Capstone Project Report

1. Churn prediction.

New features generated including recency features, fancier frequency feature, total listening time features.

Recency features

In [30]:
```
# defined as days from last event
# can generate one feature for each type of event

from pyspark.sql.functions import collect_list, sort_array
from pyspark.sql.functions import datediff, to_date, lit
```

In [31]:
```
#df.groupBy("uid","event").count().show()
def genarate_recency_feature(event,snapshot_date):


    df_grouped = df.filter(F.col("event") == "P").groupBy("uid").agg(F.collect_set("event").alias("event"),F.collect_list("date")
    df_sorted = df_grouped.withColumn("recent",sort_array("recent",asc = False))
    #spark.sql('set spark.sql.caseSensitive=true')
    df_last_time = df_sorted.selectExpr("uid","recent[0]")

    df_recency_feature = df_last_time.withColumn('rec_'+event,datediff(to_date(lit(snapshot_date)),"recent[0]"))
    df_recency_feature.drop(F.col('recent[0]'))

    return  df_recency_feature
```

In [32]:
```
events = ['S','D','P']
snapshot_date = "2017-05-12"
df_feature_list2 = []

for event in events:
      df_feature_list2.append(genarate_recency_feature(event,snapshot_date))
```

In [33]:
```
df_feature_list2
```

Out[33]:
```
[DataFrame[uid: string, recent[0]: date, rec_S: int],
 DataFrame[uid: string, recent[0]: date, rec_D: int],
 DataFrame[uid: string, recent[0]: date, rec_P: int]]
```

Total play time features

In [41]:
```
# generate total song play time features

df_play = spark.read.csv('data/play_ds.csv',header=True).cache()

df_play_new = df_play.selectExpr("uid","play_time","song_length","date")
```

In [42]:
```
#check missing values
df_play_new.where(F.col("play_time").isNull()).count()

df_play_new.where(F.col("song_length").isNull()).count()

# fill na with 0
df_play_nonull = df_play_new.na.fill('0')

df_play_nonull.where(F.col("uid").isNull()).count()
```

Out[42]:
```
0
```

In [43]:
```
df_play_time = df_play_nonull.selectExpr("uid","play_time","date")
```

Fancier frequency features

In [47]:
```
# generate counts of songs play 80% of their song length

df_play = spark.read.csv('data/play_ds.csv',header=True).cache()
```

In [48]:
```
df_play_date = df_play.selectExpr("uid","play_time","song_length","date").show()
```

```
+---------+-------------------+-----------+----------+
|      uid|          play_time|song_length|      date|
+---------+-------------------+-----------+----------+
|168549788|                 16|        242|2017-03-30|
|168551248|                 87|         87|2017-03-30|
|168550728|                282|        282|2017-03-30|
|168551221|                  2|        276|2017-03-30|
|168537509|                262|        262|2017-03-30|
|168548223|                  1|        179|2017-03-30|
|168551626|                247|        271|2017-03-30|
|168551397|                215|        215|2017-03-30|
|168551087|                226|        225|2017-03-30|
|168551495|                260|        260|2017-03-30|
|168550382|                251|        252|2017-03-30|
|168548223|                  1|        228|2017-03-30|
|168548945|                 58|        178|2017-03-30|
|168551016|                 14|         28|2017-03-30|
```

In [49]:
```
df_play_new = df_play.selectExpr("uid","play_time","song_length","date")
df_play_new.dtypes
```

Out[49]:
```
[('uid', 'string'),
 ('play_time', 'string'),
 ('song_length', 'string'),
 ('date', 'string')]
```

In [50]:
```
#check missing values
df_play_new.where(F.col("play_time").isNull()).count()

df_play_new.where(F.col("song_length").isNull()).count()

# fill na with 0
df_play_nonull = df_play_new.na.fill('0')

df_play_nonull.where(F.col("uid").isNull()).count()
```

Out[50]: 0

In [51]:
```
def generate_play_feature(time_window_list,snapshot_date):

        # fill na in columns

        df_play_date = df_play_nonull.filter((F.col('date')>=snapshot_date-datetime.timedelta(time_window-1)) & (F.col('date')<=s
                       selectExpr("uid","play_time","song_length","date").\
                       withColumn("play_percentage", F.col("play_time") / F.col("song_length"))
        df_play_feature = df_play_date.groupBy("uid").agg(F.count(F.col("play_percentage") > 0.8).alias("play_perc_in_last" + str

        return df_play_feature
```

In [52]:
```
time_window_list = [1,3,7,14,30]
df_feature_list4 = []
snapshot_date = feature_window_end_date

for time_window in time_window_list:
    df_feature_list4.append(generate_play_feature(time_window_list,snapshot_date))
```

In [53]:
```
df_feature_list4
```

Out[53]:
```
[DataFrame[uid: string, play_perc_in_last1: bigint],
 DataFrame[uid: string, play_perc_in_last3: bigint],
 DataFrame[uid: string, play_perc_in_last7: bigint],
 DataFrame[uid: string, play_perc_in_last14: bigint],
```

N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE
req_S_las	freq_S_las	freq_S_las	freq_S_las	rec_S	rec_D	rec_P	total_time	total_time	total_time	total_time	total_time	play_perc	play_perc_i	play_perc	play_perc	play_perc	device_type
0	0	0	0	42	42	42	0	0	0	0	37	0	0	0	0	3	2
0	0	0	0	41	41	41	0	0	0	0	611	0	0	0	0	10	2
0	0	0	0	41	41	41	0	0	0	0	2894	0	0	0	0	10	1
0	0	2	10	11	11	11	0	0	0	28024	49240	0	0	0	121	260	2
0	0	0	0	38	38	38	0	0	0	0	515	0	0	0	0	3	2
0	0	0	0	43	43	43	0	0	0	0	30	0	0	0	0	3	2
0	0	0	0	19	19	19	0	0	626	1046	3705	0	0	5	19	47	2

The results for that is:

Logistic Regression

In [17]:
```
# Import logistic regression from sklearn
from sklearn.linear_model import LogisticRegression

# Initialize model by providing parameters
clf = LogisticRegression(C=1.0, penalty='l2')
# Fit a model by providing X and y from training set
clf.fit(X_train, y_train)

# Train test model
train_test_model(clf, X_train, y_train, X_test, y_test)
```

metrics	train	test
AUC	0.861464	0.859649
Accuracy	0.726015	0.720645
Precision	0.954125	0.954995
Recall	0.587541	0.581290
f1-score	0.727249	0.722691

Random Forest

In [18]:
```
from sklearn.ensemble import RandomForestClassifier

# Choose some parameter to try
parameters = {'n_estimators': 50,
              'max_features': 'auto',
              'criterion': 'gini',
              'max_depth': 20,
              'min_samples_split': 2,
              'min_samples_leaf': 20,
              'random_state': 0,
              'n_jobs': -1
              }

clf = RandomForestClassifier(**parameters)

# Fit a model by providing X and y from training set
clf.fit(X_train, y_train)

# Train test model
train_test_model(clf, X_train, y_train, X_test, y_test)
```

metrics	train	test
AUC	0.999784	0.997910
Accuracy	0.996826	0.996998
Precision	0.995023	0.995229

Gradient Boosting Trees

In [19]:
```
from sklearn.ensemble import GradientBoostingClassifier

# Choose some parameter to try
parameters = {
    'n_estimators': 100,
    'max_depth': 5,
    'learning_rate': 0.1,
    'random_state': 42
}

# parameters = {
#     'n_estimators': 50,
#     'max_depth': 5,
#     'learning_rate': 0.2,
#     'subsample': 0.7,
#     'max_features':0.8,
#     'random_state': 42
# }

clf = GradientBoostingClassifier(**parameters)

# Train test model
train_test_model(clf, X_train, y_train, X_test, y_test)
```

```
              train      test
metrics
AUC        0.999836  0.997785
Accuracy   0.998092  0.996398
```

Neural Network

In [20]:
```
from sklearn.neural_network import MLPClassifier

# Choose some parameter combinations to try
parameters = {
    'solver':'adam',
    'activation':'relu',
    'alpha':1e-5, #increase alpha->increase penalty
    'learning_rate':'adaptive',
    'random_state':1
    }
clf = MLPClassifier(**parameters)

# Train test model
train_test_model(clf, X_train, y_train, X_test, y_test)
```

```
              train      test
metrics
AUC        0.674402  0.666905
Accuracy   0.723892  0.723647
Precision  0.695298  0.696806
Recall     0.989514  0.989043
f1-score   0.816717  0.817595
```

2. Recommendation System

Used Item-Item collaborative filtering:

print the first 10 songs for user

In [34]:
```
df_utility.columns
```

Out[34]:
```
Index(['-1', '0', '100050', '10005139', '100062', '100084', '100086', '100101',
       '10020268', '1002685',
       ...
       '996918', '99723', '99745', '99747', '998097', '99873', '998791',
       '998792', '998793', '998959'],
      dtype='object', name='song_id', length=5152)
```

In [35]:
```
print ('Top 10 songs recommended for the user:\n')
for index in unrated_index:
    for name in df_cleaned[df_cleaned['song_id']==df_utility.columns[index]]['song_name']:
        print (name)
        break
```

```
Top 10 songs recommended for the user:

风筝误
七月上
漂洋过海来看你
大王叫我来巡山-(电影《万万没想到
亲爱的姑娘 (Mix)
这个年纪
你到底有没有爱过我 ( 电影《四平青年-二龙湖浩哥之风云再起》的主题曲)
小蜜蜂
蓝眼泪{电视剧《女人汤》片尾曲}
孤独な巡礼
```

Also used matrix factorization, But always got memory error. Have not got time to fix it.

```
df_cleaned.info()
#ratings_mat.__dict__
for _, row in df_cleaned.iterrows():
    # subtract 1 from id's due to match 0 indexing
    ratings_mat[row.uid-1, row.song_id-1] = row.like
```

In [41]:
```
from sklearn.decomposition import NMF

def fit_nmf(M,k):
    nmf = NMF(n_components=k)
    nmf.fit(M)
    W = nmf.transform(M);
    H = nmf.components_;
    err = nmf.reconstruction_err_
    return W,H,err

# decompose
W,H,err = fit_nmf(ratings_mat,200)
print(err)
print(W.shape,H.shape)
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-41-8c596997ad24> in <module>()
```

Conclusion: I'm not sure I understand how to evaluate the accuracy of the recommendation system. May I get some guidance for that in your feedback? Thank you very much!