CS724 High Performance Computing and Big Data

Course Project Implementation Details

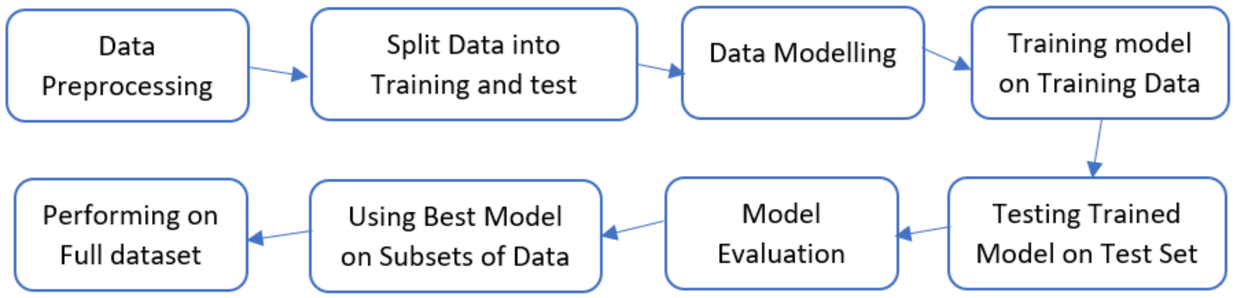
Customer Churn Prediction using PySpark

Herambeshwar Pendyala | 01130541

1. Design

For model development, most of the steps such as data understanding, feature engineering and model selection, were performed on a representative sample (1/100) of the full dataset, using Spark in local mode. Models that performed well on the smaller sample were trained and tested also other iterations of the dataset as well as on the full dataset.

Below is the flow chart showing the flow of the development process.


Data Preprocessing
Split Data into Training and test
Data Modelling
Training model on Training Data
Testing Trained Model on Test Set
Model Evaluation
Using Best Model on Subsets of Data
Performing on Full dataset

2. Implementation

2.1. Data Cleaning, Preprocessing and Exploratory Data Analysis

To transform the original dataset which has one row per user log to a dataset with user-level information or statistics one row per user, I have used dataframe from sparkSQL to represent the data in a table format similar to relational database which makes it easy to do data analysis. The final dataset is obtained through mapping (e.g. user's gender, start/end of the observation period, etc.) or aggregation (e.g. song count, advertisement count, etc.) using the functions of spark sql component.

Using SparkSQL dataframe for data preprocessing as follows,

I started with data cleaning as there are missing values in the columns userId and sessionId. I need at least the userId to feed the machine learning algorithm with user-specific input features and labels. So, as there is no possibility to keep these rows e.g. through imputing the values for IDs, I decided to drop the associated rows from the dataset. Below are the functions I have used for preprocessing

Udf – spark provides us with udf to write scalable functions

Window - we use this function to perform a calculation over a set of rows.

Defining Churn

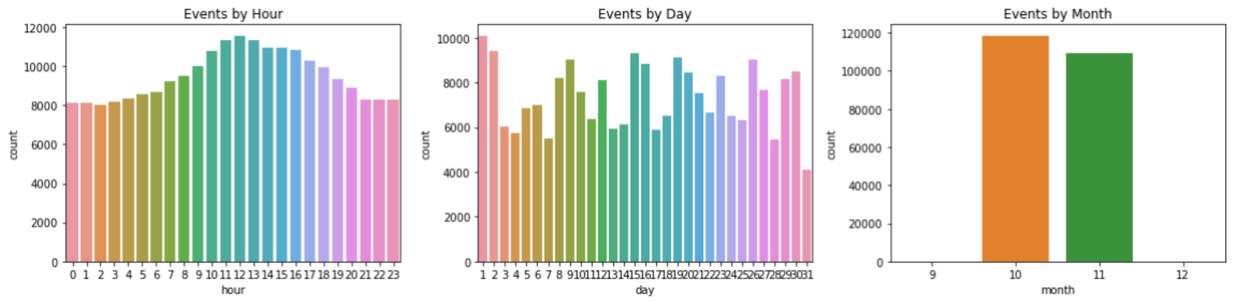
As no label specifically as 'churn' was given to predict, In this application, the label is defined by the Cancellation Confirmation or downgrade events to identify user churn, which happen for both paid and free users. All the features need to be engineered to identify churned users, e.g. aggregated statistic per unit of time, number of plan changes, songs played vs. total activity ratio, thumbs up vs. thumbs down ratio, activity trend, etc. Once churn is defined churn, I will further perform some exploratory data analysis to observe the behavior for users who stayed vs users who churned.

In the user activity log, column 'Page' stores the user activity on pages he visited for a session, I have used this column to see if the user has visited the cancellation confirmation page or downgrade page. If the user visited these pages, then they are likely to be considered as churned.

Defining and calculating the churn, I consider this as binary variable: 1 — users who cancelled their subscription, and 0 — users who kept the service throughout a time period.

Exploring the timestamp column to get insights about user behavior.

Timestamp 'ts' is one of the important columns in user event data as it gives us more insights about the user like daily stats. Below is the graph showing daily stats of the users went to 'nextsong' page in the months of October and November of 2018.



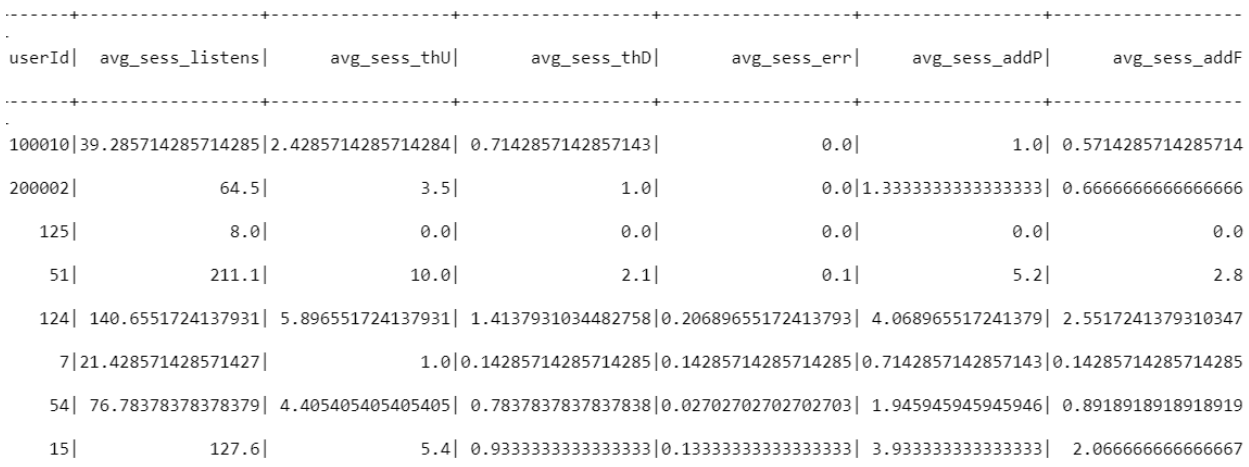
By extensively exploring this data by aggregation of pages like 'nextSong' on different days we can get features such as consecutive_listening_days, days_since_last_listen, total_listens.

I also took a look at user behavior from other pages like `*thumbs-up*`, specifying a like for the song, which can lead the user to add a song to a playlist, where `*add-to-playlist*` is another page. `*thumbs-down*`, specifying the dislike for a song, `*add-friends*` to the profile and share playlists and songs with other users, `*error*` page which specifies how often a user is not able to find his needs in the application. Features were computed by aggregating the page visits of the users to these pages.

From the above observations, the features calculated are consecutive_listening_days, days_since_last_listen, total_listens, total_thumbs_up, total_thumbs_down, total_errors, total_add_to_playlists, total_add_friends.

Exploring user session data

Another potentially useful indicator is the extent to which users behave within each session. Starting by taking the total sum of each flag behavior calculated above (listens, likes, dislikes, friends and playlists). Then taking the average over all each user's session to get a sense of how a user tends to behave in one session. Below image displays the result of those aggregations.



```
------+------------------+------------------+-------------------+-------------------+------------------+-------------------
userId|  avg_sess_listens|       avg_sess_thU|        avg_sess_thD|        avg_sess_err|       avg_sess_addP|       avg_sess_addF
------+------------------+------------------+-------------------+-------------------+------------------+-------------------
100010|39.285714285714285|2.4285714285714284| 0.7142857142857143|                0.0|               1.0| 0.5714285714285714
200002|              64.5|               3.5|                1.0|                0.0|1.3333333333333333| 0.6666666666666666
   125|               8.0|               0.0|                0.0|                0.0|               0.0|                0.0
    51|             211.1|              10.0|                2.1|                0.1|               5.2|                2.8
   124| 140.6551724137931| 5.896551724137931| 1.4137931034482758|0.20689655172413793| 4.068965517241379| 2.5517241379310347
     7|21.428571428571427|               1.0|0.14285714285714285|0.14285714285714285|0.7142857142857143|0.14285714285714285
    54| 76.78378378378379| 4.405405405405405| 0.7837837837837838|0.02702702702702703| 1.945945945945946| 0.8918918918918919
    15|             127.6|               5.4| 0.9333333333333333|0.13333333333333333| 3.933333333333333|  2.066666666666667
```

This brings our features list to 15 which include gender, consecutive_listening_days, days_since_last_listen, total_listens, total_thumbs_up, total_thumbs_down, total_errors, total_add_to_playlists, total_add_friends. avg_sess_listens, avg_sess_thU, avg_sess_thD, avg_sess_err, avg_sess_addP, avg_sess_addF. The features are aggregations of user daily stats and session data.

I have performed exploratory data analysis, comparing the engineered statistics for users who stayed vs users who churned. Below are the graphs for the same.

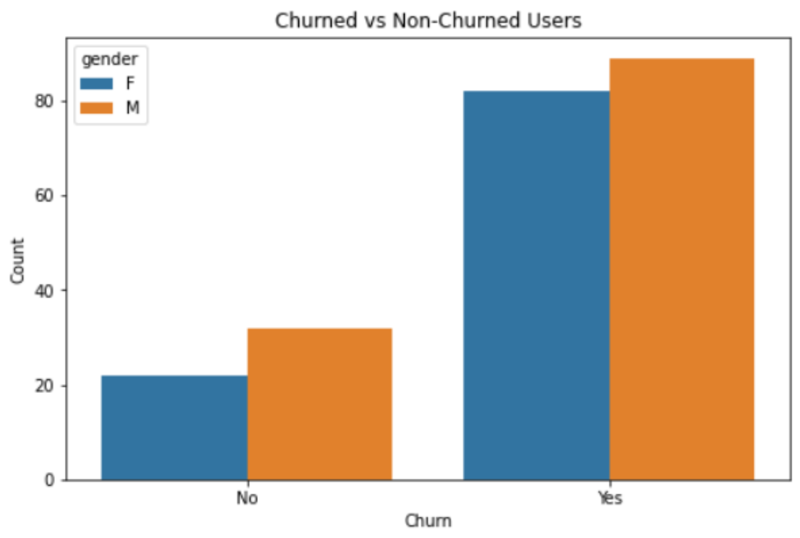


Figure: Churned vs Non-Churned users based on Gender

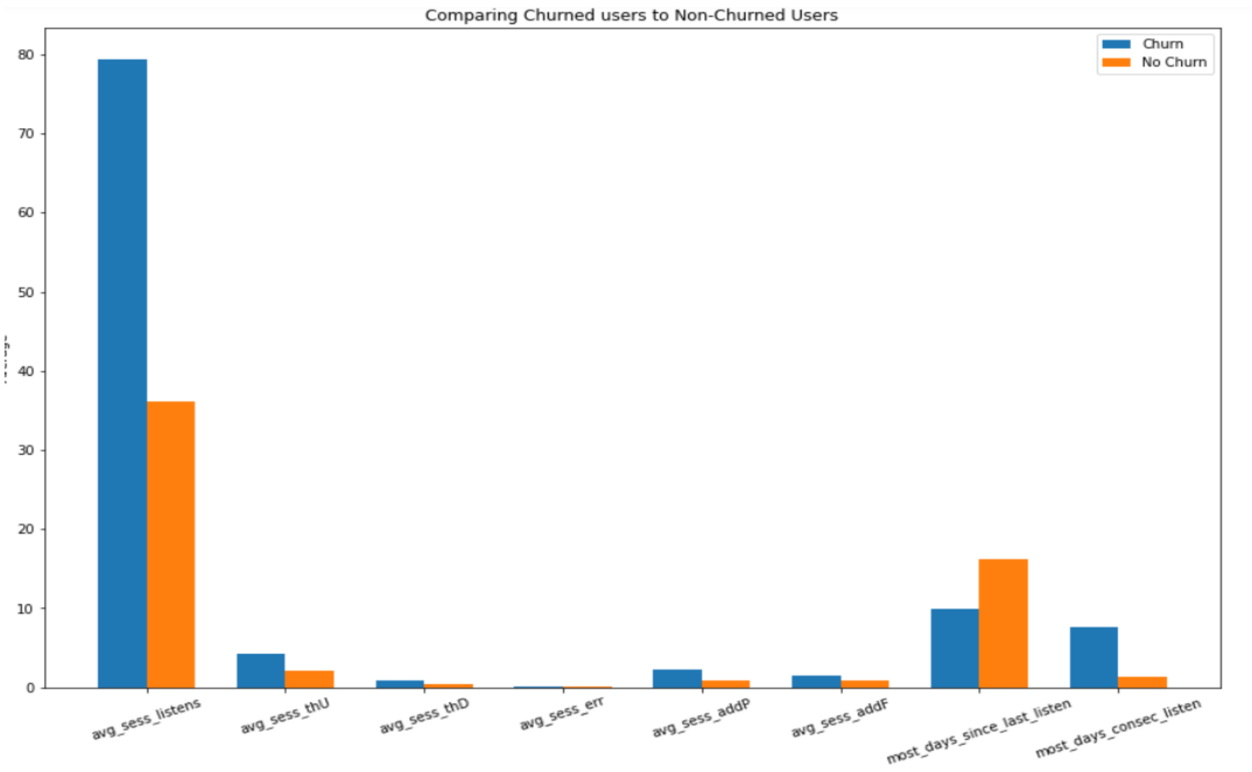


Figure: Scalability – churned vs non churned on session features.

From the preprocessing I have reduced the number of rows from 286500 to 225. Now our data has onl y one user data per row with all the engineered features. Using this data, I built a prediction model to p redict the customer churn. Steps and functions used in model building are described in the next section .

```
+-----+--------+
|churn|count(1)|
+-----+--------+
|    1|     171|
|    0|      54|
+-----+--------+
```

2.2. Model Building

As this problem is termed as a classification problem, the data is split into a training set and a test set. I am using Logistic regression to build prediction model and measure the performance of the algorithm on the dataset of different sizes.

Using PySpark for machine learning,

Logistic Regression

Mathematical expression of the algorithm**:**

For one example $x^{(i)}$:

$$z^{(i)} = w^T x^{(i)}$$
$$\widehat{y^{(i)}} = a^{(i)} = sigmoid\left(z^{(i)}\right)$$

$$\mathcal{L}(a^{(i)}, y^{(i)}) = -y^{(i)} \log(a^{(i)}) - (1 - y^{(i)}) \log(1 - a^{(i)})$$

The cost is then computed by summing over all training examples:

$$J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y^{(i)})$$

To minimize the cost(J), I used gradient descent as an optimization method, the goal is to learn w and b by minimizing the cost function J. For a parameter θ, the update rule is $\theta = \theta - \alpha \, d\theta$, where α is the learning rate.

Using the above mathematical expressions, below is the pseudocode using RDD MapReduce to perform gradient descent.

```
For given number of iterations :
    Gradient = dataRDD.map(calculate cost).reduce(add)
    # update weights
   $W = W - \alpha * Gradient$
```

After it learn the parameters, they are used to predict the labels for the dataset, by using below equation.

$$\hat{Y} = A = \sigma(w^T X)$$

Convert the entries of ***A*** into 0 (if activation <= 0.5) or 1 (if activation > 0.5), stores the predictions in a vector ***Y_prediction***.

2.3. Model Evaluation

I have used below performance metrics to evaluate my model.

Confusion Matrix

Confusion matrix is calculated from comparing true values with the predicted values. This gives us four values namely.

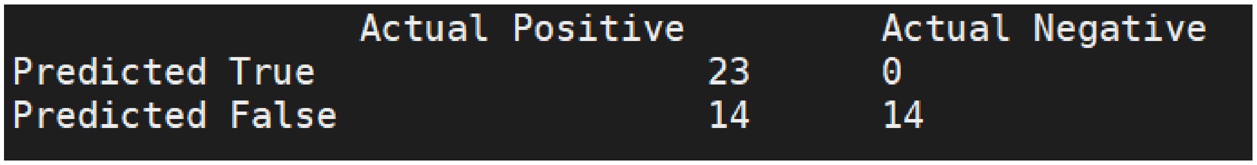
true positives(tp) – count of values that are true and predicted true

false positives(fp) – count of values that are false but predicted true

false negatives(fn) – count of values that are true but predicted false

true negatives(tn) – count of values that are false and predicted false

below is the confusion matrix calculated for the test dataset.



```
                Actual Positive        Actual Negative
Predicted True                 23      0
Predicted False                14      14
```

Accuracy

Accuracy tells us how well out model performs on the test dataset. Below is the formula for accuracy.

Accuracy = TP+TN/(TP+FP+FN+TN)

Precision

Ration of correctly predicted positive observations to the total predicted positive observations.

Precision = TP/TP+FP

Recall

Ratio of correctly predicted positive observations to total predicted positive observations.

Recall = TP/TP+FN

F1 – Score

This is weighted average of precision and recall.

F1 score = 2*(Precision*Recall)/(Precision + Recall)