import numpy as np

from keras import initializers

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import OneHotEncoder

from sklearn.feature\_extraction.text import CountVectorizer

import os

def pre\_processing(): #separates the words from its classes

classes\_set = set() # this list will keep track of all classes of words

classes = open("words.txt","w")

words = open("classes.txt", "w")

with open("train\_data.txt","r") as f:

text = f.read()

words\_list = text.split()

for word in words\_list: # get all different tokens in text

class\_aux = word.split("\\")

classes\_set.add(class\_aux[1])

words.write(str(class\_aux[1])+" ")

classes.write(str(class\_aux[0])+" ")

classes.close()

words.close()

print("Pre-processing Done")

def getOneHot(test\_corpus): #returns one-hot index

for i in range(0,len(test\_corpus)):

if(test\_corpus[i]==1):

return i

def return\_training\_data(text, window\_size, epochs):

data\_train = []

classes\_train = []

vectorizer = CountVectorizer(lowercase=False, token\_pattern='[A-Z;+;-]+')

corpus = vectorizer.fit\_transform(text)

corpus = corpus.toarray()

window\_start=0 #sliding window

window\_end=window\_size-1 #sliding window

while(window\_end<len(corpus)-1):

window\_end += 1

data\_train.append(corpus[window\_start:window\_end])

classes\_train.append(corpus[window\_end])

window\_start += 1

data\_train = np.array(data\_train)

classes\_train = np.array(classes\_train)

return data\_train,classes\_train,vectorizer,corpus

def return\_testing\_data(vectorizer, window\_size, corpus, text):

data\_test = []

classes\_test = []

test\_corpus = vectorizer.fit\_transform(text)

test\_corpus = test\_corpus.toarray()

knownTestByClass = {}

predictedTestByClass = {}

window\_start=0 #sliding window

window\_end=window\_size-1 #sliding window

while(window\_end<len(test\_corpus)-1):

index = getOneHot(test\_corpus[(window\_end)])

if index not in knownTestByClass:

knownTestByClass[index] = []

if index not in predictedTestByClass:

predictedTestByClass[index] = []

window\_end += 1

knownTestByClass[index].append(test\_corpus[window\_start:window\_end])

predictedTestByClass[index].append(test\_corpus[window\_end])

data\_test.append(corpus[window\_start:window\_end])

classes\_test.append(corpus[window\_end])

window\_start += 1

for i in knownTestByClass:

knownTestByClass[i] = np.array(knownTestByClass[i])

for i in predictedTestByClass:

predictedTestByClass[i] = np.array(predictedTestByClass[i])

data\_test = np.array(data\_test)

classes\_test = np.array(classes\_test)

return data\_test,classes\_test,knownTestByClass,predictedTestByClass

def create\_model(window\_size,data\_train,classes\_train,epochs,batch\_size):

np.random.seed(13)

model = Sequential()

model.add(LSTM(68,input\_shape=(window\_size,41)))

model.add(Dense(40,activation='relu'))

model.add(Dense(5, activation='sigmoid'))

model.add(Dense(41, activation=lambda x: x))

model.compile(loss='binary\_crossentropy',optimizer ='rmsprop',metrics=['accuracy'])

model.fit(data\_train,classes\_train,epochs=epochs, batch\_size=batch\_size,validation\_split=0.2,verbose=1)

return model

def main(window\_size,epochs,batch\_size):

with open("classes.txt","r") as classes:

text = classes.read().split(" ")

data\_train = []

classes\_train = []

data\_train,classes\_train,vectorizer,corpus = return\_training\_data(text, window\_size, epochs)

model = create\_model(window\_size,data\_train,classes\_train,epochs,batch\_size)

#generating test samples

data\_test = []

classes\_test = []

data\_test,classes\_test,knownTestByClass,predictedTestByClass = return\_testing\_data(vectorizer, window\_size, corpus, text)

result\_file\_name = str(window\_size)+'-'+str(epochs) # stores the LSTM's parameters as string to use as file name

with open("total\_accuracy.csv", "a+") as f:

f.write("window\_size,epochs,batch\_size,accuracy\n")

f.write(str(window\_size)+","+str(epochs)+","+str(batch\_size)+","+str(model.evaluate(data\_test,classes\_test,batch\_size=batch\_size,verbose=2)[1])+"\n")

with open("result.csv","w") as f:

f.write("index,accuracy\n")

classes\_list = vectorizer.get\_feature\_names() # will be used to return each class's accuracy, but without using an index

for index in knownTestByClass:

score = model.evaluate(knownTestByClass[index],predictedTestByClass[index],batch\_size=batch\_size,verbose=2)

with open("result.csv","a") as f:

f.write(str(classes\_list[index])+","+str(score[1])+"\n")

pre\_processing()

for i in range(3,6):

main(i,100,200)