Applied Artificial Intelligence

INFO T780, Drexel University

Assignment 2

Kiana Montazeri

Content-Based Musical Genre Recommender System

Table of Contents

Introduction

I am developing a model to classify the top level genre of around 106,574 tracks that were extracted from artist's input and the information available on Free Music Archive website about each track. I am using this data as content to build a recommender system with case-based reasoning method from the results of the classifier.
I will devide the data to test and train . For the test set, I am only keeping 50 samples which are evaluated at the end.
The data was aquired from here (https://github.com/mdeff/fma), and was cleaned by me previously. All the information on cleaning can be found here (https://nbviewer.jupyter.org/github/kianamon/MusicalFeaturesPrediction/blob/master/Music.ipynb).

In [55]:
```
#Libraries in Use
from pprint import pprint
import csv
import pandas as pd
import numpy as np
%matplotlib inline
from matplotlib import pyplot as plt
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
from functools import reduce
from sklearn.preprocessing import MultiLabelBinarizer, LabelEncoder, StandardScaler
```

Loading Data

In [79]:
```
df_track = pd.read_csv("./trackinfo.csv", low_memory=False)
df_albums = pd.read_csv("./Albums.csv", low_memory=False)
df_artist = pd.read_csv("./Artist.csv", low_memory=False)
```

In [80]:
```
df_track.columns
```

Out[80]:
```
Index(['Unnamed: 0', 'trackID', 'bit_rate', 'comments', 'date_created',
       'duration', 'favorites', 'genre_top', 'genres', 'genres_all',
       'interest', 'license', 'listens', 'number', 'tags', 'title'],
      dtype='object')
```

In [81]:
```
df_albums.columns
```

Out[81]:
```
Index(['Unnamed: 0', 'trackID', 'comments', 'date_created', 'date_released',
       'favorites', 'id', 'information', 'listens', 'tags', 'title', 'tracks',
       'type'],
      dtype='object')
```

In [82]:
```
df_artist.columns
```

Out[82]:
```
Index(['Unnamed: 0', 'trackID', 'favorites', 'id', 'members', 'name', 'tags'], dtype='object')
```

We have 3 datasets about each track and we want to merge them together to use them as our content for the recommender system:

In [83]:
```
dfs = [df_track, df_albums, df_artist]
```

In [84]:
```
df_final = reduce(lambda left,right: pd.merge(left,right,on='trackID'), dfs)
```

In [85]:
```
df_final.columns
```

Out[85]:
```
Index(['Unnamed: 0_x', 'trackID', 'bit_rate', 'comments_x', 'date_created_x',
       'duration', 'favorites_x', 'genre_top', 'genres', 'genres_all',
       'interest', 'license', 'listens_x', 'number', 'tags_x', 'title_x',
       'Unnamed: 0_y', 'comments_y', 'date_created_y', 'date_released',
       'favorites_y', 'id_x', 'information', 'listens_y', 'tags_y', 'title_y',
       'tracks', 'type', 'Unnamed: 0', 'favorites', 'id_y', 'members', 'name',
       'tags'],
      dtype='object')
```

Data Pre-processing

In [86]:
```
columnstodel = ['Unnamed: 0_x', 'Unnamed: 0_y', 'Unnamed: 0']
df_final.drop(columnstodel, inplace=True, axis=1)
```

In [87]:
```
df_final[['trackID', 'bit_rate', 'comments_x', 'date_created_x',
       'duration', 'favorites_x', 'genre_top', 'genres', 'genres_all',
       'interest', 'license']].head()
```

Out[87]:

	trackID	bit_rate	comments_x	date_created_x	duration	favorites_x	genre_top	genres	genres_all	interest	license
0	2	256000	0	2008-11-26 01:48:12	168	2	Hip-Hop	[21]	[21]	4656	Attribution-NonCommercial-ShareAlike 3.0 Inter...
1	3	256000	0	2008-11-26 01:48:14	237	1	Hip-Hop	[21]	[21]	1470	Attribution-NonCommercial-ShareAlike 3.0 Inter...
2	5	256000	0	2008-11-26 01:48:20	206	6	Hip-Hop	[21]	[21]	1933	Attribution-NonCommercial-ShareAlike 3.0 Inter...
3	10	192000	0	2008-11-25 17:49:06	161	178	Pop	[10]	[10]	54881	Attribution-NonCommercial-NoDerivatives (aka M...
4	20	256000	0	2008-11-26 01:48:56	311	0	Hip-Hop	[76, 103]	[17, 10, 76, 103]	978	Attribution-NonCommercial-NoDerivatives (aka M...

In [88]:
```
df_final[['listens_x', 'number', 'tags_x', 'title_x','comments_y', 'date_created_y', 'date_released',
       'favorites_y', 'id_x', 'information', 'listens_y', 'tags_y', 'title_y',
       'tracks', 'type']].head()
```

Out[88]:

	listens_x	number	tags_x	title_x	comments_y	date_created_y	date_released	favorites_y	id_x	information	listens_y	tags_y	titl
0	1293	3	[]	Food	0	2008-11-26 01:44:45	2009-01-05 00:00:00	4	1		6073	[]	AW(A \ Of
1	514	4	[]	Electric Ave	0	2008-11-26 01:44:45	2009-01-05 00:00:00	4	1		6073	[]	AW(A \ Of
2	1151	6	[]	This World	0	2008-11-26 01:44:45	2009-01-05 00:00:00	4	1		6073	[]	AW(A \ Of
3	50135	1	[]	Freeway	0	2008-11-26 01:45:08	2008-02-06 00:00:00	4	6	No Additional Information Available	47632	[]	Cons Hitm
4	361	3	[]	Spiritual Level	0	2008-11-26 01:45:05	2009-01-06 00:00:00	2	4	"spiritual songs" from Nicky Cook	2710	[]	I

In [89]:
```
df_final[['favorites', 'id_y', 'members', 'name',
       'tags']].head()
```

Out[89]:

	favorites	id_y	members	name	tags
0	9	1	Sajje Morocco,Brownbum,ZawidaGod,Custodian of ...	AWOL	['awol']
1	9	1	Sajje Morocco,Brownbum,ZawidaGod,Custodian of ...	AWOL	['awol']
2	9	1	Sajje Morocco,Brownbum,ZawidaGod,Custodian of ...	AWOL	['awol']
3	74	6	Kurt Vile, the Violators	Kurt Vile	['philly', 'kurt vile']
4	10	4	Nicky Cook\n	Nicky Cook	['instrumentals', 'experimental pop', 'post pu...

Column `top_genre` is our our label column. We are recommending the top level genre of the music based on the content provided by the user. Now let us see how many categories we have in the genre column:

In [90]:
```
df_final['genre_top'].unique() #we have 16 diffrent classes
```

Out[90]:
```
array(['Hip-Hop', 'Pop', 'Rock', 'Experimental', 'Folk', 'Jazz',
       'Electronic', 'Spoken', 'International', 'Soul-RnB', 'Blues',
       'Country', 'Classical', 'Old-Time / Historic', 'Instrumental',
       'Easy Listening'], dtype=object)
```

In [91]:
```
len(df_final.columns)
```

Out[91]:
```
31
```

In [92]:
```
df_final.columns
```

Out[92]:
```
Index(['trackID', 'bit_rate', 'comments_x', 'date_created_x', 'duration',
       'favorites_x', 'genre_top', 'genres', 'genres_all', 'interest',
       'license', 'listens_x', 'number', 'tags_x', 'title_x', 'comments_y',
       'date_created_y', 'date_released', 'favorites_y', 'id_x', 'information',
       'listens_y', 'tags_y', 'title_y', 'tracks', 'type', 'favorites', 'id_y',
       'members', 'name', 'tags'],
      dtype='object')
```

Let us keep the most important features based on my own (Kiana's) opinion.

In [93]:
```
columnstodel2 = ['license', 'tags_x', 'date_created_y', 'date_released', 'information', 'tags_y', 'type'
                 'date_created_x', 'tags', 'members', 'id_y', 'id_x',
                  'tags_x', 'comments_x', 'tracks', 'title_x', 'title_y']
df_final.drop(columnstodel2, inplace=True, axis=1)
```

In [94]:
```
df_final.columns
```

Out[94]:
```
Index(['trackID', 'bit_rate', 'duration', 'favorites_x', 'genre_top', 'genres',
       'genres_all', 'interest', 'listens_x', 'number', 'comments_y',
       'favorites_y', 'listens_y', 'favorites', 'name'],
      dtype='object')
```

In [96]:
```
len(df_final.columns) #we have 15 features
```

Out[96]:
```
15
```

Now, we want to convert all the non-numeric data to numbers:

In [97]:
```
df_final.dtypes
```

Out[97]:
```
trackID         int64
bit_rate        int64
duration        int64
favorites_x     int64
genre_top      object
genres         object
genres_all     object
interest        int64
listens_x       int64
number          int64
comments_y      int64
favorites_y     int64
listens_y       int64
favorites       int64
name           object
dtype: object
```

In [98]:
```
# create the Labelencoder object
le = LabelEncoder()

#convert the categorical columns into numeric
df_final['genre_top'] = le.fit_transform(df_final['genre_top'])
df_final['genres'] = le.fit_transform(df_final['genres'])
df_final['genres_all'] = le.fit_transform(df_final['genres_all'])
df_final['name'] = le.fit_transform(df_final['name'])
```

In [99]:
```
df_final.dtypes
```

Out[99]:
```
trackID         int64
bit_rate        int64
duration        int64
favorites_x     int64
genre_top       int64
genres          int64
genres_all      int64
interest        int64
listens_x       int64
number          int64
comments_y      int64
favorites_y     int64
listens_y       int64
favorites       int64
name            int64
dtype: object
```

let us write the pre-processed data in a new file:

In [100]:
```
df_final.to_csv("./preprocessed.csv", sep=',')
```

Data Encoding

Now we load the data from memory:

In [182]:
```
df = pd.read_csv("./preprocessed.csv")
```

In [183]:
```
df.drop('Unnamed: 0', inplace=True, axis=1)
```

In [184]:
```
labels = df[['genre_top']]
```

In [185]:
```
df.drop('genre_top', inplace=True, axis=1)
```

In [186]:
```
df['solution'] = labels
```

In [187]:
```
df.head()
```

Out[187]:

	trackID	bit_rate	duration	favorites_x	genres	genres_all	interest	listens_x	number	comments_y	favorites_y	listens_y	favorites	na
0	2	256000	168	2	2472	1568	4656	1293	3	0	4	6073	9	
1	3	256000	237	1	2472	1568	1470	514	4	0	4	6073	9	
2	5	256000	206	6	2472	1568	1933	1151	6	0	4	6073	9	
3	10	192000	161	178	744	592	54881	50135	1	0	4	47632	74	7
4	20	256000	311	0	4533	930	978	361	3	0	2	2710	10	9

We have 14 features and one solution.

Weight Calculation

Before we start, we should set aside a few instances for testing. These would be our `new_problems` .

In [188]:
```
train = df.sample(frac=0.9995,random_state=200)
test = df.drop(train.index)
```

In [189]:
```
test.shape, train.shape # we excluded 53 samples from df for testing.
```

Out[189]:
```
((53, 15), (106521, 15))
```

Now we run the weight calculation code to see how it changes and at the end I get an average of 50 times of running the code:

In [190]:
```
from sklearn.ensemble import ExtraTreesClassifier
```

In [191]:
```
def Get_weights(df, num):
    final_weights = []
    weights = []
    for n in range(0, num):
        array = train.values
        X = array[:,0:14] # replace X with number of features
        Y = array[:,14]  # replace X with number of features
        # feature extraction
        model = ExtraTreesClassifier()
        model.fit(X, Y)
        weights.append(model.feature_importances_)
    for i in range(len(weights[0])):
        weightsum = 0
        for n in range(0, num):
            weightsum += weights[n][i]
        final_weights.append(weightsum/num)
    return final_weights
```

In [192]:
```
Weights = Get_weights(train, 50)
```

In [193]:
```
print(Weights)
```

```
[0.08959901057038833, 0.0464052302864505, 0.019661682732259107, 0.016670007271394803, 0.23317707016228
123, 0.22576038362575493, 0.025319259940333506, 0.025457220928572985, 0.025056516107460007, 0.02930710
2576391356, 0.04419029899763914, 0.07996537724926925, 0.07377670345212634, 0.06565413609967855]
```

Now let us make the dictionary of weights for the json file:

In [194]:
```
from collections import defaultdict
```

In [195]:
```
featurenames = ['F'+str(n) for n in range(1,15)]
```

In [196]:
```
print(featurenames)
```

```
['F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', 'F9', 'F10', 'F11', 'F12', 'F13', 'F14']
```

In [197]:
```
weightDict = defaultdict(float)
for i, eachF in enumerate(featurenames):
    weightDict[eachF] = Weights[i]
```

In [198]:
```
WeightsDict = dict(weightDict)
```

In [199]:
```
pprint(WeightsDict)
```

```
{'F1': 0.08959901057038833,
 'F10': 0.029307102576391356,
 'F11': 0.04419029899763914,
 'F12': 0.07996537724926925,
 'F13': 0.07377670345212634,
 'F14': 0.06565413609967855,
 'F2': 0.0464052302864505,
 'F3': 0.019661682732259107,
 'F4': 0.016670007271394803,
 'F5': 0.23317707016228123,
 'F6': 0.22576038362575493,
 'F7': 0.025319259940333506,
 'F8': 0.025457220928572985,
 'F9': 0.025056516107460007}
```

In [200]:
```
import json
json = json.dumps(WeightsDict)
f = open("weights.json","w")
f.write(json)
f.close()
```

Cases Dataset

In [202]:
```
cases_df = pd.DataFrame({'F1': train['trackID'], 'F2': train['bit_rate'],
                         'F3': train['duration'], 'F4': train['favorites_x'],
                         'F5': train['genres'], 'F6': train['genres_all'],
                         'F7': train['interest'], 'F8': train['listens_x'],
                         'F9': train['number'], 'F10': train['comments_y'],
                         'F11': train['favorites_y'], 'F12': train['listens_y'],
                         'F13': train['favorites'], 'F14': train['name'],
                         'classification': train['solution']})
```

In [203]:
```
cases_df.head()
```

Out[203]:

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	classification
60463	89872	320000	213	0	4512	704	662	259	3	0	2	10262	0	6171	13
8653	14567	320000	443	0	4767	4150	5915	3128	3	0	1	21752	0	2031	7
52955	77567	320000	278	4	1792	891	1739	649	13	0	2	20823	16	15818	4
69679	106442	320000	188	13	1891	2496	7268	5423	1	0	1	30427	1	13315	7
48317	71044	320000	372	4	1261	571	1668	881	6	0	1	21968	21	15600	7

In [204]:
```
writer = pd.ExcelWriter("cases.xlsx", engine='xlsxwriter')
cases_df.to_excel(writer, index=False)
writer.save()
```

New_Problems Dataset

In [205]:
```
newproblems_df = pd.DataFrame({'F1': test['trackID'], 'F2': test['bit_rate'],
                               'F3': test['duration'], 'F4': test['favorites_x'],
                               'F5': test['genres'], 'F6': test['genres_all'],
                               'F7': test['interest'], 'F8': test['listens_x'],
                               'F9': test['number'], 'F10': test['comments_y'],
                               'F11': test['favorites_y'], 'F12': test['listens_y'],
                               'F13': test['favorites'], 'F14': test['name']})
```

In [206]:
```
newproblems_df.head()
```

Out[206]:

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14
1440	1797	256000	89	0	1010	780	932	485	5	0	0	6319	6	5281
1818	3472	256000	137	0	795	1718	1797	367	6	0	0	5709	7	2520
5526	10422	192000	93	0	3342	2291	2513	988	1	0	1	988	4	15553
7769	13467	160000	22	0	2745	2164	988	327	10	0	0	2499	2	7372
8377	14215	320000	172	1	4684	3987	1423	627	1	0	1	5852	7	10487

In [207]:
```
writer = pd.ExcelWriter("new_problems.xlsx", engine='xlsxwriter')
newproblems_df.to_excel(writer, index=False)
writer.save()
```

Classifier

In [209]:
```
# %load run_classification.py
from pandas import read_excel
from numpy import amax
from numpy import amin
from CBR_Model_classification import CBR_Model_classification
import json

def init(data, cols, w):
    max_diff = amax(data, axis=0) - amin(data, axis=0)
    maxdiff = {col: max_diff[i] for i, col in enumerate(cols)}
    # Entering local similarity functions
    fns = {
        'F1':
            lambda x, y: 1 - abs(x-y) / maxdiff.get('F1', 1),
        'F2':
            lambda x, y: 1 - abs(x-y) / maxdiff.get('F2', 1),
        'F3':
            lambda x, y: 1 - abs(x-y) / maxdiff.get('F3', 1),
        'F4':
            lambda x, y: 1 - abs(x-y) / maxdiff.get('F4', 1),
        'F5':
            lambda x, y: 1 - abs(x-y) / maxdiff.get('F5', 1),
        'F6':
            lambda x, y: 1 - abs(x-y) / maxdiff.get('F6', 1),
        'F7':
            lambda x, y: 1 - abs(x-y) / maxdiff.get('F7', 1),
        'F8':
            lambda x, y: 1 - abs(x-y) / maxdiff.get('F8', 1),
        'F9':
            lambda x, y: 1 - abs(x-y) / maxdiff.get('F9', 1),
        'F10':
            lambda x, y: 1 - abs(x-y) / maxdiff.get('F10', 1),
        'F11':
            lambda x, y: 1 - abs(x-y) / maxdiff.get('F11', 1),
        'F12':
            lambda x, y: 1 - abs(x-y) / maxdiff.get('F12', 1),
        'F13':
            lambda x, y: 1 - abs(x-y) / maxdiff.get('F13', 1),
        'F14':
            lambda x, y: 1 - abs(x-y) / maxdiff.get('F14', 1)
    }
```

```
    return CBR_Model_classification(diff_fns=[fns.get(col, 0) for col in cols[:-1]], weights=w)

if __name__ == '__main__':
    # ---------- Hyper Parameters Start Here ----------
    json_path = 'weights.json' # Json file with feature weights
    new_path = 'new_problems.xlsx' # File that has new cases (problems) to be classified
    case_path = "cases.xlsx" # Cases in the case base

    with open(json_path) as f:
         json_data = json.load(f)

        # --- Init CBR Model ---
    cols = []
    new_case = read_excel(new_path, header=0) # New case to be classified
    old_cases = read_excel(case_path, header=0) # Cases used to determine the feature weights
    for key in new_case.keys():
        if "f" in key.lower() and key[1].isdigit():
            cols.append(key)
    # --- Data Cleaning ---
    new_data = new_case[cols].values
    cols.append('classification')
    old_data = old_cases[cols].values
    weights = list(json_data.values()) # Copies weights

    model = init(old_data, cols, weights[:])
    for case in new_data:
        print(model.predict(case, old_data, k=1))
```

```
7.0
7.0
4.0
7.0
7.0
7.0
7.0
7.0
7.0
7.0
7.0
7.0
7.0
7.0
7.0
7.0
```

```
7.0
7.0
7.0
7.0
4.0
7.0
7.0
4.0
7.0
4.0
7.0
7.0
7.0
7.0
7.0
7.0
7.0
7.0
7.0
4.0
7.0
4.0
7.0
4.0
4.0
7.0
7.0
4.0
4.0
7.0
7.0
7.0
12.0
12.0
7.0
12.0
4.0
7.0
4.0
```