EE660_Project

December 9, 2019

0.0.1 dataset_constructor.py

[ ]:
```
### Dataset creation  -  Genre Prediction Dataset ###

'''

To create the dataset for this project, a custom code is written to extract␣
↪audio from Youtube using the tool `youtube-dl`
and data collected for the 7 specific music genres from the `AudioSet` released␣
↪by Google.

source: https://arxiv.org/pdf/1804.01149.pdf

'''

import os
import re
import youtube_dl
from tqdm import tqdm
import numpy as np
import pandas as pd
from audio_utils import pre_emphasis, MFCC, Zero_crossing_rate,␣
↪Spectral_centroid, Spectral_rolloff, Chroma_feat

WAV_DIR = 'wav_files/'
genre_dict = {'/m/064t9': 'Pop_music',
                      '/m/0glt670': 'Hip_hop_music',
                      '/m/06by7': 'Rock_music',
                      '/m/06j6l': 'Rhythm_blues',
                      '/m/06cqb': 'Reggae',
                      '/m/0y4f8': 'Vocal',
                      '/m/07gxw': 'Techno'}

genre_set = set(genre_dict.keys())
temp_str = []
os.system('tar -xvf data-info.tar.gz | grep data-files')
with open('data-files/unbalanced_train_segments.csv', 'r') as f:
```

```
    temp_str = f.readlines()
data = np.ones(shape=(1,4))

print('Downloading audio files:')

for line in tqdm(temp_str):
    line = re.sub('\s?"', '', line.strip())
    elements = line.split(',')
    common_elements = list(genre_set.intersection(elements[3:]))
    if  common_elements != []:
        data = np.vstack([data, np.array(elements[:3] +␣
 ↪[genre_dict[common_elements[0]]]).reshape(1, 4)])

df = pd.DataFrame(data[1:], columns=['url', 'start_time', 'end_time',␣
 ↪'class_label'])

# Remove 10k Techno audio clips - to make the data more balanced

np.random.seed(10)
drop_indices = np.random.choice(df[df['class_label'] == 'Techno'].index,␣
 ↪size=10000, replace=False)
df.drop(labels=drop_indices, axis=0, inplace=True)
df.reset_index(drop=True, inplace=False)
df['start_time'] = df['start_time'].map(lambda x: np.int32(np.float(x)))
df['end_time'] = df['end_time'].map(lambda x: np.int32(np.float(x)))

for i, row in tqdm(df.iterrows()):
    url = "'https://www.youtube.com/embed/" + row['url'] + "'"
    file_name = str(i)+"_"+row['class_label']
    try:
        command_1 = "ffmpeg -ss "+str(row['start_time'])+" -i $(youtube-dl -f␣
 ↪140 --get-url "+url+") -t 10 -c:v copy -c:a copy "+file_name+".mp4"
        command_2 = "ffmpeg -i "+file_name+".mp4 -vn -acodec pcm_s16le -ar␣
 ↪44100 -ac 1 "+WAV_DIR+file_name+".wav"
        command_3 = 'rm '+file_name+'.mp4'
        os.system(command_1 + ';' + command_2 + ';' + command_3 + ';')
    except:
        print(i, url)
        pass

print('Download complete')
### Feature extraction and building dataset from downloaded audio files ###

cols = ['file_name'] + ['signal_mean'] + ['signal_std'] +\
       ['mfcc_' + str(i+1) + '_mean' for i in range(20)] + ['mfcc_' + str(i+1)␣
 ↪+ '_std' for i in range(20)] + \
```

```
    ␣
↪['zero_crossing_mean','zero_crossing_std','spec_centroid_mean','spec_centroid_std',␣
↪\
        'spec_rolloff_mean','spec_rolloff_std'] + \
       ['chroma_' + str(i+1) + '_mean' for i in range(12)] + ['chroma_' +␣
↪str(i+1) + '_std' for i in range(12)] +\
       ['label']
labels = {'Hip':0,'Pop':1,'Vocal':2,'Rhythm':3,'Reggae':4,'Rock':5,'Techno':6}

print('Feature extraction started')
dataset = pd.DataFrame(columns=cols)
for file in tqdm(os.listdir('wav_files')):
    signal, sample_rate = librosa.load('wav_files/'+file, sr = 22050)
    pre_emphasized_signal = pre_emphasis(signal)
    signal_mean = np.mean(abs(pre_emphasized_signal))
    signal_std = np.std(pre_emphasized_signal)
    mel_scaled_out = MFCC(pre_emphasized_signal)
    zero_crossing = Zero_crossing_rate(pre_emphasized_signal)
    spec_centroid = Spectral_centroid(pre_emphasized_signal)
    spec_rolloff = Spectral_rolloff(pre_emphasized_signal)
    chroma = Chroma_feat(pre_emphasized_signal)
    res_list = []
    res_list.append(file)
    res_list.append(signal_mean)
    res_list.append(signal_std)
    res_list.extend(np.mean(mel_scaled_out, axis = 1))
    res_list.extend(np.std(mel_scaled_out, axis = 1))
    res_list.extend((np.mean(zero_crossing), np.std(zero_crossing), np.
↪mean(spec_centroid), np.std(spec_centroid)))
    res_list.extend((np.mean(spec_rolloff), np.std(spec_rolloff)))
    res_list.extend(np.mean(chroma, axis = 1))
    res_list.extend(np.std(chroma, axis = 1))
    res_list.extend(str(labels.get(file.replace('.','_').split('_')[1])))
    dataset = dataset.append(pd.DataFrame(res_list, index = cols).T,␣
↪ignore_index = True)
dataset.to_csv("dataset_genre_pred.csv", index = False)

dataset_genre = pd.read_csv('dataset_genre_pred.csv')
dataset_genre['label'] = pd.to_numeric(dataset_genre['label'])
data_train_genre, data_sec_genre = train_test_split(dataset_genre.
↪drop('file_name', axis = 1), test_size = 0.2, random_state=5)
data_val_genre, data_test_genre = train_test_split(data_sec_genre, test_size =␣
↪0.4, random_state=5)
scaler = MinMaxScaler()
```

```
data_train_genre[data_train_genre.columns[1:len(data_train_genre.columns)-1]] =␣
 ↪scaler.fit_transform(data_train_genre[data_train_genre.columns[1:
 ↪len(data_train_genre.columns)-1]])
data_val_genre[data_val_genre.columns[1:len(data_val_genre.columns)-1]] =␣
 ↪scaler.transform(data_val_genre[data_val_genre.columns[1:len(data_val_genre.
 ↪columns)-1]])
data_test_genre[data_test_genre.columns[1:len(data_test_genre.columns)-1]] =␣
 ↪scaler.transform(data_test_genre[data_test_genre.columns[1:
 ↪len(data_test_genre.columns)-1]])
os.system('rm -rf data/')
os.system('mkdir data/')
data_train_genre.to_csv('data/data_genre_training.csv', index = False)
data_val_genre.to_csv('data/data_genre_validation.csv', index = False)
data_test_genre.to_csv('data/data_genre_test.csv', index = False)
print('Genre Dataset successfully constructed')

####### Construct dataset for Song Hit Prediction #######

import billboard
import datetime
import time
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
import warnings
warnings.filterwarnings("ignore")

### Collect Songs from Billboard for the given time range of 20 years ###

'''

To collect songs from using the Billboard API, query data for a maximum␣
 ↪duration of two years as the API stops responding
if the number of requests made by the function is too large

'''
print('Billboard audio extraction starts')
num_years = 2
for year in ["%.2d" % i for i in range(19-num_years+1, 19)]:
    prev_date_list = []
    date1 = '20'+year+'-01-01'
    date2 = '20'+str(int(year)+1)+'-11-30'
    start = datetime.datetime.strptime(date1, '%Y-%m-%d')
    end = datetime.datetime.strptime(date2, '%Y-%m-%d')
    step = datetime.timedelta(days=60)
    while start <= end:
        prev_date_list.append(str(start.date()))
        start += step
```

```
    print(prev_date_list)

    cols = ['Artist','Track','Label']
    billboard_df = pd.DataFrame()
    chart = billboard.ChartData('hot-100', prev_date_list[0])
    for i in range(1, len(prev_date_list)):
        for ind in range(1, len(chart))[:30]:
            song = chart[ind]
            if i != 1 and song.title in billboard_df[1]:
                pass
            else:
                entry = []
                entry.extend((song.artist, song.title, str(1)))
                billboard_df = billboard_df.append(pd.DataFrame(entry).T)
        chart = billboard.ChartData('hot-100', prev_date_list[i])
        time.sleep(1)
billboard_df.to_csv("billboard_data.csv", index=False)
print('Billboard audio extraction complete')
### Collect Songs which did not make it to the Billboard for the given␣
 ↪range of 20 years ###

print('Non-Billboard audio extraction starts')
billboard_df = pd.read_csv("billboard_data.csv", names=cols).iloc[1:,:]
chart = billboard.ChartData('radio-songs', prev_date_list[0])
for i in range(1, len(prev_date_list)):
    for ind in range(1, len(chart))[:30]:
        song = chart[ind]
        if i != 1 and song.title in billboard_df.iloc[:,1]:
            pass
        else:
            entry = []
            entry.extend((song.artist, song.title, str(0)))
            billboard_df = billboard_df.append(pd.DataFrame(entry, index=cols).
 ↪T)
    chart = billboard.ChartData('radio-songs', prev_date_list[i])
    time.sleep(2)
billboard_df.to_csv("billboard_data.csv", index=False)
print('Non-Billboard audio extraction complete')

### Extract Song features from Spotify and construct the dataset ###

billboard_df = pd.read_csv("billboard_data.csv")
client_credentials_manager =␣
 ↪SpotifyClientCredentials(client_id="769ef3519e8444238fde9c8981c6371c",\
                                                                        ␣
 ↪client_secret="b17e4a7ca0b4426f9962645ba5c74a63")
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
```

```
features_df = pd.DataFrame()
time_df = pd.DataFrame()
release_feat = ['Year','Month']
spotify_feat =␣
↪['Danceability','Energy','Key','Loudness','Mode','Speechiness','Acousticness','Instrumentali

for ind in range(len(billboard_df.iloc[:,0:2])):
    artist, track = billboard_df.iloc[ind,0:2]
    songs=sp.search(q='track:'+track+' '+'artist:'+artist+'*' , type='track')
    items = songs['tracks']['items']
    features_to_df = []

    if len(items) == 0:
        features_df = features_df.append(pd.Series(['None']*18), ignore_index =␣
↪True)
        time_df = time_df.append(pd.Series(['None']*2), ignore_index = True)

    else:
        track = items[0]
        song_id = str(track["id"])
        track_features=sp.audio_features(song_id)
        if int(track['album']['release_date'].split('-')[0]) < 2000:
            y = 'None'
            m = 'None'
        else:
            y = track['album']['release_date'].split('-')[0]
            m = track['album']['release_date'].split('-')[1]
        rel = [y,m]
        time_df = time_df.append(pd.DataFrame(rel).T)
        features_to_df = [val for val in (track_features)[0].values()]
        features_df = features_df.append(pd.DataFrame(features_to_df).T)

features_df = features_df.drop([11, 12, 13, 14, 15, 16, 17], axis=1)
features_df.columns = spotify_feat
time_df.columns = release_feat
output = pd.concat([billboard_df.iloc[:-1,:],features_df.iloc[:-1,:],time_df.
↪iloc[:-1,:]],axis=1)
output.to_csv("billboard_data_with_spotify.csv", index = False)
dataset = pd.read_csv('billboard_data_with_spotify.csv').
↪drop(['Artist','Track'], axis = 1)
colnames = list(dataset.columns)
dataset_no_label = dataset.drop(['Label'], axis = 1)
dataset['Label'] = pd.to_numeric(dataset['Label'])
data_train, data_sec = train_test_split(dataset, test_size = 0.1,␣
↪random_state=5)
data_val, data_test = train_test_split(data_sec, test_size = 0.5,␣
↪random_state=5)
```

```
scaler = MinMaxScaler()
data_train[data_train.columns[:2]] = scaler.fit_transform(data_train[data_train.
 ↪columns[:2]])
data_val[data_val.columns[:2]] = scaler.transform(data_val[data_val.columns[:
 ↪2]])
data_test[data_test.columns[:2]] = scaler.transform(data_test[data_test.
 ↪columns[:2]])
data_train[data_train.columns[3:4]] = scaler.
 ↪fit_transform(data_train[data_train.columns[3:4]])
data_val[data_val.columns[3:4]] = scaler.transform(data_val[data_val.columns[3:
 ↪4]])
data_test[data_test.columns[3:4]] = scaler.transform(data_test[data_test.
 ↪columns[3:4]])
data_train[data_train.columns[5:len(data_train.columns)-3]] = scaler.
 ↪fit_transform(data_train[data_train.columns[5:len(data_train.columns)-3]])
data_val[data_val.columns[5:len(data_val.columns)-3]] = scaler.
 ↪transform(data_val[data_val.columns[5:len(data_val.columns)-3]])
data_test[data_test.columns[5:len(data_test.columns)-3]] = scaler.
 ↪transform(data_test[data_test.columns[5:len(data_test.columns)-3]])
data_train.to_csv('data/data_hit_training.csv', index = False)
data_val.to_csv('data/data_hit_validation.csv', index = False)
data_test.to_csv('data/data_hit_test.csv', index = False)

### Visualizing the Datapoints - Hit Predictor ###

color = []
for i in dataset['Label']:
    if i == 1:
        color.append('blue')
    else:
        color.append('red')
# Used Pandas to plot the scatter plot of the independent variables
pd.plotting.scatter_matrix(dataset_no_label,figsize=(15,15),marker='.
 ↪',c=color,alpha=0.5,s=50)
plt.subplots_adjust(top=0.95)
plt.suptitle('Fig.1: Scatterplot of Independent Variables', fontsize=16)
# Extra commands to display legend
c0, = plt.plot([1,1],'r.')
c1, = plt.plot([1,1],'b.')
plt.legend((c0, c1),('Normal Song', 'Hit Song'),loc=(-0.5,13.1))
c0.set_visible(False)
c1.set_visible(False)
plt.savefig('results/conf_matrices/hit_pred_scatter_matrix.jpg')

print('Audio features extracted and Hit Prediction Dataset Construction␣
 ↪complete')
```

0.0.2 audio_utils.py

[ ]:
```python
import numpy as np
import librosa
from IPython.display import Audio
from librosa import display

'''

Perform Pre-emphasis

source: https://haythamfayek.com/2016/04/21/
↪speech-processing-for-machine-learning.html

'''

def pre_emphasis(signal, pre_emphasis_coeff = 0.95):  # most commonly used␣
↪values are 0.95 and 0.97
    pre_emphasis_coeff = pre_emphasis_coeff
    emphasized_signal = np.append(signal[0], signal[1:] - pre_emphasis_coeff *␣
↪signal[:-1])
    return emphasized_signal

def MFCC(emphasized_signal):
    mel = librosa.feature.mfcc(emphasized_signal)
    mel_scaled = scale(mel, axis = 1)
    return mel_scaled

def Zero_crossing_rate(emphasized_signal, eps = 0.001):     # To prevent␣
↪silence being mistaken as noise
    zero_crossing = librosa.feature.zero_crossing_rate(emphasized_signal + eps)
    zero_crossing = zero_crossing[0]
    return zero_crossing

def Spectral_centroid(emphasized_signal, eps = 0.001):
    spec_centroid = librosa.feature.spectral_centroid(emphasized_signal + eps)
    spec_centroid = spec_centroid[0]
    return spec_centroid

def Spectral_rolloff(emphasized_signal, eps = 0.001):
    spec_rolloff = librosa.feature.spectral_rolloff(emphasized_signal + eps)
    spec_rolloff = spec_rolloff[0]
    return spec_rolloff

def Chroma_feat(emphasized_signal):
    chroma = librosa.feature.chroma_stft(emphasized_signal, hop_length=1024)
    return chroma
```

0.0.3 train.py

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import cm
import seaborn as sn
import os
import re
import sys

#Pre-processing
from sklearn.preprocessing import StandardScaler, MinMaxScaler, scale

# Splitting Data into Train and Test
from sklearn.model_selection import train_test_split

# Models
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegressionCV
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.utils.multiclass import unique_labels
import xgboost as xgboost
from xgboost import XGBClassifier

# Save / Load models
import joblib

os.system('rm -rf models/')
os.system('mkdir models/')

### Load and Preprocess the data for Genre Prediction ###

labels = {'Hip':0,'Pop':1,'Vocal':2,'Rhythm':3,'Reggae':4,'Rock':5,'Techno':6}
data_train_genre = pd.read_csv('data/data_genre_training.csv')
ytrain_genre, xtrain_genre = data_train_genre['label'],␣
↪data_train_genre[data_train_genre.columns[:len(data_train_genre.columns)-1]]
print('Loaded Genre Dataset!')

### Naive Bayes Classifier - Genre ###

naive = GaussianNB().fit(xtrain_genre, ytrain_genre)
joblib.dump(naive,'models/naive_bayes_genre_trained.pkl')
```

```
### Logistic Regression Classifier - Genre ###

logregcv = LogisticRegressionCV(cv=10, multi_class='multinomial').
    ↪fit(xtrain_genre, ytrain_genre)
joblib.dump(logregcv,'models/logreg_genre_trained.pkl')

### Support Vector Machine Classifier - Genre ###

C_est = range(100, 1001, 100)
acc_test = []
for i in C_est:
    svc = SVC(C = i, probability = True, class_weight=dict(ytrain_genre.
    ↪value_counts(normalize = True)))
    svc.fit(xtrain_genre, ytrain_genre)
    svm_pred_test = svc.predict(xval_genre)
    acc_test.append(accuracy_score(yval_genre,svm_pred_test))
C_est_opt = 50 * acc_test.index(np.max(acc_test))

svc = SVC(C = C_est_opt, probability = True, class_weight=dict(ytrain_genre.
    ↪value_counts(normalize = True)))
svc.fit(xtrain_genre, ytrain_genre)
joblib.dump(svc,'models/svm_genre_trained.pkl')

### Random Forest Classifier - Genre ###

N_est = range(100, 1001, 100)
acc_test = []
for i in N_est:
    rf = RandomForestClassifier(n_estimators=i, min_samples_split=10)
    rf.fit(xtrain_genre,ytrain_genre)
    rf_pred_test = rf.predict(xval_genre)
    acc_test.append(accuracy_score(yval_genre,rf_pred_test))
N_est_opt = 100 * acc_test.index(np.max(acc_test))

rf = RandomForestClassifier(n_estimators=N_est_opt, min_samples_split=10)
rf.fit(xtrain_genre, ytrain_genre)
joblib.dump(rf,'models/random_forest_genre_trained.pkl')

### Gradient Boosting Tree Classifier - Genre ###

N_est = range(100, 1001, 100)
acc_test = []
for i in N_est:
    boost = XGBClassifier(n_estimators=i, max_depth=10, subsample=0.8,␣
    ↪num_class = len(labels), objective='multi:softprob')
    boost.fit(xtrain_genre, ytrain_genre)
```

```
    boost_pred_test = boost.predict(xval_genre)
    acc_test.append(accuracy_score(yval_genre,boost_pred_test))
N_est_opt = 100 * acc_test.index(np.max(acc_test))

boost = XGBClassifier(n_estimators=N_est_opt, max_depth=5, subsample=0.8,␣
 ↪num_class = len(labels), objective='multi:softprob')
boost.fit(xtrain_genre, ytrain_genre)
joblib.dump(boost,'models/xgboost_genre_trained.pkl')

print('Training models for Genre Prediction Complete!')

### Load and Preprocess the data for Hit Prediction ###

data_train = pd.read_csv('data/data_hit_training.csv')
ytrain, xtrain = data_train['Label'], data_train.iloc[:,:-1]
print('Loaded Billboard Hits Dataset!')

### K Nearest Neighbors Classifier - Hit Predictor ###

err_test = []
step_k = 1
k = range(1, 50, step_k)
for i in k:
    knn = KNeighborsClassifier(n_neighbors=i,p=2)
    knn.fit(xtrain, ytrain)
    predict_train = knn.predict(xtrain)
    predict_test = knn.predict(xval)
    err_test.append(np.mean(predict_test != yval))

k_opt = 1 + step_k * err_test.index(np.min(err_test))
print('Optimal K for Test data using Manhattan distance metric is', k_opt)

knn = KNeighborsClassifier(n_neighbors = k_opt, p = 1)
knn.fit(xtrain, ytrain)
joblib.dump(knn,'models/knn_hit_trained.pkl')

### Logistic Regression Classifier - Hit Predictor ###

lreg = LogisticRegressionCV(cv = 10, solver = 'liblinear', penalty =␣
 ↪'l1',refit=True)
lreg.fit(xtrain,ytrain)
joblib.dump(lreg, 'models/logreg_hit_trained.pkl')

### Support Vector Machine Classifier - Hit Predictor ###

C_est = range(1, 101, 5)
acc_test = []
```

```python
for i in C_est:
    svc = SVC(C = i, probability = True, class_weight=dict(ytrain.value_counts(normalize = True)))
    svc.fit(xtrain, ytrain)
    svc_pred_test = svc.predict(xval)
    acc_test.append(accuracy_score(yval,svc_pred_test))
C_est_opt = 5 * acc_test.index(np.max(acc_test))

svc = SVC(C = C_est_opt, probability = True, class_weight=dict(ytrain.value_counts(normalize = True)))
svc.fit(xtrain, ytrain)
joblib.dump(svc,'models/svm_hit_trained.pkl')

### Random Forest Classifier - Hit Predictor ###

N_est = range(100, 1001, 25)
acc_test = []
for i in N_est:
    rf = RandomForestClassifier(n_estimators=i, min_samples_split=10)
    rf.fit(xtrain,ytrain)
    rf_pred_test = rf.predict(xval)
    acc_test.append(accuracy_score(yval,rf_pred_test))
N_est_opt = 100 * acc_test.index(np.max(acc_test))

rf = RandomForestClassifier(n_estimators=N_est_opt, min_samples_split=10)
rf.fit(xtrain, ytrain)
joblib.dump(rf, 'models/random_forest_hit_trained.pkl')

### Gradient Boosting Trees - Hit Predictor ###

N_est = range(100, 1001, 100)
acc_test = []
for i in N_est:
    boost = XGBClassifier(n_estimators=i, max_depth=10, subsample=0.9, num_class = 2, objective='multi:softmax')
    boost.fit(xtrain, ytrain)
    boost_pred_test = boost.predict(xval)
    acc_test.append(accuracy_score(yval,boost_pred_test))
N_est_opt = 100 * (acc_test.index(np.max(acc_test))-1)

boost = XGBClassifier(n_estimators=N_est_opt, max_depth=10, subsample=0.9, num_class = 2, objective='multi:softmax')
boost.fit(xtrain, ytrain)
joblib.dump(boost, 'models/xgboost_hit_trained.pkl')

print('Training models for Hit Prediction Complete!')
```

0.0.4 validate.py

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import cm
import seaborn as sn
import os
import re
import sys
import warnings

# Evaluation Metrics
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix,␣
↪precision_score, recall_score

# Save / Load models
import joblib

### Load the data for Validating the predicted Genre ###

labels = {'Hip':0,'Pop':1,'Vocal':2,'Rhythm':3,'Reggae':4,'Rock':5,'Techno':6}
data_val_genre = pd.read_csv('data/data_genre_validation.csv')
yval_genre, xval_genre = data_val_genre['label'], data_val_genre[data_val_genre.
↪columns[:len(data_val_genre.columns)-1]]
os.system('rm -rf results/')
os.system('mkdir results/')
os.system('rm -rf results/conf_matrices/')
os.system('mkdir results/conf_matrices/')

### Naive Bayes Classifier - Genre ###

naive = joblib.load('models/naive_bayes_genre_trained.pkl')
naive_pred = naive.predict(xval_genre)
mat = confusion_matrix(yval_genre, naive_pred)
temp = pd.DataFrame(mat, index=list(labels.keys()), columns=list(labels.keys()))
fig = plt.figure()
sn.heatmap(temp, annot=True, fmt="d")
plt.xticks(rotation=45)
plt.title('Naive Bayes Model Genre: Confusion matrix')
fig.savefig('results/conf_matrices/genre_naive_bayes_cm.jpg')

### Logistic Regression Classifier - Genre ###

logregcv = joblib.load('models/logreg_genre_trained.pkl')
logreg_pred = logregcv.predict(xval_genre)
mat = confusion_matrix(yval_genre, logreg_pred)
```

```python
temp = pd.DataFrame(mat, index=list(labels.keys()), columns=list(labels.keys()))
fig = plt.figure()
sn.heatmap(temp, annot=True, fmt="d")
plt.xticks(rotation=45)
plt.title('Logistic Regression Model Genre: Confusion matrix')
fig.savefig('results/conf_matrices/genre_logreg_cm.jpg')

### Support Vector Machine Classifier - Genre ###

svc = joblib.load('models/svm_genre_trained.pkl')
svm_pred = svc.predict(xval_genre)
mat = confusion_matrix(yval_genre, svm_pred)
temp = pd.DataFrame(mat, index=list(labels.keys()), columns=list(labels.keys()))
fig = plt.figure()
sn.heatmap(temp, annot=True, fmt="d")
plt.xticks(rotation=45)
plt.title('SVM Model Genre: Confusion matrix')
fig.savefig('results/conf_matrices/genre_svm_cm.jpg')

### Random Forest Classifier - Genre ###

rf = joblib.load('models/random_forest_genre_trained.pkl')
rf_pred = rf.predict(xval_genre)
mat = confusion_matrix(yval_genre, rf_pred)
temp = pd.DataFrame(mat, index=list(labels.keys()), columns=list(labels.keys()))
fig = plt.figure()
sn.heatmap(temp, annot=True, fmt="d")
plt.xticks(rotation=45)
plt.title('Random Forest Model Genre: Confusion matrix')
fig.savefig('results/conf_matrices/genre_random_forest_cm.jpg')

### Gradient Boosting Tree Classifier - Genre ###

boost = joblib.load('models/xgboost_genre_trained.pkl')
boost_pred = boost.predict(xval_genre)
mat = confusion_matrix(yval_genre, boost_pred)
temp = pd.DataFrame(mat, index=list(labels.keys()), columns=list(labels.keys()))
fig = plt.figure()
sn.heatmap(temp, annot=True, fmt="d")
plt.xticks(rotation=45)
plt.title('Gradient Boosted Tree Model Genre: Confusion matrix')
fig.savefig('results/conf_matrices/genre_xgboost_cm.jpg')

model_accuracy = []
model_accuracy.append(round(accuracy_score(yval_genre,naive_pred),3))
model_accuracy.append(round(accuracy_score(yval_genre,logreg_pred),3))
model_accuracy.append(round(accuracy_score(yval_genre,svm_pred),3))
```

```
model_accuracy.append(round(accuracy_score(yval_genre,rf_pred),3))
model_accuracy.append(round(accuracy_score(yval_genre,boost_pred),3))

model_fscore = []
model_fscore.append(round(f1_score(yval_genre,naive_pred,average='weighted'),3))
model_fscore.append(round(f1_score(yval_genre,logreg_pred,average =␣
↪'weighted'),3))
model_fscore.append(round(f1_score(yval_genre,svm_pred,average = 'weighted'),3))
model_fscore.append(round(f1_score(yval_genre,rf_pred,average = 'weighted'),3))
model_fscore.append(round(f1_score(yval_genre,boost_pred,average =␣
↪'weighted'),3))

res_list_genre = []
res_list_genre.append(model_accuracy)
res_list_genre.append(model_fscore)

eval_metrics_genre = ['Accuracy','F-Score']
models_to_test_genre = ['Naive Bayes','Logistic Regression','Support Vector␣
↪Machine','Random Forest','XGBoost']
metrics_genre = pd.DataFrame(res_list_genre, columns=models_to_test_genre,␣
↪index=eval_metrics_genre)
metrics_genre.index.name = 'Metric'
metrics_genre.to_csv('results/eval_metrics_training.csv')

### Load the data for validating Hit Prediction ###

data_val = pd.read_csv('data/data_hit_validation.csv')
yval, xval = data_val['Label'], data_val.iloc[:,:-1]

### K Nearest Neighbors Classifier - Hit Predictor ###

knn = joblib.load('models/knn_hit_trained.pkl')
predict_test = knn.predict(xval)
predict_test = predict_test[:,np.newaxis]
mat = confusion_matrix(yval, predict_test)
temp = pd.DataFrame(mat, index=['Hit Song','Normal'], columns=['Hit␣
↪Song','Normal'])
fig = plt.figure()
sn.heatmap(temp, annot=True, fmt="d")
plt.title('Confusion Matrix')
fig.savefig('results/conf_matrices/hit_knn_cm.jpg')

### Logistic Regression Classifier - Hit Predictor ###

lreg = joblib.load('models/logreg_hit_trained.pkl')
pred_lreg = lreg.predict(xval)
mat = confusion_matrix(yval, pred_lreg)
```

```
temp = pd.DataFrame(mat, index=['Hit Song','Normal'], columns=['Hit␣
↪Song','Normal'])
fig = plt.figure()
sn.heatmap(temp, annot=True, fmt="d")
plt.title('Confusion Matrix')
fig.savefig('results/conf_matrices/hit_logreg_cm.jpg')

### Support Vector Machine Classifier - Hit Predictor ###

svc = joblib.load('models/svm_hit_trained.pkl')
svm_pred = svc.predict(xval)
mat = confusion_matrix(yval, svm_pred)
temp = pd.DataFrame(mat, index=['Hit Song','Normal'], columns=['Hit␣
↪Song','Normal'])
fig = plt.figure()
sn.heatmap(temp, annot=True, fmt="d")
plt.title('Confusion Matrix')
fig.savefig('results/conf_matrices/hit_svm_cm.jpg')

### Random Forest Classifier - Hit Predictor ###

rf = joblib.load('models/random_forest_hit_trained.pkl')
rf_pred = rf.predict(xval)
mat = confusion_matrix(yval, rf_pred)
temp = pd.DataFrame(mat, index=['Hit Song','Normal'], columns=['Hit␣
↪Song','Normal'])
fig = plt.figure()
sn.heatmap(temp, annot=True, fmt="d")
plt.title('Confusion Matrix')
fig.savefig('results/conf_matrices/hit_random_forest_cm.jpg')

### Gradient Boosting Trees - Hit Predictor ###

boost = joblib.load('models/xgboost_hit_trained.pkl')
boost_pred = boost.predict(xval)
mat = confusion_matrix(yval, boost_pred)
temp = pd.DataFrame(mat, index=['Hit Song','Normal'], columns=['Hit␣
↪Song','Normal'])
fig = plt.figure()
sn.heatmap(temp, annot=True, fmt="d")
plt.title('Confusion Matrix')
fig.savefig('results/conf_matrices/hit_xgboost_cm.jpg')


model_accuracy = []
model_accuracy.append(round(accuracy_score(yval,predict_test),3))
model_accuracy.append(round(accuracy_score(yval,pred_lreg),3))
```

```
model_accuracy.append(round(accuracy_score(yval,svm_pred),3))
model_accuracy.append(round(accuracy_score(yval,rf_pred),3))
model_accuracy.append(round(accuracy_score(yval,boost_pred),3))

model_fscore = []
model_fscore.append(round(f1_score(yval,predict_test,average = None)[1],3))
model_fscore.append(round(f1_score(yval,pred_lreg,average = None)[1],3))
model_fscore.append(round(f1_score(yval,svm_pred,average = None)[1],3))
model_fscore.append(round(f1_score(yval,rf_pred,average = None)[1],3))
model_fscore.append(round(f1_score(yval,boost_pred,average = None)[1],3))

model_precision = []
model_precision.append(round(precision_score(yval,predict_test,average =␣
 ↪None)[1],3))
model_precision.append(round(precision_score(yval,pred_lreg,average =␣
 ↪None)[1],3))
model_precision.append(round(precision_score(yval,svm_pred,average =␣
 ↪None)[1],3))
model_precision.append(round(precision_score(yval,rf_pred,average = None)[1],3))
model_precision.append(round(precision_score(yval,boost_pred,average =␣
 ↪None)[1],3))

model_recall = []
model_recall.append(round(recall_score(yval,predict_test,average = None)[1],3))
model_recall.append(round(recall_score(yval,pred_lreg,average = None)[1],3))
model_recall.append(round(recall_score(yval,svm_pred,average = None)[1],3))
model_recall.append(round(recall_score(yval,rf_pred,average = None)[1],3))
model_recall.append(round(recall_score(yval,boost_pred,average = None)[1],3))

res_list = []
res_list.append(model_accuracy)
res_list.append(model_fscore)
res_list.append(model_precision)
res_list.append(model_recall)

eval_metrics_hit = ['Accuracy','F-Score','Precision','Recall']
models_to_test_hit = ['K-Nearest Neighbors','Logistic Regression','Support␣
 ↪Vector Machine','Random Forest','XGBoost']
metrics_hit = pd.DataFrame(res_list, columns=models_to_test_hit,␣
 ↪index=eval_metrics_hit)
metrics_hit.index.name = 'Metric'
emptydf = pd.DataFrame()
emptydf.to_csv('results/eval_metrics_training.csv', mode='a')
metrics_hit.to_csv('results/eval_metrics_training.csv', mode='a')

print('\nEvaluation Metrics for Genre and Hit Prediction respectively:\n')
```

```
os.system('column -t -s "," results/eval_metrics_training.csv')
print('\nModel Selection:\n\nFrom the above results, we can see that the␣
↪Gradient Boosted Tree model works better for predicting \nthe Genre and the␣
↪Random Forest model works best to predict if a given song will make it to␣
↪the billboard or not. \nFurthermore, we want to see how these models perform␣
↪on new data.\n')
```

0.0.5 test.py

```
[ ]: import numpy as np
import pandas as pd
import os
import joblib

# Evaluation Metrics
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
↪recall_score

### Load the Test set to see the performance of the model in predicting the␣
↪correct Genre ###

data_test_genre = pd.read_csv('data/data_genre_test.csv')
ytest_genre, xtest_genre = data_test_genre['label'],␣
↪data_test_genre[data_test_genre.columns[:len(data_test_genre.columns)-1]]

best_model_genre = joblib.load('models/xgboost_genre_trained.pkl')
pred_best_model_genre = best_model_genre.predict(xtest_genre)
test_list_genre = []
eval_metrics_genre = ['Accuracy','F-Score']
test_list_genre.
↪append(round(accuracy_score(ytest_genre,pred_best_model_genre),3))
test_list_genre.
↪append(round(f1_score(ytest_genre,pred_best_model_genre,average='weighted'),3))
metrics_genre = pd.DataFrame(test_list_genre, columns=['XGBoost'],␣
↪index=eval_metrics_genre)
metrics_genre.index.name = 'Metric'
metrics_genre.to_csv('results/eval_metrics_test.csv')

### Load the Test set to see the performance of the model in the Hit prediction␣
↪of a Song ###

data_test = pd.read_csv('data/data_hit_test.csv')
ytest, xtest = data_test['Label'], data_test.iloc[:,:-1]

best_model_hit = joblib.load('models/random_forest_hit_trained.pkl')
```

```python
pred_best_model_hit = best_model_hit.predict(xtest)
test_list_hit = []
test_list_hit.append(round(accuracy_score(ytest,pred_best_model_hit),3))
test_list_hit.
 ↪append(round(f1_score(ytest,pred_best_model_hit,average='weighted'),3))
test_list_hit.append(round(precision_score(ytest,pred_best_model_hit,average =␣
 ↪None)[1],3))
test_list_hit.append(round(recall_score(ytest,pred_best_model_hit,average =␣
 ↪None)[1],3))
eval_metrics_hit = ['Accuracy','F-Score','Precision','Recall']
metrics_hit = pd.DataFrame(test_list_hit, columns=['Random Forest'],␣
 ↪index=eval_metrics_hit)
metrics_hit.index.name = 'Metric'
emptydf = pd.DataFrame()
emptydf.to_csv('results/eval_metrics_test.csv', mode='a')
metrics_hit.to_csv('results/eval_metrics_test.csv', mode='a')

print('\nEvaluation Metrics for Genre and Hit Prediction on Test data␣
 ↪respectively:\n')
os.system('column -t -s "," results/eval_metrics_test.csv')
```

0.0.6 main.py

[ ]:
```python
import os
import re
import sys
import argparse
import warnings
warnings.filterwarnings("ignore")

def str2bool(v):
    if isinstance(v, bool):
        return v
    if v.lower() in ('yes', 'true', 't', 'y', '1'):
        return True
    elif v.lower() in ('no', 'false', 'f', 'n', '0'):
        return False
    else:
        raise argparse.ArgumentTypeError('Boolean value expected.')

parser = argparse.ArgumentParser()
parser.add_argument("-c", "--construct",type=str2bool, nargs='?', help="Extract␣
 ↪data from sources and construct the dataset")
parser.add_argument("-t", "--train",type=str2bool, nargs='?', help="Train the␣
 ↪models to find the best performing one and test it on the Test data")
```

```python
args = parser.parse_args()
if args.construct:
    dataset_creation = True
else:
    dataset_creation = False
if args.train:
    do_train = True
else:
    do_train = False

if dataset_creation == True:
    print('Dataset Construction started')
    os.system('python3 dataset_constructor.py')
    print('Datasets successfully constructed')

if do_train == True:
    print('Training started')
    os.system('python3 train.py')

os.system('python3 validate.py')

os.system('python3 test.py')

'''

Datasets are stored in data/
Models are stored in the models/ directory
Intermediate training results are stored in conf_matrices/

'''
```