Implementation code

# Data collection from website songlyrics.com

from bs4 import BeautifulSoup

from urllib.request import urlopen

import urllib

import pandas as pd

import numpy as np

import csv

import requests

from requests.exceptions import HTTPError

lyricsvector = [] #input (bag of words)

genrevector = [] #target

songinfovector = [] #metadata (artist and songname)

# List the URLs here

urllist = [

#"http://www.songlyrics.com/news/top-genres/christian/",

#"http://www.songlyrics.com/news/top-genres/country-music/",

"http://www.songlyrics.com/news/top-genres/hip-hop-rap/",

"http://www.songlyrics.com/news/top-genres/pop/",

"http://www.songlyrics.com/news/top-genres/rock/",

"http://www.songlyrics.com/news/top-genres/rhythm-blues/"]

for i in range(0,6):

req = urllib.request.Request(urllist[i], headers={'User-Agent' : "Magic Browser"})

url = urllib.request.urlopen(req)

doc = url.read()

soup = BeautifulSoup(doc, 'html.parser')

div = soup.find( 'div', { 'class': 'box listbox' } )

#print(div)

# get genres

title = soup.title.get\_text().encode('ascii', 'ignore').decode().split(' ')

index100 = title.index('100')

indexSongs = title.index('Songs')

genre = ' '.join(title[(index100+1):(indexSongs)]).encode('utf-8')

# create list of top 100 songs by genre

print(genre)

songs = div.find\_all('a')

songlinks = []

# create loop to extract song links

for j in range(0,200): #[0::2]:

songlink = songs[j].get('href').encode('ascii', 'ignore')

songlinks.append(songlink) #output links to a list called songlinks

songlinks = filter(None, songlinks)

songlinks = [songlink for songlink in songlinks if (len(songlink.decode().split('/'))==6)]

count=0

for k in range(0,len(songlinks)):

try:

r = requests.get(songlinks[k].decode())

r.raise\_for\_status()

except HTTPError:

print(songlinks[k].decode()+"- Error while retreiving page")

else:

req = urllib.request.Request(songlinks[k].decode(), headers={'User-Agent' : "Magic Browser"})

songurl = urllib.request.urlopen(req)

songdoc = songurl.read()

songsoup = BeautifulSoup(songdoc, 'html.parser')

songinfo = songsoup.title.get\_text().encode('ascii', 'ignore')

print(songinfo, 'is number', count)

count=count+1

songdiv = songsoup.find( 'div', { 'id': 'songLyricsDiv-outer' } )

try:

lyrics = songdiv.getText().replace("\n", " ").replace("\'", "").replace("\r", " ").encode('utf-8')

lyricsvector.append(lyrics.decode())

genrevector.append(genre.decode())

songinfovector.append(songinfo.decode())

except:

print("Error")

df = pd.DataFrame({ "SongInfo" : songinfovector, "Lyrics" : lyricsvector ,"Genre" : genrevector })

df.to\_csv("dataset.csv", index=False)

# Data collection from million song dataset

import pandas as pd

df = pd.read\_csv("msd\_genre\_dataset.csv", encoding = "ISO-8859-1")

df

'''

df = df.filter(['track\_id','artist\_name','title','genre'], axis=1)

df

'''

df['lyrics'] = pd.Series('', index=df.index)

df.tail()

df= df.drop\_duplicates(subset=['artist\_name','title'])

df

from bs4 import BeautifulSoup

from urllib.request import urlopen

import urllib

import pandas as pd

import numpy as np

import csv

import requests

from requests.exceptions import HTTPError

def remove\_ascii(text):

return ''.join([i if ord(i) < 128 else ' ' for i in text])

def replace\_with\_newlines(element):

text = ''

for elem in element.recursiveChildGenerator():

if isinstance(elem, str):

text += elem.strip()

elif elem.name == 'br':

text += '\n'

return text

def check\_url(link):

try:

r = requests.get(link)

r.raise\_for\_status()

except HTTPError:

print("- Error while retreiving page")

return False

else:

return True

def getDiv(link,soup):

req = urllib.request.Request(link, headers={'User-Agent' : "Magic Browser"})

url = urllib.request.urlopen(link)

doc = url.read()

soup = BeautifulSoup(doc, 'lxml')

div = soup.find( 'div', { 'class': 'lyricbox' } )

return div

def getlyrics(artist,title):

artist = artist.replace(" ","\_")

artist = remove\_ascii(artist)

title = title.replace(" ","\_")

title = remove\_ascii(title)

link = "http://lyrics.wikia.com/wiki/" + artist.title() + ":" + title.title()

print(link)

lyrics = ""

soup = None

global soup

if check\_url(link) is True:

div = getDiv(link,soup)

if div is None:

try:

redirect = soup.find( 'div', { 'class': 'redirectMsg' } )

link = redirect.find\_all('a')

link = link.get('href').encode('ascii', 'ignore')

if check\_url(link) is True:

div = getDiv(link,soup)

else:

return lyrics

except:

print("Error- no redirect")

return lyrics

lyrics = replace\_with\_newlines(div)

return lyrics

df

from tqdm import tqdm

df = df.reset\_index(drop=True)

n = len(df.index)

for row\_id in tqdm(range(25000,30000)):

lyrics = getlyrics(df.loc[row\_id]['artist\_name'], df.loc[row\_id]['title'])

df.loc[row\_id,'lyrics'] = lyrics

print(row\_id)

print('downloaded Lyrics for %s songs' %sum(df.lyrics!=''))

df.head()

df.to\_csv('lyr\_backup.csv')

df.to\_csv('msd\_genre\_dataset.csv')

# Preprocessing and training for songlyrics.com

%matplotlib inline

import matplotlib

import pandas as pd

import numpy as np

df = pd.read\_csv("dataset.csv", encoding = "ISO-8859-1")

temp = df['Lyrics']

temp2 = df['Genre']

df = pd.DataFrame(df.SongInfo.str.split(' - ',1).tolist(),columns=['Artist','SongTitle'])

df['SongTitle'] = df.SongTitle.str.replace(" LYRICS","")

df = df.join(temp)

df = df.join(temp2)

df['Genre'] = df.Genre.str.replace("Country Music","Country")

df['Genre'] = df.Genre.str.replace("Hip Hop/Rap","Rap")

df

df= df.drop\_duplicates(subset="SongTitle")

df['Genre'].value\_counts()

wordcounts = df.Lyrics.str.split().apply(len)

df['WordCounts'] = wordcounts

df

stats = df.groupby('Genre',as\_index=False)['WordCounts'].mean()

stats = stats.rename(columns = {'WordCounts':'Mean'})

median = df.groupby('Genre')['WordCounts'].median()

minvalue = df.groupby('Genre')['WordCounts'].min()

maxvalue = df.groupby('Genre')['WordCounts'].max()

stats['Median'] = median.get\_values()

stats['Min'] = minvalue.get\_values()

stats['Max'] = maxvalue.get\_values()

stats

stats.plot(kind="bar",x="Genre")

from wordcloud import WordCloud

import matplotlib.pyplot as plt

from sklearn.feature\_extraction import text

from wordcloud import WordCloud, STOPWORDS

genres = df['Genre'].unique()

for i in range(0,len(genres)):

words = ' '.join(df.loc[df['Genre']==genres[i], 'Lyrics'])

wordcloud = WordCloud(

stopwords=STOPWORDS,

background\_color='white',

width=800,

height=400

).generate(words)

print(genres[i])

plt.figure(figsize=(10, 5))

plt.imshow(wordcloud)

plt.axis('off')

plt.show()

words = ' '.join(df.loc[:, 'Lyrics'])

wordcloud = WordCloud(

stopwords=STOPWORDS,

background\_color='white',

width=800,

height=400

).generate(words)

print("ALL GENRES:")

plt.figure(figsize=(10, 5))

plt.imshow(wordcloud)

plt.axis('off')

plt.show()

from sklearn.preprocessing import LabelEncoder

import pickle

import numpy as np

from sklearn.model\_selection import train\_test\_split

train, test = train\_test\_split(df, test\_size=0.4, random\_state=0)

X\_train = train['Lyrics'].values

y\_train = train['Genre'].values

X\_test = test['Lyrics'].values

y\_test = test['Genre'].values

print('before: %s ...' %y\_train[:5])

le = LabelEncoder()

le.fit(y\_train)

y\_train = le.transform(y\_train)

print('after: %s ...' %y\_train[:5])

# Porter Stemmer

import nltk

import string

import re

from nltk.stem.snowball import EnglishStemmer

snowball = EnglishStemmer()

porter\_stemmer = nltk.stem.porter.PorterStemmer()

def porter\_tokenizer(text, stemmer=porter\_stemmer):

lower\_txt = text.lower()

tokens = nltk.wordpunct\_tokenize(lower\_txt)

stems = [porter\_stemmer.stem(t) for t in tokens]

no\_punct = [s for s in stems if re.match('^[a-zA-Z]+$', s) is not None]

return no\_punct

def snowball\_tokenizer(text, stemmer=snowball):

lower\_txt = text.lower()

tokens = nltk.wordpunct\_tokenize(lower\_txt)

stems = [snowball.stem(t) for t in tokens]

no\_punct = [s for s in stems if re.match('^[a-zA-Z]+$', s) is not None]

return no\_punct

import nltk.corpus

# Commented out to prevent overwriting files:

#

stp = nltk.corpus.stopwords.words('english')

with open('./stopwords\_eng.txt', 'w') as outfile:

outfile.write('\n'.join(stp))

with open('./stopwords\_eng.txt', 'r') as infile:

stop\_words = infile.read().splitlines()

print('stop words %s ...' %stop\_words[:5])

#stop\_words = []

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.naive\_bayes import BernoulliNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.cluster import KMeans

from sklearn.pipeline import Pipeline

from mlxtend.preprocessing import DenseTransformer

import seaborn as sns

from sklearn import metrics

final\_clf = Pipeline([

('vect', CountVectorizer(

encoding='utf-8',

decode\_error='replace',

strip\_accents='unicode',

analyzer='word',

stop\_words=stop\_words,

tokenizer=porter\_tokenizer,

binary=True,

ngram\_range=(1,1),

)

),

#('dense', DenseTransformer()),

#('clf', MultinomialNB(alpha=0.1,fit\_prior=False)),

('clf', BernoulliNB(alpha=0.001,fit\_prior=False)),

#('clf', DecisionTreeClassifier(criterion="gini",random\_state=0,splitter="best",min\_samples\_split=2)),

#('clf', RandomForestClassifier(criterion="entropy",random\_state=0,oob\_score=False,max\_depth=40,n\_estimators=500)),

#('clf', KNeighborsClassifier(weights="distance",algorithm="auto",p=2,n\_neighbors=6)),

])

final\_clf.fit(X\_train, y\_train)

#import matplotlib as plt

import numpy as np

from matplotlib import pyplot as plt

from sklearn.metrics import accuracy\_score

print(accuracy\_score(y\_train, final\_clf.predict(X\_train)))

plt.figure(figsize=(5, 5))

cm = metrics.confusion\_matrix(y\_train, final\_clf.predict(X\_train))

np.set\_printoptions(suppress=True)

hm = sns.heatmap(cm,

cbar=False,

annot=True,

square=True,

fmt='d',

yticklabels=['christian','country','rock','pop','rap','r&b'],

xticklabels=['christian','country','rock','pop','rap','r&b'],

cmap='Blues'

)

plt.title('Confusion matrix - Training dataset')

plt.ylabel('actual class')

plt.xlabel('predicted class')

plt.tight\_layout()

plt.show()

y\_test = le.transform(y\_test)

print(accuracy\_score(y\_test, final\_clf.predict(X\_test)))

plt.figure(figsize=(5, 5))

cm = metrics.confusion\_matrix(y\_test, final\_clf.predict(X\_test))

np.set\_printoptions(suppress=True)

hm = sns.heatmap(cm,

cbar=False,

annot=True,

square=True,

fmt='d',

yticklabels=['christian','country','rock','pop','rap','r&b'],

xticklabels=['christian','country','rock','pop','rap','r&b'],

cmap='Blues'

)

plt.title('Confusion matrix - Training dataset')

plt.ylabel('actual class')

plt.xlabel('predicted class')

plt.tight\_layout()

plt.show()

# Preprocessing and training for million song dataset

%matplotlib inline

import matplotlib

import pandas as pd

import numpy as np

df = pd.read\_csv("million\_10000.csv", encoding = "ISO-8859-1")

df['genre'] = df.genre.str.replace("Hip Hop/Rap","Rap")

df

df= df.drop\_duplicates(subset="title")

df['genre'].value\_counts()

wordcounts = df.lyrics.str.split().apply(len)

df['WordCounts'] = wordcounts

df

stats = df.groupby('genre',as\_index=False)['WordCounts'].mean()

stats = stats.rename(columns = {'WordCounts':'Mean'})

median = df.groupby('genre')['WordCounts'].median()

minvalue = df.groupby('genre')['WordCounts'].min()

maxvalue = df.groupby('genre')['WordCounts'].max()

stats['Median'] = median.get\_values()

stats['Min'] = minvalue.get\_values()

stats['Max'] = maxvalue.get\_values()

stats

stats.plot(kind="bar",x="genre")

from wordcloud import WordCloud

import matplotlib.pyplot as plt

from sklearn.feature\_extraction import text

from wordcloud import WordCloud, STOPWORDS

genres = df['genre'].unique()

for i in range(0,len(genres)):

words = ' '.join(df.loc[df['genre']==genres[i], 'lyrics'])

wordcloud = WordCloud(

stopwords=STOPWORDS,

background\_color='white',

width=800,

height=400

).generate(words)

print(genres[i])

plt.figure(figsize=(10, 5))

plt.imshow(wordcloud)

plt.axis('off')

plt.show()

words = ' '.join(df.loc[:, 'lyrics'])

wordcloud = WordCloud(

stopwords=STOPWORDS,

background\_color='white',

width=800,

height=400

).generate(words)

print("ALL GENRES:")

plt.figure(figsize=(10, 5))

plt.imshow(wordcloud)

plt.axis('off')

plt.show()

from sklearn.preprocessing import LabelEncoder

import pickle

import numpy as np

from sklearn.model\_selection import train\_test\_split

train, test = train\_test\_split(df, test\_size=0.4, random\_state=0)

X\_train = train['lyrics'].values

y\_train = train['genre'].values

X\_test = test['lyrics'].values

y\_test = test['genre'].values

print('before: %s ...' %y\_train[:5])

le = LabelEncoder()

le.fit(y\_train)

y\_train = le.transform(y\_train)

print('after: %s ...' %y\_train)

# Porter Stemmer

import nltk

import string

import re

from nltk.stem.snowball import EnglishStemmer

snowball = EnglishStemmer()

porter\_stemmer = nltk.stem.porter.PorterStemmer()

def porter\_tokenizer(text, stemmer=porter\_stemmer):

lower\_txt = text.lower()

tokens = nltk.wordpunct\_tokenize(lower\_txt)

stems = [porter\_stemmer.stem(t) for t in tokens]

no\_punct = [s for s in stems if re.match('^[a-zA-Z]+$', s) is not None]

return no\_punct

def snowball\_tokenizer(text, stemmer=snowball):

lower\_txt = text.lower()

tokens = nltk.wordpunct\_tokenize(lower\_txt)

stems = [snowball.stem(t) for t in tokens]

no\_punct = [s for s in stems if re.match('^[a-zA-Z]+$', s) is not None]

return no\_punct

import nltk.corpus

# Commented out to prevent overwriting files:

#

stp = nltk.corpus.stopwords.words('english')

with open('./stopwords\_eng.txt', 'w') as outfile:

outfile.write('\n'.join(stp))

with open('./stopwords\_eng.txt', 'r') as infile:

stop\_words = infile.read().splitlines()

print('stop words %s ...' %stop\_words[:5])

#stop\_words = []

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.naive\_bayes import BernoulliNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.cluster import KMeans

from sklearn.pipeline import Pipeline

from mlxtend.preprocessing import DenseTransformer

import seaborn as sns

from sklearn import metrics

final\_clf = Pipeline([

('vect', CountVectorizer(

encoding='utf-8',

decode\_error='replace',

strip\_accents='unicode',

analyzer='word',

stop\_words=stop\_words,

tokenizer=porter\_tokenizer,

binary=True,

ngram\_range=(1,1),

)

),

#('dense', DenseTransformer()),

#('clf', MultinomialNB(alpha=0.1,fit\_prior=False)),

('clf', BernoulliNB(alpha=0.001,fit\_prior=False)),

#('clf', DecisionTreeClassifier(criterion="gini",random\_state=0,splitter="best",min\_samples\_split=2)),

#('clf', RandomForestClassifier(criterion="entropy",random\_state=0,oob\_score=False,max\_depth=40,n\_estimators=500)),

#('clf', KNeighborsClassifier(weights="distance",algorithm="auto",p=2,n\_neighbors=6)),

])

final\_clf.fit(X\_train, y\_train)

labels = np.arange(len(genres)-1)

labels = le.inverse\_transform(labels)

#import matplotlib as plt

import numpy as np

from matplotlib import pyplot as plt

from sklearn.metrics import accuracy\_score

print(accuracy\_score(y\_train, final\_clf.predict(X\_train)))

plt.figure(figsize=(5, 5))

cm = metrics.confusion\_matrix(y\_train, final\_clf.predict(X\_train))

np.set\_printoptions(suppress=True)

hm = sns.heatmap(cm,

cbar=False,

annot=True,

square=True,

fmt='d',

yticklabels=labels,

xticklabels=labels,

cmap='Blues'

)

plt.title('Confusion matrix - Training dataset')

plt.ylabel('actual class')

plt.xlabel('predicted class')

plt.tight\_layout()

plt.show()

import bisect

test['genre'] = test['genre'].map(lambda s: 'other' if s not in le.classes\_ else s)

le\_classes = le.classes\_.tolist()

bisect.insort\_left(le\_classes, 'other')

le.classes\_ = le\_classes

y\_test = le.transform(y\_test)

print(accuracy\_score(y\_test, final\_clf.predict(X\_test)))

plt.figure(figsize=(5, 5))

cm = metrics.confusion\_matrix(y\_test, final\_clf.predict(X\_test))

np.set\_printoptions(suppress=True)

hm = sns.heatmap(cm,

cbar=False,

annot=True,

square=True,

fmt='d',

yticklabels=labels,

xticklabels=labels,

cmap='Blues'

)

plt.title('Confusion matrix - Training dataset')

plt.ylabel('actual class')

plt.xlabel('predicted class')

plt.tight\_layout()

plt.show()