My role:

Working as an employee at Saavn as a Big Data engineer working closely with the company's ML team.

Problem Statement:

Build a system that keeps the users updated based on their music preferences. Suppose, a new track of some particular artist has been released. Now, my responsibility would be to push the notification about this song to the appropriate set of audience. But the challenge is to push a song notification only to its interested and relevant audience.

Objective:

In this problem, I need to group users based on their listening patterns. The data that has been provided to me does not contain any explicit information such as age, genre preferences, gender etc.

The data provides me with information about users' click history, and I need to build a model that can learn from implicit features and group similar users together. This problem revolves around training models based on inherent behaviours.

References:

https://spark.apache.org/docs/latest/ml-guide.html

https://spark.apache.org/docs/2.3.0/ml-collaborative-filtering.html

https://jessesw.com/Rec-System/

StackOverflow

Based on the references, I arrived at the conclusion that I will make use of Spark MLLib to build this model and train it based on implicit features. The various libraries of Spark MLLib will be used for data handling, clustering, feature extraction, transformation, dimensionality reduction, selection and collaborative filtering. I will be making use of ALS algorithm for deriving implicit feedback from the dataset provided.

1. Analysing the datasets:

Four datasets are provided by Saavn as under:

User Click-Stream Activity

Attributes: "User ID", " Timestamp ", "Song ID", and "Date"

MetaData

Attributes: "Song ID" and "Artist ID"

A single "Song ID" can be linked to multiple "Artist IDs"

Notification Clicks

Attributes: "Notification ID", "User ID", and "Date"

Notification Artists

Attributes: "Notification ID" and "Artist ID"

One "Notification ID" can be mapped to multiple artists

2. Working on the datasets: Training the Model

The **clickstream activity** data is read into a Dataset. I would only use three columns – UserID, SongID, Date. For collaborative filtering, I would need a rating of some kind for each user/item interaction that occurred . In this case there is no explicit rating available so I would use UserId and SongId to get the frequency(no. of times) with user has heard a song. [Implicit: Not as obvious in terms of preference, such as a click, view, or purchase]

I will make use of **ALS (Alternating Least Squares)** method for collaborative filtering.

(i) Derive **frequency** to find how many times a user heard a particular song.
(ii) Since the userID and songID are String fields, ALS will need numeric fields to work on. Hence, I derived userIndex and songIndex using **StringIndexer** class.
(iii) setImplicitPrefs is set to **True .** Setting the parameters- setRank , setMaxIter to values as below:

Using below lines of code to feed to ALS:

```
ALS als = new ALS()
        .setRank(10)
        .setMaxIter(5)
        .setImplicitPrefs(true)
        .setUserCol("userIndex")
        .setItemCol("songIndex")
        .setRatingCol("Frequency");
```

When the dataset is fitted into the ALS model, it results in a dataset which has **userID** (based on **userIndex** passed as the parameter in **setUserCol()**) and the **factors** that can be considered as features for the further clustering models.

I made use of **VectorUDT** to convert the factors obtained from ALS to a feature vector.

After obtaining the feature vector, feed it into **K-means algorithm** , one of the most popular algorithms for clustering to get the clusters predictions.

After various trials of values for **K (150 , 200 , 250 , 300)** , I concluded that **K=300** gives me the best results on the dataset provided.

After K-means algorithm runs on the dataset with userID and features vector, it results in appropriate cluster predictions for each user. Now the dataset contains every userID corresponding to a predicted cluster.

I then derive the ***UserID in string format*** (as was given in raw dataset) ***from the userID column which is in numeric format in current dataset*** by joining the appropriate datasets.

Another raw data named – **Metadata** which contains SongID and ArtistID is now read into another dataset.

A join operation on Metadata and the dataset obtained above will give the **final predicted dataset** having 4 columns :

```
"SongID" , "ArtistID" , "prediction" , "UserID"
```

I use this dataset to arrive at 2 extremely important datasets:

1. **Finding how many unique users each predicted cluster has.**
   This is a 2 step process – find [predictedcluster and UserID] groups. Then groupby predictedcluster to find count of unique users.
   Step 1 will give:

C1	U1	1
C1	U2	1
C2	U2	1
C2	U3	1

   Step 2 will give:

C1	2
C2	2

2. **Finding the most popular artist in each predicted cluster.**
   This is a 2-step process – find [predictedcluster and ArtistID] groups and get the count. Then use WindowSpec to use row_number() to find the artist with highest count.

   *This will resolve the cases when there are multiple Artists associated with a cluster.
   *This will resolve the cases when there are ties in the Artist IDs in the cluster.

3. Validating the Model :

To validate the model , I need to maximize the click-through rate of each cluster.
Clickthrough rate = No. of users who actually clicked on the notification received / No. of users who the notification was sent to.

For getting this information, a series of steps has been performed:

(i) Find out the dataset which is the resultant of **NotificationArtist** data received from Saavn and the **PopularArtist** (derived by me in training section) in each cluster. This will have 3 columns:
NotificationID , ArtistID , PredictedCluster

(ii) Find a new dataset which is the resultant of the **final predicted dataset** from training section and the one which we derived in (i). Keep only UserID , NotificationID in this.

(iii) Groupby the dataset in (ii) by NotificationID to predict the count of Users who will receive this notification.

(iv) Find a dataset which is the resultant of **dataset derived in (ii)** and the **NotificationClick** data received from Saavn.

(v) Groupby the dataset in (iv) by NotificationID to get the actual count of users who clicked on the notification.
(vi) Perform a join on datasets in (iii) and in (v) to get NotificationID , PredictedCount and ActualCount of users.
(vii) CTR can be calculated by dividing the ActualCount by PredictedCount.
(viii) The final resultant dataset with CTR was copied into a csv file on the specified location.

PLEASE NOTE: After various runs, I came to the conclusion that K=300 will help me get a better Clickthrough rate.

4. Running / Testing the Code:

Step 1:

Once the solution has been developed in IDE, I first made use of smaller datasets(100 MB) to test the working of the solution, with datasets copied to my local.

Step 2:

Once the program was stable, I moved to testing with actual datasets (1 GB) along with entire datasets for newmetadata , notificationclick and notificationartist , all copied to my local.

Step 3:

Once overall results were achieved , I created the jar file pointing to the S3 buckets where data was stored.

Creating a FAT Jar:

Right click on project -> Run As -> Maven Install

Once the build is successful , a jar will be created under the target folder.

Either transfer this JAR from your machine to EC2 instance through **WinSCP** OR upload it on **S3 bucket** and then get it on your EC2 instance with following command in **root** directory.

aws s3 cp s3://musicrecommendation-dee/MusicRecommendation-0.0.1-SNAPSHOT.jar .

Setting : Go to Cloudera Manager, in Spark2 -> Configurations, search for *Lineage.*

Set the path as: **/tmp**

On EC2 instance, in *root* , run the following command:

spark2-submit --class com.deepika.MusicRecommendation.ClickAnalysis --master yarn --deploy-mode client --name saavnAnalytics --conf "spark.app.id=app01 spark.driver.memory=12g spark.executor.memory=12g spark.executor.instances=2" MusicRecommendation-0.0.1-SNAPSHOT.jar <AWS Access Key> <AWS Secret key> <OutputFileDirectory>

PLEASE NOTE: In the above command three parameters need to be necessarily specified, without fail.

Music Recommendation- ReadMe

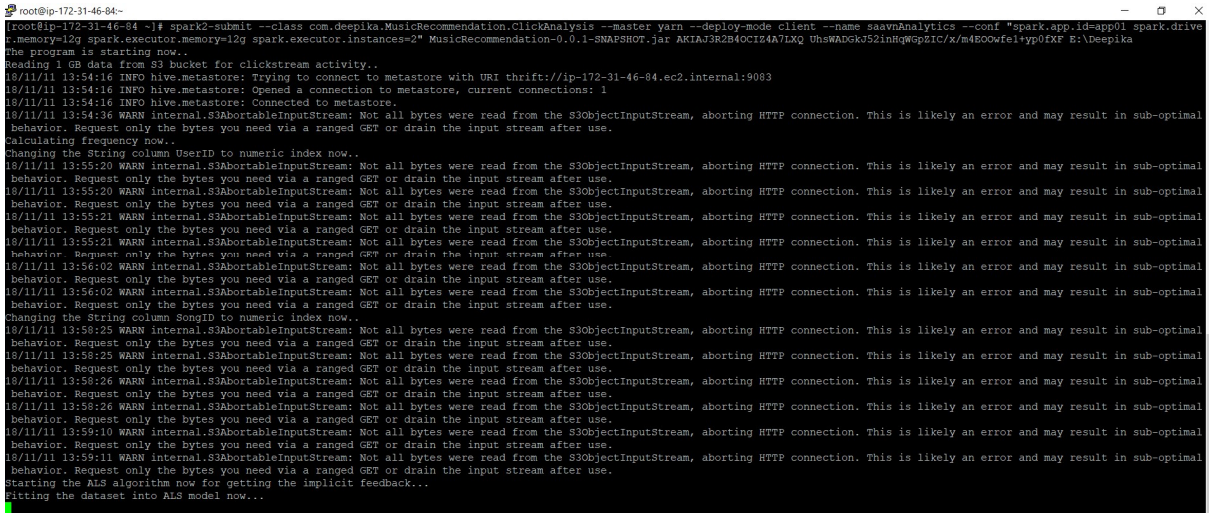
<AWS Access Key> -

<AWS Secret key>

<OutputFileDirectory>

Spark.driver.memory and Spark.exeutor.memory need to be set to 12gb for this specific program.

Once the command is specified correctly, program will start running as below:



****The final CTR results I am attaching as program output have been obtained after run on local on 1 GB data.***

However, on EC2 instance , the program failed with java.lang.OutOfMemoryError: GC overhead limit exceeded error. Hence, the entire program on EC2 was run with 100MB sample data.

5. **Once the program is complete, shut down the EC2 instance.**