#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Wed Mar 20 22:46:46 2019

@author: jiangzhaobo

"""

import pandas as pd

import numpy as np

#import xgboost as xgb

from sklearn.svm import SVC

from keras.models import Sequential

from keras.layers.recurrent import LSTM, GRU

from keras.layers.core import Dense, Activation, Dropout

from keras.layers.embeddings import Embedding

from keras.layers.normalization import BatchNormalization

from keras.utils import np\_utils

from sklearn import preprocessing, decomposition, model\_selection, metrics, pipeline

from sklearn.model\_selection import GridSearchCV

from sklearn.feature\_extraction.text import TfidfVectorizer, CountVectorizer

from sklearn.decomposition import TruncatedSVD

from sklearn.linear\_model import LogisticRegression

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

from sklearn.metrics import classification\_report

from sklearn.naive\_bayes import MultinomialNB

from keras.layers import GlobalMaxPooling1D, Conv1D, MaxPooling1D, Flatten, Bidirectional, SpatialDropout1D

from keras.preprocessing import sequence, text

from keras.callbacks import EarlyStopping

from nltk import word\_tokenize

import nltk

from nltk.stem import WordNetLemmatizer

import string

from nltk.corpus import stopwords

from nltk.stem.porter import \*

import csv

#read relevant csv file

data=pd.read\_csv('Growth.csv',encoding='ISO-8859-1')

''' Converting data to two different lists'''

data=np.array(data.loc[:,:])

list\_sentences = []

list\_classifier = []

for i in range (0,data.shape[0]):

temp\_clasifier = data[i][0]

temp\_sentence = data[i][1]

list\_classifier.append(temp\_clasifier)

list\_sentences.append(str(temp\_sentence).lower())

#print(len(list\_sentences))

#print(len(list\_classifier))

''' Removing the punctuation '''

list\_sentences\_no\_punc = []

for sent in list\_sentences:

for c in string.punctuation:

sent = sent.replace(c," ")

list\_sentences\_no\_punc.append(sent)

#print(list\_sentences\_no\_punc)

''' Lemmatization '''

lemma = nltk.wordnet.WordNetLemmatizer()

lematized\_list = []

for sent in list\_sentences\_no\_punc:

temp\_list = []

temp\_line\_list = word\_tokenize(sent)

for word in temp\_line\_list:

temp\_list.append(lemma.lemmatize(word, pos = "v").strip())

lematized\_line = ' '.join(temp\_list)

lematized\_list.append(lematized\_line)

#print(lematized\_list)

''' Removing stopwords '''

list\_without\_stopwords = []

list\_stopwords = stopwords.words('english')

for sent in lematized\_list:

temp\_word\_list = word\_tokenize(sent)

temp\_list = []

for word in temp\_word\_list:

if word not in list\_stopwords:

temp\_list.append(word)

line\_without\_stopwords = ' '.join(temp\_list)

list\_without\_stopwords.append(line\_without\_stopwords)

#print(list\_without\_stopwords)

#print(len(list\_without\_stopwords))

''' Writing a csv file '''

with open('Lemmatized\_file.csv', mode='w', encoding = "utf-8") as csv\_file:

writer = csv.writer(csv\_file)

writer.writerow(["Classification", "Sentences"])

writer.writerows(zip(list\_classifier, lematized\_list))

#csv\_file.close()

''' Reading the file '''

data = pd.read\_csv('Lemmatized\_file.csv' )

#print(data)

#Read csv file

data=pd.read\_csv('Lemmatized\_file.csv')

data.head()

data.info()

#identify how many labels

data.Classification.unique()

ab=data.Sentences.shape

#put the text to integer

lbl\_enc = preprocessing.LabelEncoder()

y = lbl\_enc.fit\_transform(data.Classification.values)

#randome split the data

xtrain, xvalid, ytrain, yvalid = train\_test\_split(data.Sentences.values, y, stratify=y,random\_state=42, test\_size=0.1, shuffle=True)

print (xtrain.shape)

print (xvalid.shape)

#put number as placeholder

def number\_normalizer(tokens):

return ("#NUMBER" if token[0].isdigit() else token for token in tokens)

class NumberNormalizingVectorizer(TfidfVectorizer):

def build\_tokenizer(self):

tokenize = super(NumberNormalizingVectorizer, self).build\_tokenizer()

return lambda doc: list(number\_normalizer(tokenize(doc)))

tfv = NumberNormalizingVectorizer(min\_df=3,

max\_df=0.5,

max\_features=None,

ngram\_range=(1, 2),

use\_idf=True,

smooth\_idf=True)

#using tf-idf to fit the training dataset and test dataset

tfv.fit(list(xtrain) + list(xvalid))

xtrain\_tfv = tfv.transform(xtrain)

xvalid\_tfv = tfv.transform(xvalid)

#knn

from sklearn.neighbors import KNeighborsClassifier

clf=KNeighborsClassifier(n\_neighbors = 10 , weights='distance')

clf.fit(xtrain\_tfv, ytrain)

predictions = clf.predict(xvalid\_tfv)

print(predictions)

type(yvalid)

actual= np.array(yvalid.tolist())

actual

rere=0

reir=0

irre=0

irir=0

nnn=len(actual)

for ii in range(0,nnn):

if actual[ii]==1:

if predictions[ii]==1:

rere+=1

else:

irre+=1

else:

if predictions[ii]==1:

reir+=1

else:

irir+=1

#The accurary of KNN

accur=(irir+rere)/(irir+rere+reir+irre)

print('The accuracy of KNN classifier is :',+accur)

#The relevant recall

reca=(rere)/(rere+reir)

print('The relevant recall of KNN classifier is :',+reca)

#The irrelevant recall

irca=(irir)/(irre+irir)

print('The irrelevant recall of KNN classifier is :',+irca)

#The relevant precision

repr=(rere)/(rere+irre)

print('The relevant precision of KNN classifier is :',+repr)

#The irrelevant precision

irpr=(irir)/(irir+reir)

print('The irrelevant precision of KNN classifier is :',+irpr)

from sklearn.metrics import roc\_curve

fpr, tpr, thresholds = roc\_curve(actual, predictions)

print(fpr)

print(tpr)

print(thresholds)

from sklearn.metrics import roc\_auc\_score

auc = roc\_auc\_score(actual, predictions)

print("The auc:",+auc)

def class\_logloss(actual, predicted, eps=1e-15):

clip = np.clip(predicted, eps, 1 - eps)

rows = actual.shape[0]

vsota = np.sum(actual \* np.log(clip))

return -1.0 / rows \* vsota

print ("logloss: %0.3f " % class\_logloss(yvalid, predictions))

import matplotlib.pyplot as plt

plt.plot(fpr,tpr)

plt.title("KNN-Growth")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.fill\_between(fpr, tpr, where=(tpr>=0), color='Blue', alpha=0.5)

plt.show()