PROCEDURE FOR RUNNING CODE

Softwares used:

1. **Anaconda**
2. **Anaconda Spyder**

Code:

1. Creating Dataset:

In order to create the dataset for this experiment, download “genres.list” and “plot.list” files from a mirror FTP and then parse files in order to associate the titles, plots and genres.

The following code will parse both files in order to generate a single file “movies_genres.csv” containing the plot and the genres associated to each movie.

```
from collections import defaultdict

import codecs

import pandas as pd


def get_genres():

    genres = defaultdict(list)

    unique_genres = set()

    with codecs.open("genres.list", "r", encoding=None, errors="ignore") as f:

        for line in f:

            if line.startswith('"'):

                data = line.split('\t')

                movie = data[0]
            genre = data[-1].strip()
            genres[movie].append(genre)
            unique_genres.add(genre)
    return genres,sorted(unique_genres)
def get_plots():
    with codecs.open("plot.list", "r",encoding=None, errors="ignore") as f:
        data = []
        inside = False
        plot = ' '
        full_title = ' '
        for line in f:
            if line.startswith("MV:") and not inside:
                inside = True
                full_title = line.split("MV:")[1].strip()
            elif line.startswith("PL:") and inside:
                plot += line.split("PL:")[1].replace("\n", "")
            elif line.startswith("MV:") and inside:
                short_title = full_title.split('{')[0].strip()
                data.append((short_title, full_title, plot))
                plot = ''
                inside = False
    return data
```

```
def main():
    # for TV shows plots contain names and episodes, e.g.:
    #   "#LawstinWoods" (2013) {The Case of the Case (#1.5)}
    # genres contain only main title, e.g.
    #   #LawstinWoods"  (2013) Sci-Fi
    genres,unique_genres = get_genres()
    data = []
    for movie in genres:
        row = [0]*len(unique_genres)
        for g in genres[movie]:
            row[unique_genres.index(g)] = 1
        row.insert(0, movie)
        data.append(row)
    genres_df = pd.DataFrame(data)
    genres_df.columns = ['short_title'] + unique_genres
    print(genres_df.shape)
    plots = get_plots()
    plots_df = pd.DataFrame(plots)
    plots_df.columns = ['short_title', 'title', 'plot']
    print(plots_df.shape)
    data_df = plots_df.merge(genres_df, how='inner', on='short_title')
    data_df.dropna(inplace=True)
    data_df.drop('short_title', axis=1, inplace=True)
    # 'Sci-fi' is not associated with any plot
    data_df.drop('Sci-fi', axis=1, inplace=True)
    print(data_df.shape)
    data_df.to_csv(path_or_buf='movies_genres.csv', sep='\t',
                   header=True, encoding='utf8', index=False)
if __name__ == "__main__":
    main()
```

2. Preprocessing:

The dataset “movie_genres.csv” thus obtained from the previous step will be preprocessed by doing some exploratory analysis. Calculate the absolute number of movies per genre. And then remove the genre which have zero number of instances i.e., “Lifestyle”. One thing that notice when working with this dataset is that there are plots written in different languages. So use “langdetect” package to identify the languages and remove the languages that are other than English. Then create another file with this preprocessed data.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("movies_genres.csv", delimiter='\t')
df.info()
df_genres = df.drop(['plot', 'title'], axis=1)
counts = []
```

```
categories = list(df_genres.columns.values)

for i in categories:

    counts.append((i, df_genres[i].sum()))

df_stats = pd.DataFrame(counts, columns=['genre', '#movies'])

df_stats.plot(x='genre', y='#movies', kind='bar', legend=False, grid=True, figsize=(15, 8))

df.drop('Lifestyle', axis=1, inplace=True)

df.drop('Lifestyle', axis=1, inplace=True)

from langdetect import detect

df['plot_lang'] = df.apply(lambda row: detect(row['plot'].decode("utf8")), axis=1)

df['plot_lang'].value_counts()

df = df[df.plot_lang.isin(['en'])]

df.to_csv("movies_genres_en.csv", sep='\t', encoding='utf-8', index=False)

data = pd.read_csv("movies_genres_en.csv", delimiter="\t")

def get_data(row):

    for c in data.columns:

        if row[c]==1:

            return c

data_genres = data.drop(['plot', 'title'], axis=1)

genre_list = list(data.apply(get_data, axis=1))

categories_list = list(data_genres.columns.values)

categories_list.append('plot_lang')

dframe1 = data.drop(categories_list ,axis=1)

title_list = dframe1[['title']].values.tolist()
```

```
plot_list = dframe1[['plot']].values.tolist()

dframe = pd.DataFrame()

dframe['title']=title_list

dframe['plots']=plot_list

dframe['genres']=genre_list

dframe.to_csv(path_or_buf='movie_genre_new.csv', sep='\t',header=True, encoding='utf8', index=False)
```

3. Vector Representation

To train supervised classifiers, we first need to transform the plot into a vector of numbers. Term Frequency – Inverse Document Frequency is used to convert the plots into vector representation.

Dataset is split into 80% - 20% for Training and Testing respectively by using train_test_split class. TfidfVectorizer package has been used from sklearn.feature_extraction package.

```
from sklearn import model_selection, preprocessing

from sklearn.feature_extraction.text import TfidfVectorizer

import numpy

import pandas as pd

data_df = pd.read_csv("movie_genre_new.csv", delimiter='\t')

data_x = data_df[['plots']].as_matrix()

data_y = data_df[['genres']].as_matrix()

x_train, x_test, y_train, y_test = model_selection.train_test_split(data_x,data_y,test_size=0.2, random_state=42)

encoder = preprocessing.LabelEncoder()

train_y = encoder.fit_transform(y_train)
```

```
test_y = encoder.fit_transform(y_test)

train_x = [x[0].strip() for x in x_train.tolist()]

test_x = [x[0].strip() for x in x_test.tolist()]


# Word – Level TF-IDF Vectorizer

tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}', max_features=5000)

xtrain_tfidf =  tfidf_vect.fit_transform(train_x)

xtest_tfidf =  tfidf_vect.transform(test_x)


#N-grams Level TF-IDF Vectorizer

tfidf_vect_ngram=TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}', ngram_range=(2,3),
max_features=5000)

xtrain_tfidf_ngram =  tfidf_vect_ngram.fit_transform(train_x)

xtest_tfidf_ngram =  tfidf_vect_ngram.transform(test_x)
```

4. Classification and Metrics

For Naïve Bayes classifier import “naïve_bayes” class from “sklearn” package. For XGBoost classifier import “xgboost” package. The ‘fit( )’ method will make the machine to learn based on the training dataset. Then Metrics like Jackard index, Accuracy score and F1 score will be calculated using “metrics” class from “sklearn” package.

.

```
from sklearn import model_selection, preprocessing, naive_bayes, metrics

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

import xgboost, numpy
import pandas as pd

def train_model(classifier , feature_vector_train, label , feature_vector_valid):

    classifier.fit(feature_vector_train, label)

    predictions = classifier.predict(feature_vector_valid)

    ac = metrics.accuracy_score(predictions ,test_y)

    jc = metrics.jaccard_similarity_score(predictions, test_y)

    f = metrics.f1_score(test_y, predictions, average="weighted")

    return [ac,jc,f]

accuracy = train_model(naive_bayes.MultinomialNB(), xtrain_tfidf , train_y , xtest_tfidf)

accuracy2=train_model(xgboost.XGBClassifier(), xtrain_tfidf.tocsc(), train_y, xtest_tfidf.tocsc())

accuracy3=train_model(naive_bayes.MultinomialNB(),xtrain_tfidf_ngram,train_y,
xtest_tfidf_ngram)
```