LetsGrowMore (LGM) - Virtual Internship Program

Name - Shiva Dagdu Mehenge

Task - Music Recommendation:

Music recommender systems can suggest songs to users based on their listening patterns.

Importing Packages

In [ ]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Load the Dataset

In [ ]:

```python
ntr = 7000
nts = 3000
data_path = 'C:\\Users\\Shivv"'
train = pd.read_csv(data_path + 'train.csv',nrows=ntr , encoding = 'ISO-8859-1')
names=['msno','song_id','source_system_tab','source_screen_name',\
      'source_type','target']
test1 = pd.read_csv(data_path+'train.csv',names=names,skiprows=ntr,nrows=nts)
songs = pd.read_csv(data_path + 'songs.csv')
members = pd.read_csv(data_path + 'members.csv')
```

In [ ]:

```
train.head()
```

In [ ]:

```
songs.head()
```

DATA VISUALIZATION

In [ ]:

```
sns.countplot(x = train['source_system_tab'],hue=train['source_system_tab'])
```

In [ ]:

```
sns.countplot(x = train['source_system_tab'],hue=train['target'])
```

In [ ]:

```
sns.countplot(x = train['source_screen_name'],hue=train['target'],data = train,orient='v')
plt.xticks(rotation =90)
plt.show()
```

In [ ]:

```
sns.countplot(x = train['source_type'],hue=train['source_type'],data = train,orient='v')
plt.xticks(rotation =90)
plt.show()
```

In [ ]:

```
sns.countplot(x = train['source_type'],hue=train['target'],data = train,orient='v')
plt.xticks(rotation =90)
plt.show()
```

In [ ]:

```
sns.countplot(x = songs['language'],data =train,hue=songs['language'],orient='v')
```

In [ ]:

```
sns.countplot(x = members['registered_via'],hue=members['registered_via'],orient='v')
plt.xticks(rotation =90)
plt.show()
```

DATA PREPROCESSING AND CLEANING

In [ ]:

```
test = test1.drop(['target'],axis=1)
ytr = np.array(test1['target'])
```

In [ ]:

```
test_name = ['id','msno','song_id','source_system_tab',\
             'source_screen_name','source_type']
test['id']=np.arange(nts)
test = test[test_name]
```

In [ ]:

```
song_cols = ['song_id', 'artist_name', 'genre_ids', 'song_length', 'language']
train = train.merge(songs[song_cols], on='song_id', how='left')
test = test.merge(songs[song_cols], on='song_id', how='left')
```

In [ ]:

```
members['registration_year'] = members['registration_init_time'].apply(lambda x: int(str(x)[0:4]))
members['registration_month'] = members['registration_init_time'].apply(lambda x: int(str(x)[4:6]))
members['registration_date'] = members['registration_init_time'].apply(lambda x: int(str(x)[6:8]))
```

In [ ]:

```
members = members.drop(['registration_init_time'], axis=1)
members_cols = members.columns
train = train.merge(members[members_cols], on='msno', how='left')
test = test.merge(members[members_cols], on='msno', how='left')
```

In [ ]:

```
members_cols = members.columns
train = train.merge(members[members_cols], on='msno', how='left')
test = test.merge(members[members_cols], on='msno', how='left')
```

In [ ]:

```
train = train.fillna(-1)
test = test.fillna(-1)
```

In [ ]:

```
import gc
del members, songs; gc.collect();
```

In [ ]:

```
cols = list(train.columns)
cols.remove('target')
```

MODEL BUILDING

In [ ]:

```
from tqdm import tqdm
from sklearn.preprocessing import LabelEncoder
for col in tqdm(cols):
    if train[col].dtype == 'object':
        train[col] = train[col].apply(str)
        test[col] = test[col].apply(str)

        le = LabelEncoder()
        train_vals = list(train[col].unique())
        test_vals = list(test[col].unique())
        le.fit(train_vals + test_vals)
        train[col] = le.transform(train[col])
        test[col] = le.transform(test[col])
```

TRYING OUT BASIC CLASSIFICATION MODELS

In [ ]:

```
unique_songs = range(max(train['song_id'].max(), test['song_id'].max()))
song_popularity = pd.DataFrame({'song_id': unique_songs, 'popularity':0})

train_sorted = train.sort_values('song_id')
train_sorted.reset_index(drop=True, inplace=True)
test_sorted = test.sort_values('song_id')
test_sorted.reset_index(drop=True, inplace=True)
```

In [ ]:

```
!pip install lightgbm
```

In [ ]:

```
from sklearn.model_selection import train_test_split
import lightgbm as lgb
X = np.array(train.drop(['target'], axis=1))
y = train['target'].values

X_test = np.array(test.drop(['id'], axis=1))
ids = test['id'].values

del train, test; gc.collect();

X_train, X_valid, y_train, y_valid = train_test_split(X, y, \
    test_size=0.1, random_state = 12)

del X, y; gc.collect();

d_train = lgb.Dataset(X_train, label=y_train)
d_valid = lgb.Dataset(X_valid, label=y_valid)

watchlist = [d_train, d_valid]
```

In [ ]:

```
def predict(m1_model):
    model = m1_model.fit(X_train,y_train)
    print('Training Score : {}'.format(model.score(X_train,y_train)))
    y_pred = model.predict(X_valid)
    #accuracy_score = m1_model.metrics.accuracy_score(y_valid,y_pred)
    #print('Accuracy Score : {}'.format(accuracy_score))
    v_test = model.predict(X_test)
    yhat = (v_test>0.5).astype(int)
    comp = (yhat==ytr).astype(int)
    acc = comp.sum()/comp.size*100
    print("Accuracy on test data for the model", acc)
```

In [ ]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
predict(LogisticRegression())
```

In [ ]:

```
predict(RandomForestClassifier())
```

PREDICTION USING LIGHTGBM

In [ ]:

```
params = {}
params['learning_rate'] = 0.4
params['application'] = 'binary'
params['max_depth'] = 15
params['num_leaves'] = 2**8
params['verbosity'] = 0
params['metric'] = 'auc'

model1 = lgb.train(params, train_set=d_train, num_boost_round=200, valid_sets=watchlist, \
early_stopping_rounds=10, verbose_eval=10)
```

In [ ]:

```
p_test = model1.predict(X_test)
```

In [ ]:

```
yhat = (p_test>0.5).astype(int)
comp = (yhat==ytr).astype(int)
acc = comp.sum()/comp.size*100
print('The accuracy of lgbm model on test data is: {0:f}%'.format(acc))
```

#Conclusion

This brings us to a conclusion that RandomForest and LGM perform very well in test data. But RandomForest in some cases tend to overfit the data. So LigthGBM is better at predicting the music the user needs. It provides the users 78% accuracy on global data, basically it generalizes very well