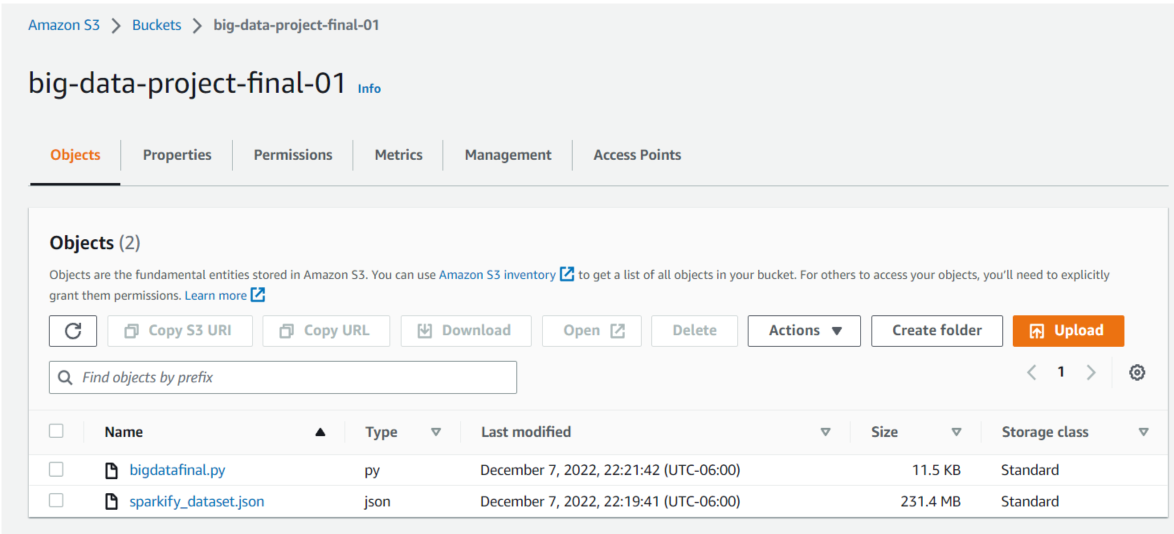
- F1 Score = 83.28%

- Implementing the above trained models and predicting accuracies using them on test dataset

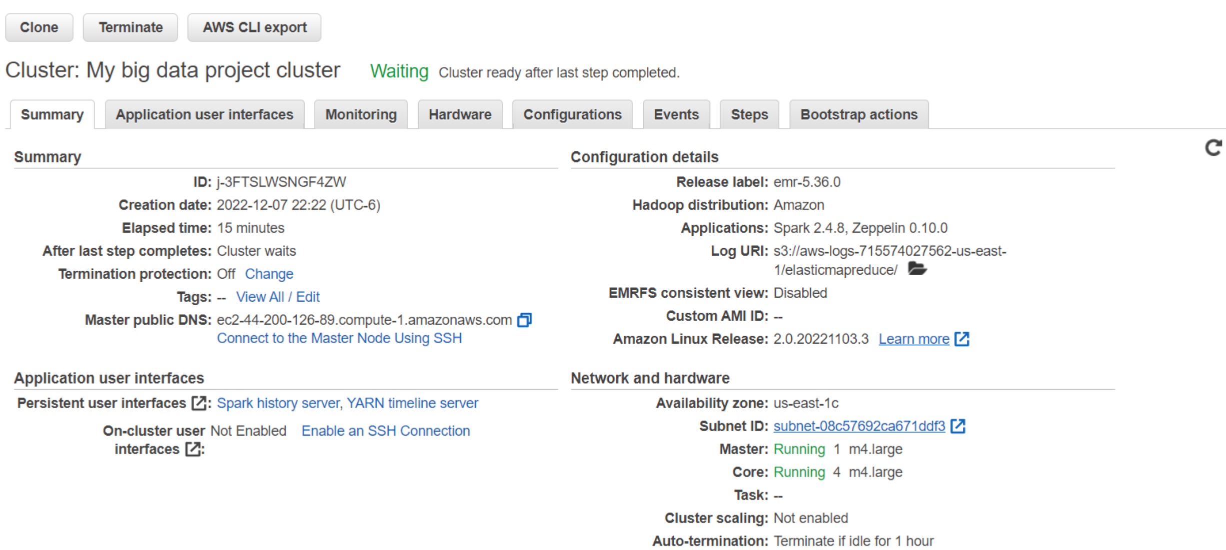
```
## Spliting data into 70-30 Ratio (i.e. 70% Train Dataset, 30% Test Dataset with sandom seed=69)
X_train, X_test = finalmodeldata.randomSplit([0.7, 0.3], seed=69)

[44] # for model in ['logistic_regression','random_forest','gradient_boosting']:
     #     predictionfunction(train, test, model)

[45] model1='Logistic-Regression'
     predictionfunction(X_train,X_test,model1)

     model2='Random-Forest'
     predictionfunction(X_train,X_test,model2)

     model3='Gradient-Boost'
     predictionfunction(X_train,X_test,model3)

     The F1 score on the test set is 85.29%
     The F1 score on the test set is 83.28%
     The F1 score on the test set is 88.55%
```

XI. SPARK CLUSTER IMPLEMENTATION - AWS

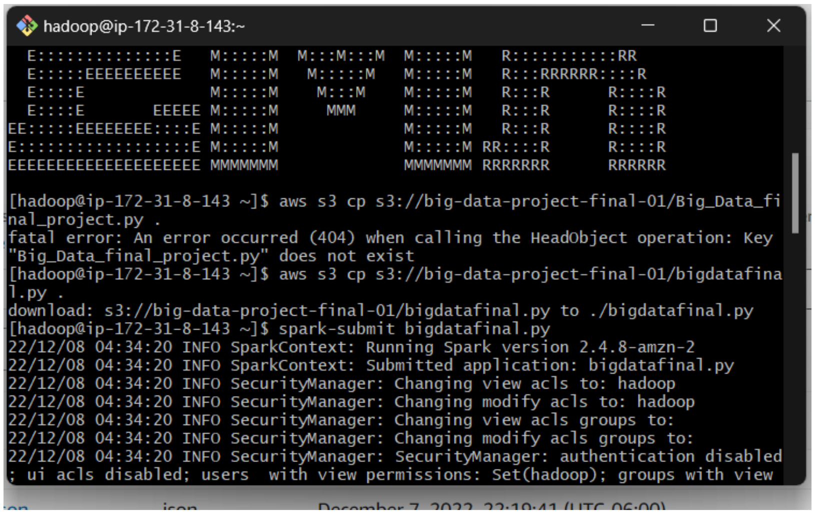
- S3 Bucket (Python script file and Dataset)



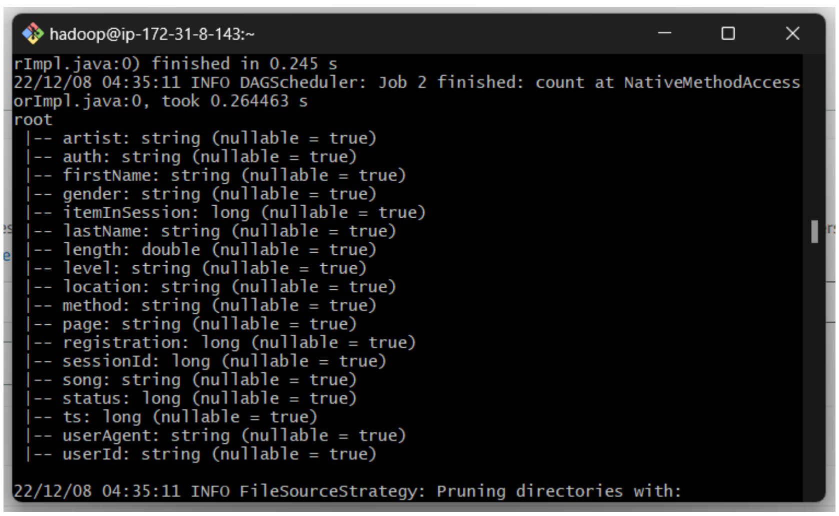
- EMR Cluster using spark



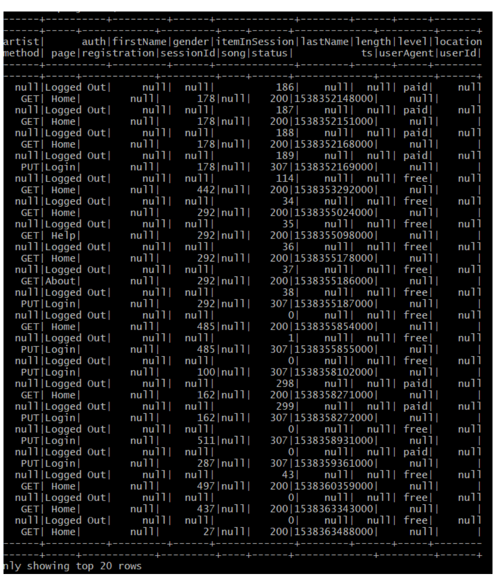
- Running python script using hadoop



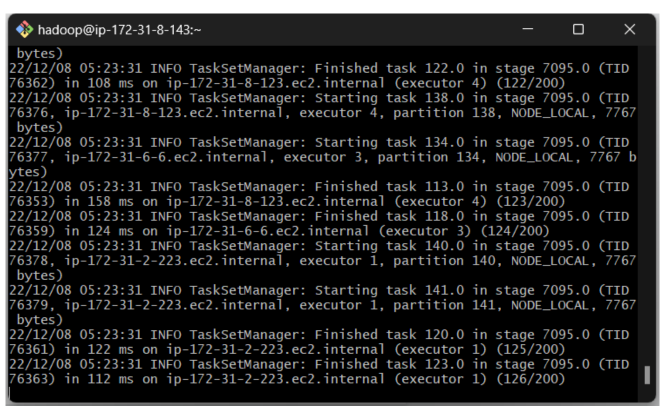
```
hadoop@ip-172-31-8-143:~
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R::::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-8-143 ~]$ aws s3 cp s3://big-data-project-final-01/Big_Data_final_project.py .
fatal error: An error occurred (404) when calling the HeadObject operation: Key "Big_Data_final_project.py" does not exist
[hadoop@ip-172-31-8-143 ~]$ aws s3 cp s3://big-data-project-final-01/bigdatafinal.py .
download: s3://big-data-project-final-01/bigdatafinal.py to ./bigdatafinal.py
[hadoop@ip-172-31-8-143 ~]$ spark-submit bigdatafinal.py
22/12/08 04:34:20 INFO SparkContext: Running Spark version 2.4.8-amzn-2
22/12/08 04:34:20 INFO SparkContext: Submitted application: bigdatafinal.py
22/12/08 04:34:20 INFO SecurityManager: Changing view acls to: hadoop
22/12/08 04:34:20 INFO SecurityManager: Changing modify acls to: hadoop
22/12/08 04:34:20 INFO SecurityManager: Changing view acls groups to:
22/12/08 04:34:20 INFO SecurityManager: Changing modify acls groups to:
22/12/08 04:34:20 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  with view permissions: Set(hadoop); groups with view
```



```
hadoop@ip-172-31-8-143:~
rImpl.java:0) finished in 0.245 s
22/12/08 04:35:11 INFO DAGScheduler: Job 2 finished: count at NativeMethodAccessorImpl.java:0, took 0.264463 s
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)

22/12/08 04:35:11 INFO FileSourceStrategy: Pruning directories with:
```



```
artist|      auth|firstName|gender|itemInSession|lastName|length|level|location
method| page|registration|sessionId|song|status|            ts|userAgent|userId|

  null|Logged Out|     null|  null|          186|    null|  null| paid|    null
   GET| Home|        null|       178|null|   200|1538352148000|     null|      |
  null|Logged Out|     null|  null|          187|    null|  null| paid|    null
   GET| Home|        null|       178|null|   200|1538352151000|     null|      |
  null|Logged Out|     null|  null|          188|    null|  null| paid|    null
   GET| Home|        null|       178|null|   200|1538352168000|     null|      |
  null|Logged Out|     null|  null|          189|    null|  null| paid|    null
   PUT|Login|        null|       178|null|   307|1538352169000|     null|      |
  null|Logged Out|     null|  null|          114|    null|  null| free|    null
   GET| Home|        null|       442|null|   200|1538353292000|     null|      |
  null|Logged Out|     null|  null|           34|    null|  null| free|    null
   GET| Home|        null|       292|null|   200|1538355024000|     null|      |
  null|Logged Out|     null|  null|           35|    null|  null| free|    null
   GET| Help|        null|       292|null|   200|1538355098000|     null|      |
  null|Logged Out|     null|  null|           36|    null|  null| free|    null
   GET| Home|        null|       292|null|   200|1538355178000|     null|      |
  null|Logged Out|     null|  null|           37|    null|  null| free|    null
   GET|About|        null|       292|null|   200|1538355186000|     null|      |
  null|Logged Out|     null|  null|           38|    null|  null| free|    null
   PUT|Login|        null|       292|null|   307|1538355187000|     null|      |
  null|Logged Out|     null|  null|            0|    null|  null| free|    null
   GET| Home|        null|       485|null|   200|1538355854000|     null|      |
  null|Logged Out|     null|  null|            1|    null|  null| free|    null
   PUT|Login|        null|       485|null|   307|1538355855000|     null|      |
  null|Logged Out|     null|  null|            0|    null|  null| free|    null
   PUT|Login|        null|       100|null|   307|1538358102000|     null|      |
  null|Logged Out|     null|  null|          298|    null|  null| paid|    null
   GET| Home|        null|       162|null|   200|1538358271000|     null|      |
  null|Logged Out|     null|  null|          299|    null|  null| paid|    null
   PUT|Login|        null|       162|null|   307|1538358272000|     null|      |
  null|Logged Out|     null|  null|            0|    null|  null| free|    null
   PUT|Login|        null|       511|null|   307|1538358931000|     null|      |
  null|Logged Out|     null|  null|            0|    null|  null| paid|    null
   PUT|Login|        null|       287|null|   307|1538359361000|     null|      |
  null|Logged Out|     null|  null|           43|    null|  null| free|    null
   GET| Home|        null|       497|null|   200|1538360359000|     null|      |
  null|Logged Out|     null|  null|            0|    null|  null| free|    null
   GET| Home|        null|       437|null|   200|1538363343000|     null|      |
  null|Logged Out|     null|  null|            0|    null|  null| free|    null
   GET| Home|        null|        27|null|   200|1538363488000|     null|      |

nly showing top 20 rows
```



```
hadoop@ip-172-31-8-143:~
 bytes)
22/12/08 05:23:31 INFO TaskSetManager: Finished task 122.0 in stage 7095.0 (TID 76362) in 108 ms on ip-172-31-8-123.ec2.internal (executor 4) (122/200)
22/12/08 05:23:31 INFO TaskSetManager: Starting task 138.0 in stage 7095.0 (TID 76376, ip-172-31-8-123.ec2.internal, executor 4, partition 138, NODE_LOCAL, 7767 bytes)
22/12/08 05:23:31 INFO TaskSetManager: Starting task 134.0 in stage 7095.0 (TID 76377, ip-172-31-6-6.ec2.internal, executor 3, partition 134, NODE_LOCAL, 7767 bytes)
22/12/08 05:23:31 INFO TaskSetManager: Finished task 113.0 in stage 7095.0 (TID 76353) in 158 ms on ip-172-31-8-123.ec2.internal (executor 4) (123/200)
22/12/08 05:23:31 INFO TaskSetManager: Finished task 118.0 in stage 7095.0 (TID 76359) in 124 ms on ip-172-31-6-6.ec2.internal (executor 3) (124/200)
22/12/08 05:23:31 INFO TaskSetManager: Starting task 140.0 in stage 7095.0 (TID 76378, ip-172-31-2-223.ec2.internal, executor 1, partition 140, NODE_LOCAL, 7767 bytes)
22/12/08 05:23:31 INFO TaskSetManager: Starting task 141.0 in stage 7095.0 (TID 76379, ip-172-31-2-223.ec2.internal, executor 1, partition 141, NODE_LOCAL, 7767 bytes)
22/12/08 05:23:31 INFO TaskSetManager: Finished task 120.0 in stage 7095.0 (TID 76361) in 122 ms on ip-172-31-2-223.ec2.internal (executor 1) (125/200)
22/12/08 05:23:31 INFO TaskSetManager: Finished task 123.0 in stage 7095.0 (TID 76363) in 112 ms on ip-172-31-2-223.ec2.internal (executor 1) (126/200)
```

XII. OBSERVATIONS AND CONCLUSION

With regard to a fictitious music streaming business, we wished to forecast subscriber churn. Each stage of the machine learning workflow uses Apache Spark. For that, a binary classifier for Churner and Engaged clients was required.

In order to eliminate log events without a user ID, we first cleaned the data and looked for any missing values in the dataset. Then, we conducted several data analyses to see how different indicators may aid in differentiating between Churned and Engaged consumers. Based on whether a user visited the sites for cancellation confirmation and downgrade submission or not, we determined the customer churn indicator. We then retrieved categorical and numerical characteristics during the features engineering process. In order to do that, we made advantage of the data exploration's observed indications. We also looked at the previous 20 days of service use to indicate the user's behavior prior to the churn event based on the number of sessions and songs they listened to daily.

We divided the data into sets for training and validation. We tested a variety of models—Logistic Regression, Random Forest, and Gradient-Boosted Trees—ranging in complexity from simple to complicated. Finally, we selected the “Gradient boosting model “ with 88.55% accuracy the winning model by comparing their evaluation metrics.

XIII. FUTURE WORKS

We believe that we could test other models and methods. However, in order to have a more accurate model for determining if a customer is likely to churn or not, we would like to perform more extensive data investigation and feature engineering first. Additionally, we worked with a small dataset at this time, but if we have the opportunity in the future, we can work with a larger dataset to get findings that are more precise. In exchange, we would apply more Spark best practices to improve the data analysis and feature engineering procedures for effective data exploration, model training, and model testing.

Due to potential statistical discrepancies with the huge dataset, do data exploration on larger batches of data subsets before utilizing the big dataset. Furthermore, tweaking models using better hyperparameters on the Spark Cluster with more processing power helps in boosting the performance and the accuracy of the model.

XIV. REFERENCES

- Shoro, Abdul & Soomro, Tariq. (2015). Big Data Analysis: Apache Spark Perspective. Global Journal of Computer Science and Technology. 15.
- Khedikar, Kanchan A., Data Analytics for Business Using Tableau (April 27, 2021). Proceedings of the International Conference on Innovative Computing & Communication.
- Salman Salloum, Ruslan Dautov, Xiaojun Chen1, Patrick Xiaogang Peng1, Joshua Zhexue Huang, Big data analytics on Apache Spark.
- N Balaji, B H Karthik Pai, Bhaskar Bhat and Barmavatu Praveen, Data Visualization in Splunk and Tableau: A Case Study Demonstration

Project Link:

https://github.com/Jasleen-bots21/CSP_554-Customer-Churn-Prediction-on-Sparkify-Dataset