Music Streaming Service Churn Prediction

Group No – 220, Monika Pokharna, Manisha, Aditya Sharma

Introduction To Data Science

M. Tech Data Science and Engineering – Cluster Batch 4

Objective:

Predict if subscription users of a music streaming service will churn or stay after their current membership expires.

DataSet Details:

The datasets are downloaded from :

https://www.kaggle.com/c/kkbox-churn-prediction-challenge/data

Feature Details:-

1. **members_v3.csv:-**
   Size of dataset – 6769472 records, 408MB size on HD
   Type of Data –

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6769473 entries, 0 to 6769472
Data columns (total 6 columns):
 #   Column                  Dtype
---  ------                  -----
 0   msno                    object
 1   city                    int64
 2   bd                      int64
 3   gender                  object
 4   registered_via          int64
 5   registration_init_time  int64
dtypes: int64(4), object(2)
memory usage: 309.9+ MB
```

2. **user_logs_v2.csv :-**
   Size of dataset – 18396361 records, 1.33GB size on HD
   Type of Data –

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18396362 entries, 0 to 18396361
Data columns (total 9 columns):
 #   Column      Dtype
---  ------      -----
 0   msno        object
 1   date        int64
 2   num_25      int64
 3   num_50      int64
 4   num_75      int64
 5   num_985     int64
 6   num_100     int64
 7   num_unq     int64
 8   total_secs  float64
dtypes: float64(1), int64(7), object(1)
memory usage: 1.2+ GB
```

3. **transactions_v2.csv :-**
   <u>Size of dataset</u> – 1431008 records, 1.61GB size on HD
   <u>Type of Data</u> –

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1431009 entries, 0 to 1431008
Data columns (total 9 columns):
 #   Column                  Non-Null Count    Dtype
---  ------                  --------------    -----
 0   msno                    1431009 non-null  object
 1   payment_method_id       1431009 non-null  int64
 2   payment_plan_days       1431009 non-null  int64
 3   plan_list_price         1431009 non-null  int64
 4   actual_amount_paid      1431009 non-null  int64
 5   is_auto_renew           1431009 non-null  int64
 6   transaction_date        1431009 non-null  int64
 7   membership_expire_date  1431009 non-null  int64
 8   is_cancel               1431009 non-null  int64
dtypes: int64(8), object(1)
memory usage: 98.3+ MB
```

4. **train_v2.csv :-**
   <u>Size of dataset</u> – 970960 records, 43.5MB size on HD
   <u>Type of Data</u> –

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 970960 entries, 0 to 970959
Data columns (total 2 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   msno      970960 non-null  object
 1   is_churn  970960 non-null  int64
dtypes: int64(1), object(1)
memory usage: 14.8+ MB
```

Preprocessing Challenges :-

Missing/Invalid data - "**gender**" and "**bd**" in "members_v3.csv" had more than 65% of missing/invalid data, hence these attributes are dropped.
Data format - "**date**" in user_logs_v2.csv, and "**transaction_date**" & "**membership_expire_date**" in "transactions_v2.csv" were having dates in integer format. Converted these attributes to Date having "%Y%m%d" format using Panda's datetime utility.

Feature Engineering Techniques:

1. **sample_submission_v2.csv** - This is the test data set. No missing value & no duplicates. No feature engineering technique applied here.
2. **train_v2.csv** - This is the train data set. No missing value & no duplicates. No feature engineering technique applied here.
3. **members_v3.csv -**
    a. **Feature Creation**:
        i. **Hot Encoding**:
            1. **registered_via attribute** : The attribute signifies the registration method and hence its a categorical one. Thus perform **hot encoding** to create new **binary** features. Each new feature represents if the user has done the registration via that method or not.
            2. **city**: City of the user. Thus it is a categorical feature. Performed **hot encoding** to create new **binary feature** for each city. Each new feature represents if the user belongs to that city or not.

```
# Hot Encoding

# 1. registered_via attribute

encoded_columns = pd.get_dummies(member_df['registered_via'], prefix='registered_via')
member_df = member_df.join(encoded_columns)

# 2. city attribute

encoded_columns = pd.get_dummies(member_df['city'], prefix='city')
member_df = member_df.join(encoded_columns)


member_df.head()
```

b. **Feature Removed**:
    i. Drop **'city'** & '**egistered_via attribute'** as we have created more meaningful attributes out of them.

```
# drop unnecessary columns
member = member_df.drop(['city', 'registered_via'], 1)
member
```

    ii. **Gender** & **bd** columns are dropped in the preprocessing step as 65% of values are missing.

4. **user_logs_v2.csv** -
    a. **Aggregation** - There are **multiple entries** for some users. Aggregate them.

```
# Aggregate multiple user enteries

user_logs_df = user_logs_df.groupby('msno', as_index=False).sum()
user_logs_df
```

	msno	num_25	num_50	num_75	num_985	num_100	num_unq	total_secs
0	+++IZseRRiQS9aaSkH6cMYU6bGDcxUieAi/tH67sC5s=	86	11	10	5	472	530	117907.425
1	+++hVY1rZox/33YtvDgmKA2Frg/2qhkz12B9ylCvh8o=	191	90	75	144	589	885	192527.892
2	+++l/EXNMLTijfLBa8p2TUVVVp2aFGSuUl/h7mLmthw=	43	12	15	12	485	468	115411.260
3	+++snpr7pmobhLKUgSHTv/mpkqgBT0tQJ0zQj6qKrqc=	207	163	100	64	436	828	149896.558
4	++/9R3sX37CjxbY/AaGvbwr3QkwElKBCtSvVzhCBDOk=	105	24	39	35	479	230	116433.247

    b. **Feature Extracted**:
        i. **Scaling**: Convert attributes like '**num_25**', '**num_50**' etc from absolute number to fraction.

```
user_logs_df['total_songs_played'] = user_logs_df['num_25'] + user_logs_df['num_50'] + user_logs_df['num_75'] + user_lc

# Scaling - Convert num to fraction

user_logs_df['fraction_25'] = user_logs_df['num_25'] / user_logs_df['total_songs_played']
user_logs_df['fraction_50'] = user_logs_df['num_50'] / user_logs_df['total_songs_played']
user_logs_df['fraction_75'] = user_logs_df['num_75'] / user_logs_df['total_songs_played']
user_logs_df['fraction_985'] = user_logs_df['num_985'] / user_logs_df['total_songs_played']
user_logs_df['fraction_100'] = user_logs_df['num_100'] / user_logs_df['total_songs_played']


user_logs_df
```

        ii. **Binning** :

1. **num_unq**: The feature represents the number of unique songs played by the user. Divide the users in categories based on bins like **0-25%, 25-50%** etc.
2. **total_secs**: The feature represents the total seconds played by the user. Divide the users in categories based on bins like **0-25%, 25-50%** etc.

```
# Binning

# 1. Perform binning on num_unq attribute
bin_labels_unq_song = ['low_unq_song', 'lower_middle_unq_song', 'upper_middle_unq_song', 'high_unq_song']
user_logs_df['num_unq_categ'] = pd.qcut(user_logs_df['num_unq'], q=[0, 0.25, 0.5, 0.75, 1], labels = bin_labels_unq_son

# 2. Perform binning on total_secs attribute
bin_labels_total_sec = ['low_total_sec', 'lower_middle_total_sec', 'upper_middle_total_sec', 'high_total_sec']
user_logs_df['total_sec_categ'] = pd.qcut(user_logs_df['total_secs'], q=[0, 0.25, 0.5, 0.75, 1], labels = bin_labels_tc

user_logs_df
```

iii. **Hot Encoding**: In the previous step, we have created categories for **num_unq** & **total_secs** attributes. Perform hot encoding to create new binary attributes representing each category.

```
# Hot Encoding

# 1. num_unq_categ

encoded_columns = pd.get_dummies(user_logs_df['num_unq_categ'])
user_logs_df = user_logs_df.join(encoded_columns)

# 2. total_sec_categ
encoded_columns = pd.get_dummies(user_logs_df['total_sec_categ'])
user_logs_df = user_logs_df.join(encoded_columns)

user_logs_df
```

c. **Feature Removed**

i. Drop **'num_25', 'num_50','num_75', 'num_985', 'num_100', 'num_unq', 'num_unq_categ', 'total_sec_categ', 'total_secs', 'total_songs_played'** as we have created more meaningful attributes out of them.

```
# drop unnecessary columns
user_logs = user_logs_df.drop(['num_25', 'num_50','num_75', 'num_985', 'num_100', 'num_unq', 'num_unq_categ', 'total_se
user_logs
```

5. **transactions_v2.csv** -

a. **Aggregation** : There are multiple entries for the transactions of users. Therefore, we should aggregate the users based on recent membership_expire_date and other aggregate functions for the rest of the columns to preserve the knowledge represented by different entries.

```python
# Aggregate multiple user entries

transactions_df.sort_values('membership_expire_date')

transactions_df['total_order'] = 1

transactions_df_grouped = transactions_df.groupby('msno').agg(
    total_orders=pd.NamedAgg(column='total_order', aggfunc='sum'),
    plan_list_price_total=pd.NamedAgg(column='plan_list_price', aggfunc='sum'),
    payment_plan_days_total=pd.NamedAgg(column='payment_plan_days', aggfunc='sum'),
    actual_amount_paid_total=pd.NamedAgg(column='actual_amount_paid', aggfunc='sum'),
    actual_amount_paid_mean=pd.NamedAgg(column='actual_amount_paid', aggfunc='mean'),
    auto_renew_times=pd.NamedAgg(column='is_auto_renew', aggfunc=lambda x : sum(x==1)),
    cancel_times=pd.NamedAgg(column='is_cancel', aggfunc=lambda x : sum(x==1)),
    membership_expire_date_recent=pd.NamedAgg(column='membership_expire_date', aggfunc='max')
)
```

b. **Normalization** : Normalize the attributes between 0-1

```python
# Normalization

# 1. total_orders
transactions_df_grouped['total_orders'] = (transactions_df_grouped['total_orders'] - transactions_df_grouped['total_ord

# 2. plan_list_price_total
transactions_df_grouped['plan_list_price_total'] = (transactions_df_grouped['plan_list_price_total'] - transactions_df_

# 3. payment_plan_days_total
transactions_df_grouped['payment_plan_days_total'] = (transactions_df_grouped['payment_plan_days_total'] - transactions

# 4. actual_amount_paid_total
transactions_df_grouped['actual_amount_paid_total'] = (transactions_df_grouped['actual_amount_paid_total'] - transactic

# 5. actual_amount_paid_mean
transactions_df_grouped['actual_amount_paid_mean'] = (transactions_df_grouped['actual_amount_paid_mean'] - transactions

# 6. auto_renew_times
transactions_df_grouped['auto_renew_times'] = (transactions_df_grouped['auto_renew_times'] - transactions_df_grouped['a

# 7. cancel_times
transactions_df_grouped['cancel_times'] = (transactions_df_grouped['cancel_times'] - transactions_df_grouped['cancel_ti
```

6. Merge different data frames to generate tran & test data

```python
# merge with train & test data
train =pd.merge(train_df, member, how='left', on='msno')
test =pd.merge(sample_submission_df, member, how='left', on='msno')
```

```python
# merge with train & test data
train =pd.merge(train, user_logs, how='left', on='msno')
test =pd.merge(test, user_logs, how='left', on='msno')
```

```python
# merge with train & test data
train =pd.merge(train, transactions_df_grouped, how='left', on='msno')
test =pd.merge(test, transactions_df_grouped, how='left', on='msno')
```

7. Due to merging, missing values got introduced in the train & test data. Fore categorical attributes, replace with mode & for numerical attributes fill with mean.

```
# fill na values for train & test data set

# categorical columns
categorical_columns = ['registered_via_-1', 'registered_via_1', 'registered_via_2', 'registered_via_3', 'registered_via
            'registered_via_5', 'registered_via_6', 'registered_via_7', 'registered_via_8', 'registered_via_9',
            'registered_via_10', 'registered_via_11', 'registered_via_13', 'registered_via_14',
            'registered_via_16', 'registered_via_17', 'registered_via_18', 'registered_via_19',
            'city_1', 'city_3', 'city_4', 'city_5', 'city_6', 'city_7', 'city_8', 'city_9', 'city_10', 'city_11',
            'city_12', 'city_13','city_14',  'city_15', 'city_16', 'city_17', 'city_18', 'city_19','city_20',
            'city_21', 'city_22', 'low_unq_song', 'lower_middle_unq_song', 'upper_middle_unq_song', 'high_unq_song',
            'low_total_sec', 'lower_middle_total_sec', 'upper_middle_total_sec', 'high_total_sec']

for column in categorical_columns:
    train[column].fillna(train[column].mode()[0], inplace=True)
    test[column].fillna(test[column].mode()[0], inplace=True)


# numerical columns
numerical_columns = ['registration_init_time', 'total_orders', 'plan_list_price_total', 'payment_plan_days_total', 'act
                    'auto_renew_times', 'cancel_times', 'membership_expire_date_recent']

for column in numerical_columns:
    train[column].fillna(train[column].mean(), inplace=True)
    test[column].fillna(test[column].mean(), inplace=True)
```

8. **Create new feature** from datetime columns 'registration_init_time' & 'membership_expire_date_recent'. The delta of these attributes represents that how old is a user (no. of days). This information can be useful for prediction. Therefore, extract new feature & drop these columns. Normalize this newly created attribute.

```
# Extract features from date & drop datetime columns

# 1. Feature extraction

train['days'] = (train['membership_expire_date_recent'] - train['registration_init_time']).dt.days
test['days'] = (test['membership_expire_date_recent'] - test['registration_init_time']).dt.days

# 1.1 Normalize

train['days'] = (train['days'] - train['days'].min()) / (train['days'].max() - train['days'].min())
test['days'] = (test['days'] - test['days'].min()) / (test['days'].max() - test['days'].min())

# 2. Drop unnecessary columns

train = train.drop(['membership_expire_date_recent', 'registration_init_time'], 1)
test = test.drop(['membership_expire_date_recent', 'registration_init_time'], 1)
```

9. **Extract top 10 features**
    a. Calculate the correlation of each attribute with the target variable.
    b. Select top 10 attributes with max correlation (in absolute terms)

```
train.corr()['is_churn'].sort_values(ascending=False).head(11)
```

```
is_churn                    1.000000
actual_amount_paid_mean     0.354868
payment_plan_days_total     0.352293
cancel_times                0.342849
plan_list_price_total       0.328600
actual_amount_paid_total    0.322323
total_orders                0.123815
registered_via_4            0.118185
registered_via_3            0.100962
registered_via_9            0.073004
auto_renew_times            0.045047
Name: is_churn, dtype: float64
```

```
train.corr()['is_churn'].sort_values(ascending=True).head(10)
```

```
registered_via_7         -0.182445
city_1                   -0.112445
upper_middle_unq_song    -0.016976
upper_middle_total_sec   -0.015618
high_total_sec           -0.015262
high_unq_song            -0.013463
fraction_100             -0.007530
lower_middle_total_sec   -0.002121
lower_middle_unq_song    -0.001376
fraction_25              -0.000909
Name: is_churn, dtype: float64
```

c. **The top 10 features are**:
   i. actual_amount_paid_mean
   ii. payment_plan_days_total
   iii. cancel_times
   iv. plan_list_price_total
   v. actual_amount_paid_total
   vi. registered_via_7
   vii. total_orders
   viii. registered_via_4
   ix. city_1
   x. registered_via_3

Model Building-

We have implemented Logistic regression and Decision tree using the top 10 features we have engineered and to compared the performance of those two models

__For Logistic regression-__

Subsetting the data based on top features and converting all the objects into int values in order to apply Logistic regression.

```
In [274]: #subseting the datat based on top features
train_data= train[['is_churn', 'actual_amount_paid_mean', 'payment_plan_days_total', 'cancel_times', 'plan_list_price_total', 'ac
```

```
In [204]: #converting object into integar
colname=[]
for x in train_data.columns:
    if train_data[x].dtype=='object':
        colname.append(x)
colname

Out[204]: []
```

```
In [205]: colname=[]
for x in train_data.columns:
    if train_data[x].dtype=='object':
        colname.append(x)
colname
from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

for x in colname:
    train_data[x]=le.fit_transform(train[x])

    le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    print('Feature', x)
    print('mapping', le_name_mapping)
```

Post that we have **separated our independent and dependent variables(Y)**, splitted the data into test and train and apply the Logistic Regression model on data.

```
In [207]: from sklearn.model_selection import train_test_split

In [208]: #Splitting data into test and train
X_train,X_test,Y_train,Y_test= train_test_split(X,Y,test_size = 0.3,random_state=10)

In [209]: #fitting LogisticRegression model on data
from sklearn.linear_model import LogisticRegression

In [210]: # create a model
classifier= LogisticRegression()
# fitting training data to model
classifier.fit(X_train,Y_train)

Y_pred=classifier.predict(X_test)
print(list(zip(Y_test,Y_pred)))
```

Based on the Logistic Regression model, we have calculated the **confusion_matrix, accuracy_score, classification_report**

```
In [211]: from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

          cfm=confusion_matrix(Y_test,Y_pred)
          print(cfm)

          print("Classification report: ")

          print(classification_report(Y_test,Y_pred))

          #Checking train Accuracy
          acc=accuracy_score(Y_test, Y_pred)
          print("Accuracy of the model: ",acc*100)
```

```
[[262284   2684]
 [ 18234   8086]]
Classification report: 
              precision    recall  f1-score   support

           0       0.93      0.99      0.96    264968
           1       0.75      0.31      0.44     26320

    accuracy                           0.93    291288
   macro avg       0.84      0.65      0.70    291288
weighted avg       0.92      0.93      0.91    291288

Accuracy of the model:  92.81879102469034
```

Here we see that the **Accuracy of the model: 92.81879102469034**

Tuning the Model

We have further tuned our model to get the best possible Accuracy, and while doing that we see that running the threshold is between 0.4 to 0.6

```
In [276]: #Tuning the LR

In [254]: #getting the probability of Y_pred
          Y_pred_prob=classifier.predict_proba(X_test)
          print(Y_pred_prob)
```

```
[[0.9620753  0.0379247 ]
 [0.53774819 0.46225181]
 [0.97571913 0.02428087]
 ...
 [0.97571913 0.02428087]
 [0.9620753  0.0379247 ]
 [0.97571913 0.02428087]]
```

```
In [255]: #running the for loop for threshold between 0.4 to 0.6 and getting the accuracy and error corresponding to each value
          for a in np.arange(0.4,0.61,0.01):
              predict_mine = np.where(Y_pred_prob[:,1] > a, 1, 0)
              cfm=confusion_matrix(Y_test, predict_mine)
              total_err=cfm[0,1]+cfm[1,0]
              print("Errors at threshold ", a, ":",total_err, " , type 2 error :",
                    cfm[1,0]," , type 1 error:", cfm[0,1])
```

```
Errors at threshold  0.4 : 19889  , type 2 error : 15972  , type 1 error: 3917
Errors at threshold  0.41000000000000003 : 19883  , type 2 error : 15980  , type 1 error: 3903
Errors at threshold  0.42000000000000004 : 20914  , type 2 error : 17018  , type 1 error: 3896
Errors at threshold  0.43000000000000005 : 21437  , type 2 error : 17893  , type 1 error: 3544
Errors at threshold  0.44000000000000006 : 21372  , type 2 error : 17987  , type 1 error: 3385
Errors at threshold  0.45000000000000007 : 21364  , type 2 error : 18002  , type 1 error: 3362
Errors at threshold  0.4600000000000001 : 21369  , type 2 error : 18008  , type 1 error: 3361
Errors at threshold  0.4700000000000001 : 21199  , type 2 error : 18015  , type 1 error: 3184
Errors at threshold  0.4800000000000001 : 21209  , type 2 error : 18034  , type 1 error: 3175
Errors at threshold  0.4900000000000001 : 20860  , type 2 error : 18120  , type 1 error: 2740
```

We have chosen the threshold value to 0.41 based on above result

```
In [279]: #getting the accuracy based on tuned model
          cfm=confusion_matrix(Y_test,Y_pred_class)
          print(cfm)

          print("Classification report: ")

          print(classification_report(Y_test,Y_pred_class))

          acc=accuracy_score(Y_test, Y_pred_class)
          print("Accuracy of the model: ",acc*100)

          [[261065   3903]
           [ 15980  10340]]
          Classification report:
                        precision    recall  f1-score   support

                     0       0.94      0.99      0.96    264968
                     1       0.73      0.39      0.51     26320

              accuracy                           0.93    291288
             macro avg       0.83      0.69      0.74    291288
          weighted avg       0.92      0.93      0.92    291288

          Accuracy of the model:  93.17410947241218
```

Post tuning **Accuracy of the model: 93.17410947241218**

For Decision Tree-

Imported decision tree from Escalon library

```
In [285]: # Decision Tree

In [229]: #fitting DecisionTree on data
          model = tree.DecisionTreeClassifier()

In [277]: model_DecisionTree=DecisionTreeClassifier(criterion="gini",random_state=10)
```

To train our model, we have called fit here and called our input and target variable

```
In [272]: model_DecisionTree.fit(X_train,Y_train)
Out[272]: DecisionTreeClassifier(random_state=10)

In [256]: from sklearn.metrics import confusion_matrix, accuracy_score,classification_report

In [257]: Y_pred_DT=model_DecisionTree.predict(X_test)

In [258]: Y_pred_DT
Out[258]: array([0, 0, 0, ..., 0, 0, 0], dtype=int64)

In [259]: print(list(zip(Y_test,Y_pred_DT)))
          [('is_churn', 0)]
```

Now our model is ready, so we are going to predict the Accuracy score and confusion Matrix-

```
In [260]: from sklearn.metrics import confusion_matrix, accuracy_score,classification_report
```

```
In [284]: #Accuracy score and Classifiaction Report
          print(accuracy_score(Y_test,Y_pred_DT))
          print(classification_report(Y_test,Y_pred_DT))
```

```
0.9699026393122957
              precision    recall  f1-score   support

           0       0.98      0.98      0.98    264968
           1       0.84      0.82      0.83     26320

    accuracy                           0.97    291288
   macro avg       0.91      0.90      0.91    291288
weighted avg       0.97      0.97      0.97    291288
```

```
In [283]: #confusion Matrix
          print(confusion_matrix(Y_test,Y_pred_DT))
```

```
[[260972   3996]
 [  4771  21549]]
```

You can see **Accuracy score of Decision Tree is 96.99026393122957**

We are able to produce more accurate results using Decision Tree in the given problem statement.