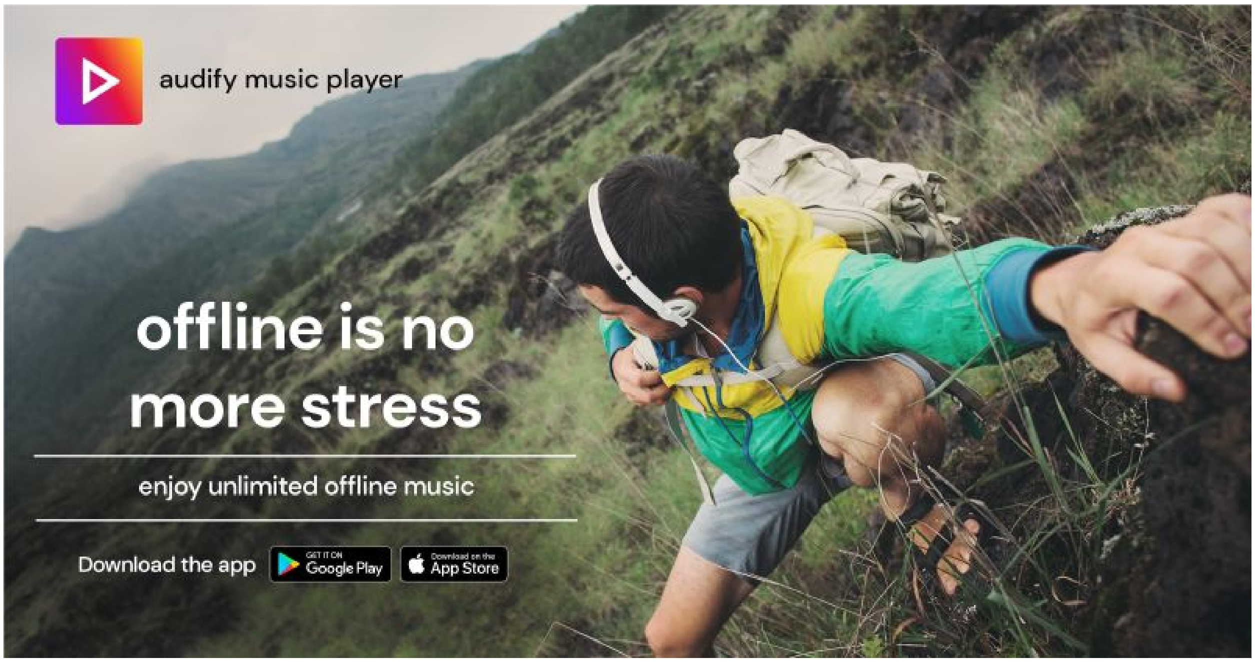
PGDBA PROJECT

HITESH MEENA

Audify App - Churn Analysis



Problem Statement

Churning in the mobile application industry is one of the major challenges. We say a customer is churned when he or she has stopped using the services or buying the products of a firm, in our case a customer is churned when the user uninstalls the Audify application.

We need to do the analysis of the data that was collected from Firebase - Google Analytics. Find out the behavior of the users who are uninstalling the app. What factors or features are affecting the churning behavior of the users and how likely (probability) users are going to uninstall the app who have not yet uninstalled.

Brief overview of solution approach

- We will be using pyspark with hadoop for loading the data as we are dealing with big data. As we have multiple entries of a single user id, we will group them by user id and sum up all the features (underlying assumptions, please find them at the bottom of the report)
- Splitting of data in terms of old users and new users and building separate models for both for better decision making and analysis.
- Feature engineering, We have 56 total event columns, all these events will be clubbed together which shows similarity
- A global test set for (users who have not yet churned) of 1 lakh users for accessing the model when run in real time.
- Train Test split of the data points for training and testing of the models that we are going to impliment.
- SMOTE with undersampling for unbalanced class problem
- Grid search for hyperparameter tuning.
- Confusion matrix and ROC curves for evaluating the accuracy of models (Logistic Regression, KNN classifier and Xgboost)

Exploratory Data Analysis

These data are for the range of date we are provided data with. 2,23,06,399 users interacted with the Audify app and 2,80,062 churned in this time frame

Total data points	2,23,06,399
Total active users	65,98,460
Total churned users	2,80,062

Initial look into the head of the data

user_pseudo_id	totalEventCount	first_activity_time	last_activity_time	sessNum	engTime_sec
2a33e972f4731248662f28c714156551	1	2022-03-13 19:18:12.015 UTC	2022-03-13 19:18:12.015 UTC	null	null
11aa7d8c808a35c1566d41d184aa5145	1	2022-03-13 21:57:56.573 UTC	2022-03-13 21:57:56.573 UTC	null	null
cb56e3a8ef28723efc95ce7812974c7e	1	2022-03-13 21:38:07.461 UTC	2022-03-13 21:38:07.461 UTC	null	null
4fa298d11f984953ed45a3aa44715d63	6	2022-03-13 19:04:10.215 UTC	2022-03-14 17:16:19.524 UTC	null	null
fbbe3e8b5d3179dc78f8b7f079fdc14e	1	2022-03-13 23:37:57.811 UTC	2022-03-13 23:37:57.811 UTC	1.0	null

bottom_option_click_event	current_search_tab_event	edit_tags_page_events	equalizer_event	feature_popup_events
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

The data is very sparse, most of the elements are zero and very few entries are greater than one. We can see Null Values also. Need to combine data of various users who are present multiple times in the rows.

We will group together all the user id and sum up all the events. This will create a feature dataframe where each row will represent one user and all its features, like his total screen view, total events, total notifications he has seen. This is based on the underlying assumption that initially we had a session of users and what he did in that session.

Here are the frequency of visits each user has made.

user_pseudo_id	count
c75a7a99f51a8f4e578943bf9bb0b2ed	1
45f3e2ee9cc17dba89227869f8c6adf5	5
f1a38b974675019f7a1f63ea8a1c075d	3
045441ffe13ac1d934659a8c9a536914	3
2280b60eb3247cf54063e3d264e10172	1
221ab279ca417be5b9fca8a258423f56	5
e2688adbe119c58b1c6ae2a4a2b08d47	5
563bef316a43e461553b99c77d002d48	1
ff2bb4945badba76c426369104ae4abd	7
bc20e5d1c01cb21551909a30ee49dc5d	3
8aa1af3f87187e53760dbc3f5144faae	7
a6d454aab932dc5bb2af3effcc56e8dd	1
cab4f8c27c935c76bebc83506b683db8	7

We have null values in sessNum and engTime columns but these will disappear when we group by and sum the features.

```
#Count the Null values in the dataframe
from pyspark.sql.functions import col,isnan,when,count
df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df_columns]
   ).show()
```

```
-+-----------------+-------+----------+---------+-------------+--------------+-------------+-
e|last_activity_time|sessNum|engTime_sec|appRemove|app_exception|app_clear_data|ad_close_event|a
-+-----------------+-------+----------+---------+-------------+--------------+-------------+-
0|                 0| 743908|    1350672|        0|            0|             0|            0|
-+-----------------+-------+----------+---------+-------------+--------------+-------------+-
```

Below is the computation of mean of all the features of users who have not churned yet and people who have churned already. As we can see, the values for people who are removed are much lower than those who are still active.

appRemove	0	1
widget_events	9.648398	0.369061
totalEventCount	357.395974	27.114028
bottom_mini_playing_bar_event	38.591762	2.133985
sessNum	8.380855	1.333212
notification_events	421.981833	17.812427
video_notification_event	0.853985	0.529083
app_clear_data	0.366268	0.081011
floating_player_event	1.616766	0.134142
interstitial_ad_events	110.447731	5.244439
lyrics_page_events	1.214717	0.178303
ad_close_event	6.520530	0.258086

New users are more likely to churn compared to old users.

This gives us the strong indication that we should segment the users by their total event count as old users and new users.

Then we can come up with two different strategies and models for analyzing their behavior. This way we will achieve user segmentation.

Why The choice of splitting on feature totalEventCount is good

We have data of a specific date range but total event count will reveal us how old is the user or how recently he has joined. Then we can segment the users based on some threshold value of this feature. The chosen threshold value will be 15 in our case and can be altered in future analysis.

Feature Engineering

We have 56 total events summed up for each unique user. Now can add up all the events that are relevant and can be described by a single feature which has the same semantic meaning as the event. **For example, events such as clicking the hamburger button, bottom option click event and all such events can be summed up can be called in_app_experience.** Few of the calculated features are given below. Total of 56 events will be scaled down to 10 features.

```
special_features = [
        'floating_player_event',
        'lyrics_open_event',
        'lyrics_page_events',
        'mini_youtube_event',
        'ringtone_cutter_event',
        'voice_assistant_event',
        'youtube_event',
        'feature_popup_events',
]
```

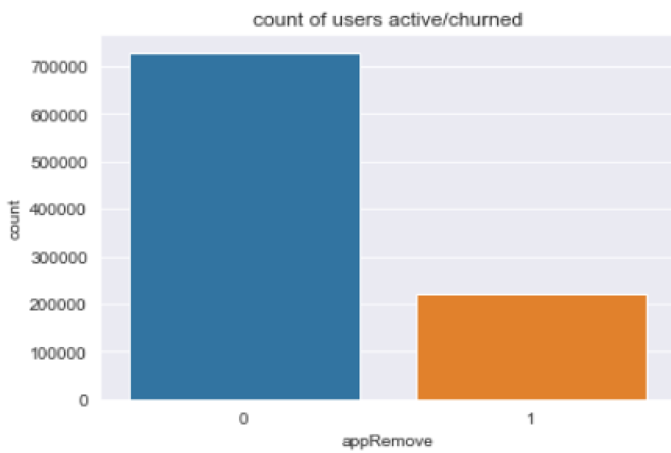
```
adsEvents = [
    'ad_close_event',
    'remove_ads_purchase_event',
    'interstitial_ad_events',
]

engagement_events =[
    'engTime_sec',
    'user_engagement',
]
```

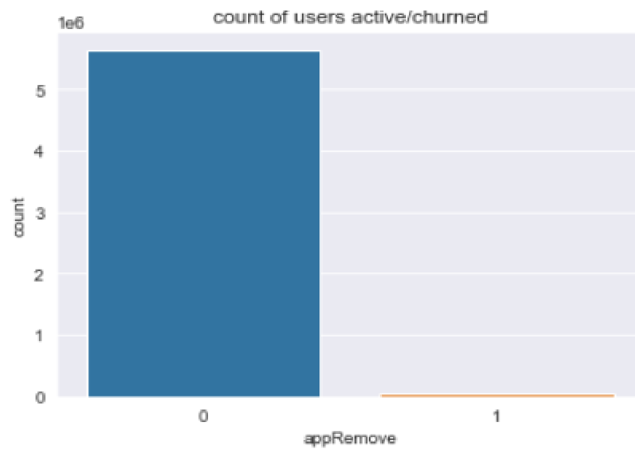
Correlation between features

	totalEventCount	sessNum	appRemove	screen_view	sum_in_app_exp	sum_special_features	sum_adsEvents
totalEventCount	1.000000	0.162095	-0.563956	0.898603	0.403640	0.098423	0.663884
sessNum	0.162095	1.000000	-0.124671	-0.002926	0.010645	0.004324	-0.025926
appRemove	-0.563956	-0.124671	1.000000	-0.535412	-0.183675	-0.072198	-0.384292
screen_view	0.898603	-0.002926	-0.535412	1.000000	0.339563	0.049980	0.601041
sum_in_app_exp	0.403640	0.010645	-0.183675	0.339563	1.000000	-0.020627	0.144499
sum_special_features	0.098423	0.004324	-0.072198	0.049980	-0.020627	1.000000	0.010709
sum_adsEvents	0.663884	-0.025926	-0.384292	0.601041	0.144499	0.010709	1.000000
sum_engagement_events	0.195790	0.027367	-0.112228	0.176404	0.090558	0.099135	0.092438
sum_notification_events	0.199844	0.009314	-0.127840	0.074602	0.164765	-0.016340	0.031242
sum_profile_visits	0.515295	-0.024046	-0.246685	0.470306	0.177150	-0.048785	0.320214

We can see we have a strong correlation of 0.89 between screen view and totalEventCount, which makes sense as more the screen view time of the user more will be the totalEventCount. We will be dropping the feature totalEventCount for the modeling purposes as it will create multicollinearity problems.

Churned new vs old users






```
df_low.appRemove.value_counts()/df_low.shape[0]
```

```
0     0.766134
1     0.233866
Name: appRemove, dtype: float64
```

```
df_high.appRemove.value_counts()/df_high.shape[0]
```

```
0     0.989785
1     0.010215
Name: appRemove, dtype: float64
```

When talking about new users 76 percent of the users have not uninstalled the app yet and 23 percent of the users have uninstalled the app. Clearly we have class unbalance.

In Old users the distribution of active users and churned users is highly unbalanced 98.9 percent of the users are still active and only 1.02 percent of the users has been churned.

This is one more indication that new users are almost 23 times more likely to to uninstall than an old user.

We have to go for sampling techniques and techniques that will handle these unbalanced class problems otherwise the model will be highly biased towards negative classes.

Outlier Detection

```
df_high.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
widget_events	5697823.0	10.685507	179.483346	0.0	0.000	0.000	0.000	73864.000
totalEventCount	5697823.0	396.733188	807.067637	15.0	58.000	160.000	421.000	369546.000
bottom_mini_playing_bar_event	5697823.0	42.840411	130.077457	0.0	0.000	8.000	40.000	42048.000
sessNum	5692037.0	9.116987	9.486014	1.0	2.000	6.000	13.000	210.000

In the old users dataframe where we have data about the users who have high total event count we can see each one has abnormally high max values. These are the outliers and need to be eliminated inter quartile range technique. Where we will eliminate only upper outliers values which are above 1.5 times above interquartile range.

```
cols = [ 'sessNum', 'screen_view', 'sum_in_app_exp', 'sum_adsEvents', 'sum_engagement_events',
       'sum_notification_events'] # one or more

Q1 = df_high[cols].quantile(0.25)
Q3 = df_high[cols].quantile(0.75)
IQR = Q3 - Q1

df_high = df_high[~( df_high[cols] > (Q3 + 1.5 * IQR)).any(axis=1)]
```

Sampling and Scaling

```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline

# define pipeline
smote = SMOTE(sampling_strategy=0.5)
under = RandomUnderSampler(sampling_strategy=0.5)
steps = [('o', smote), ('u', under)]
pipeline = Pipeline(steps=steps)
# transform the dataset
X, y = pipeline.fit_resample(X, y)
```

First we have done oversampling of positive response data points using a technique called SMOTE (Synthetic Minority Oversampling TEchnique) where we have systematically generated new data points for minority classes, this was followed by undersampling of majority classes. These two techniques combined give better results.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
fit = scaler.fit(X_train)
X_test = fit.transform(X_test)
X_train = fit.transform(X_train)
```

Since we will be dealing with algorithms which try to calculate the distance between the data points like KNN classifier and those which use gradient descent like logistic regression we have to scale the data points and then fit transform all the test points.

We are using min max scaling technique where each value in the vector will be scaled down to values between 0 and 1. This way we will make sure no single feature who just differs in domain of its values does not dominate in the modeling process.

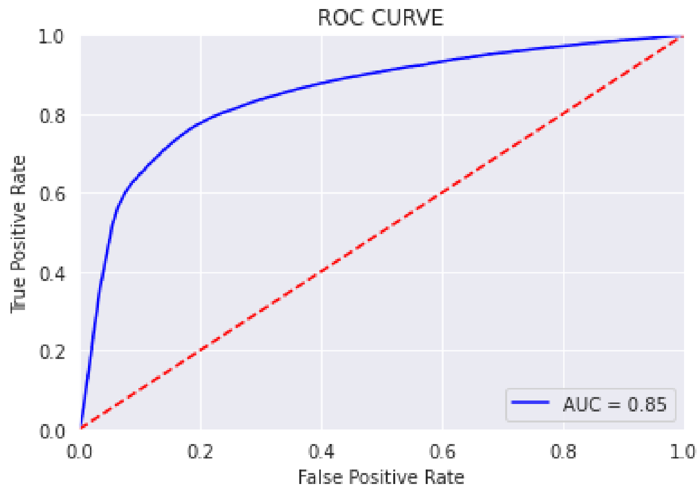
Modeling Results

So two models were trained, one for old users and the other for new users. 5 Fold cross validation technique was used for getting the appropriate test accuracies for each model.

Algorithm	5-Fold Test Accuracy	Test Accuracy	Area Under Curve(AUC)
Logistic Regression	82.04%	82%	0.85
Extreme Gradient Boosting	87.28%	87%	0.90
KNN classifier	83.42%	83%	0.86

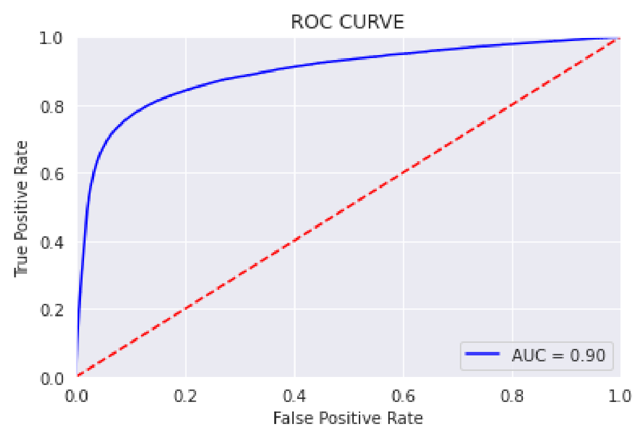
Results for Logistic Regression on test set

```
Confusion matrix :
 [[12480 11824]
 [ 3284 57154]]
Outcome values :
 57154 3284 11824 12480
Classification report :
              precision    recall  f1-score   support

           1       0.79      0.51      0.62     24304
           0       0.83      0.95      0.88     60438

    accuracy                           0.82     84742
   macro avg       0.81      0.73      0.75     84742
weighted avg       0.82      0.82      0.81     84742
```



Results for XGBoost on test set

```
Confusion matrix :
 [[16923  7381]
 [ 3444 56994]]
Outcome values :
 56994 3444 7381 16923
Classification report :
              precision    recall  f1-score   support

           1       0.83      0.70      0.76     24304
           0       0.89      0.94      0.91     60438

    accuracy                           0.87     84742
   macro avg       0.86      0.82      0.84     84742
weighted avg       0.87      0.87      0.87     84742
```



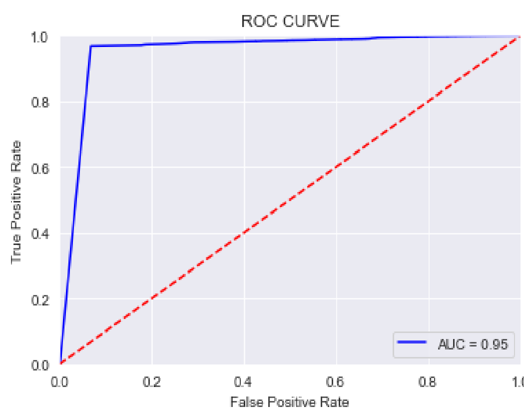
These are the results of modeling on old users.

For New users

Algorithm	5-Fold Test Accuracy	Test Accuracy	Area Under Curve(AUC)
Logistic Regression	92.4%	92%	0.95
Extreme Gradient Boosting	94.42%	96%	0.98
KNN classifier	95.42%	95%	0.96

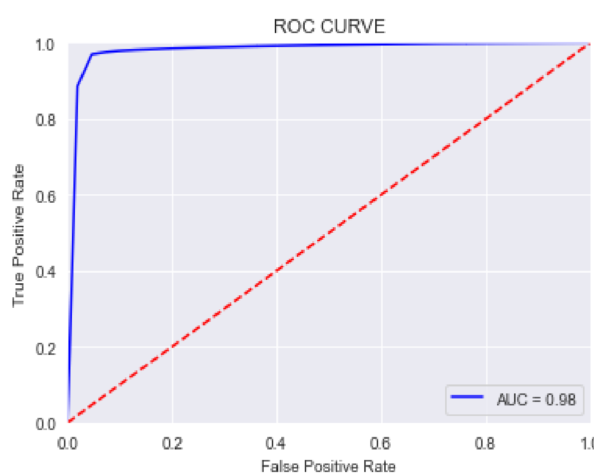
Extreme Gradient Boosting results

```
Confusion matrix :
 [[ 71101   2346]
 [ 21386 219003]]
Outcome values :
 219003 21386 2346 71101
Classification report :
              precision    recall  f1-score   support

           1       0.77      0.97      0.86     73447
           0       0.99      0.91      0.95    240389

    accuracy                           0.92    313836
   macro avg       0.88      0.94      0.90    313836
weighted avg       0.94      0.92      0.93    313836
```



Logistic regression results

```
Confusion matrix :
 [[ 65030   8417]
 [  4635 235754]]
Outcome values :
 235754 4635 8417 65030
Classification report :
              precision    recall  f1-score   support

           1       0.93      0.89      0.91     73447
           0       0.97      0.98      0.97    240389

    accuracy                           0.96    313836
   macro avg       0.95      0.93      0.94    313836
weighted avg       0.96      0.96      0.96    313836
```

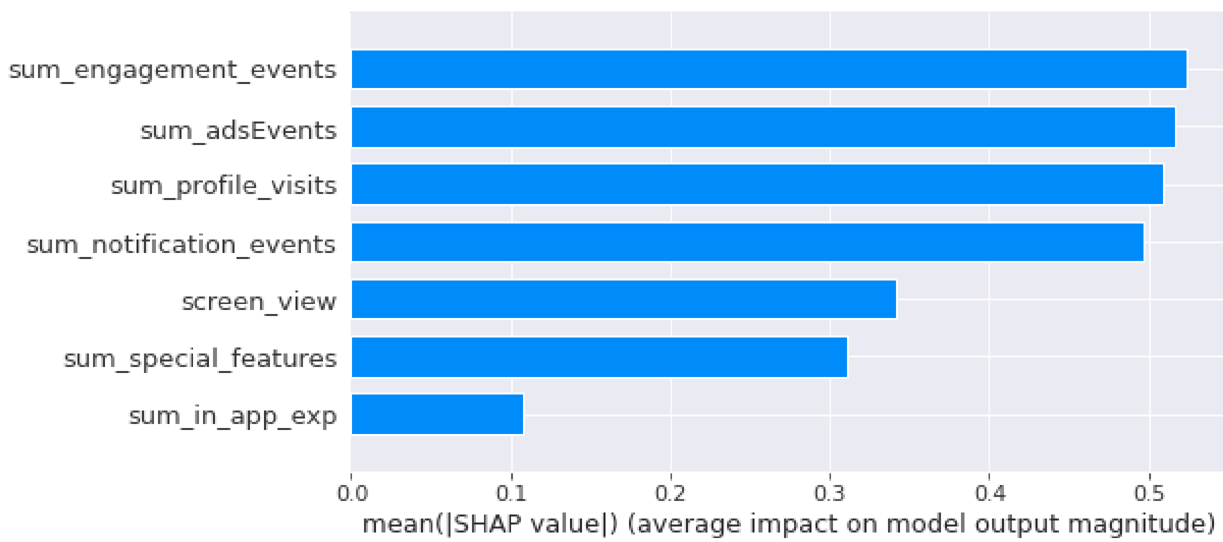


Feature importance

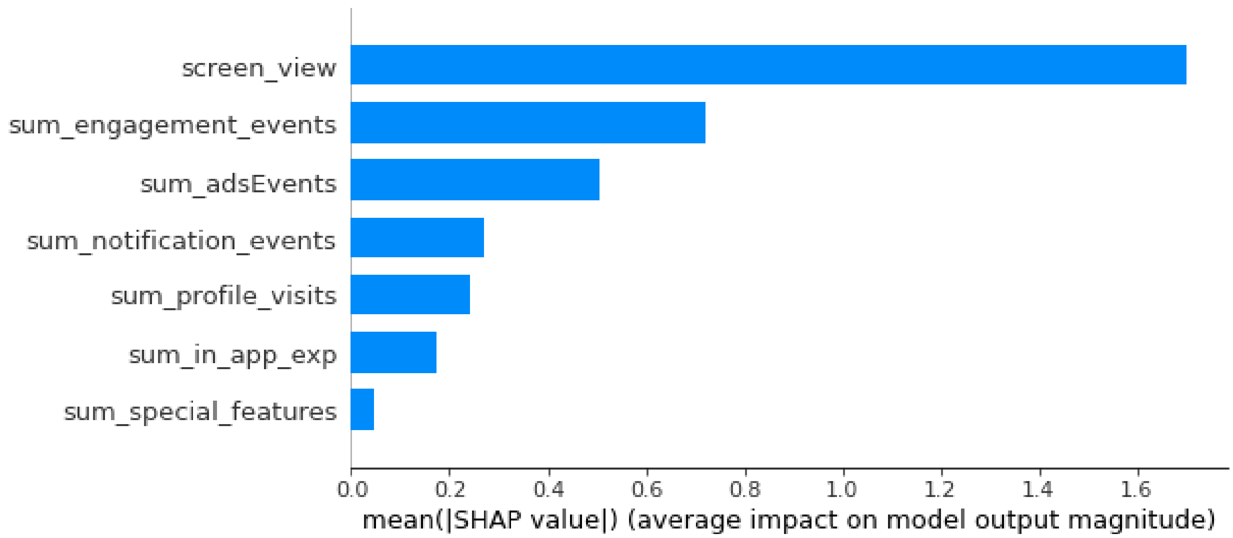
Let’s analyze the results that we got and take the necessary inference and all the outcomes we need for our problem statement. We have to compare the results for both the segments.

Most important features to consider churning of user (according to XGB boost)

Old users



New users



Findings from the feature importance results.

- **For new users most important features are screen view, engagement with the app and advertising events that they encounter. Profile visits and special features are low in feature importance which makes sense as well.**
- **The users who are quite new their churning probability is mainly determined by their engagement with the app and total screen time. How much time they spend on the app.**
- **For old users engagements events, profile visits, notification events are quite important and in app experience seems to be less important than others. Reason can be that over time they get quite used to the interface already.**

Final Output: Same rank was given to all users with same probability

New users

user_pseudo_id	churn_probability	rank
e9677fac3759a37b3d5d5e048f371b29	0.999997034	1
860364a9af68b2b8fc9eda76a9a5deda	0.999997034	1
73f28b602020fbfd7fc84189187bcd32	0.999997034	1
72e10a8d9eb99848fecbc0e65fa625fd	0.999997034	1
9b01914b58831410de4aeca92a51bbcd	0.999997034	1
0eb8629b390dcc8917866cdf10386b50	0.999997034	1
546c35d3bea676bf98b06bb0801f741c	0.999997034	1
2884b7b11f445ce24af477938e107ef0	0.999846376	4
24b5ef7ccc6069da76de2d91429b5ee8	0.999846376	4
0ae5dc97474fffa4c4b8c85661f6ad5c	0.999846376	4
ccd083c5f10be5f0d0ee80e31a91c8f1	0.999587973	6
a5be4749f0c6c8792fc5a4e260c4b7a6	0.999587973	6
ca676a87710b8a98b2bbbb2dbafe3435	0.998895408	9

For Old users

user_pseudo_id	churn_probability	rank
a37027d697cad5cb23d8e635807cb840	1	1
cc3e539b96278571a30efd9a7d7b5069	0.999982288	2
4dd5faae5dccf86ff9534c7b814ef760	0.999961668	3
336dbc16e4e254251df3693e29267b04	0.999922012	4
0a526286ceb24c68a5b49b92cc519587	0.999913654	5
de5c424019dd3c52bf2f4a6e8cb20323	0.999794159	6
383a8b70cef84f4c672e9bf4711de9ec	0.999677105	7
d4c7b25f05fd9408f0c9579cfc6875ba	0.999578826	8
5c6b624665aca8bb7b3124fb533897d3	0.999537587	9
a35a7fb6319c40ffba62ec4ca66f3858	0.999487641	10
e7b8ffeb6fb3f491f91b44434fc9dc0b	0.999440271	11
20150e78b692d62bdfd9bffc4eca6355	0.999433282	12
203467af0c4c347b278c9eceac771ce0	0.999378631	13
1d0e19e1dfbf22e8144a285795c3232b	0.999140475	14
44fee7c0b4687a832ad252620b981d95	0.998260368	15