**Report for extra Credit**

**I have trained word embeddings of word2vec using train method and store it in a model**

**Through importing keyvectors I have stored vectors for a word in the given model which is then passed to MLP function I have used that finetuned model in the multilayer prepceptron for training the data and testing it on test data.**

from gensim.models import Word2Vec

med=dict()

gh=[]

def update\_embeddings(path\_to\_train\_file):

   df2=pd.read\_csv(path\_to\_train\_file, sep='\t')

   label = df2['subtask\_a']

   label1 = le.fit\_transform(df2['subtask\_a'])

   columns = ['id','subtask\_b','subtask\_c']

   df2.drop(columns, axis=1, inplace=True)

   listoftweets = []

   for i in range(0,len(df2)):

        listoftweets.append(df2.iloc[i,0])

   gh=clean(listoftweets)

   #for gh in listoftweets:

   model = Word2Vec(sentences=gh, size=100, window=5, min\_count=1, workers=4)

   op = model.save("word2vec.model")

   print(op)

   med = Word2Vec.load("word2vec.model")

   print(med)

   model.train(gh, total\_examples=1, epochs=1)

   return op, med

from gensim.models import KeyedVectors

# Store just the words + their trained embeddings.

word\_vectors = med.wv

word\_vectors.save("word2vec.wordvectors")

# Load back with memory-mapping = read-only, shared across processes.

wv = KeyedVectors.load("word2vec.wordvectors", mmap='r')

The wv is the final dictionary through which word embeddings wiil be used:

The modified MLP function :

from keras\_preprocessing.text import Tokenizer

from keras\_preprocessing.sequence import pad\_sequences

from sklearn.neural\_network import MLPClassifier

from sklearn.datasets import make\_classification

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

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

import math

import pandas as pd

gh=[]

tokenizer=Tokenizer()

surbhi12 = dict()

vocab\_size=0

embeddings\_index = dict()

def train\_MLP\_model(path\_to\_train\_file, med, num\_layers = 2):

   df=pd.read\_csv(path\_to\_train\_file, sep='\t')

   label = df['subtask\_a']

   label1 = le.fit\_transform(df['subtask\_a'])

   columns = ['id','subtask\_b','subtask\_c']

   df.drop(columns, axis=1, inplace=True)

   listoftweets = []

   for i in range(0,len(df)):

        listoftweets.append(df.iloc[i,0])

   gh=clean(listoftweets)

   tokenizer.fit\_on\_texts(gh)

   tokenized\_documents=tokenizer.texts\_to\_sequences(gh)

   tokenized\_paded\_documents=pad\_sequences(tokenized\_documents,maxlen=64,padding='post')

   print(tokenizer.word\_index)

   vocab\_size=len(tokenizer.word\_index)+1

   print (tokenized\_paded\_documents[0])

   #reading Glove word embeddings into a dictionary with "word" as key and values as word vectors

  #  embeddings\_index = dict()

  #  with open('glove.6B.100d.txt') as file:

  #       for line in file:

  #          values = line.split()

  #          word = values[0]

  #          coefs = np.asarray(values[1:], dtype='float32')

  #          embeddings\_index[word] = coefs

   # creating embedding matrix, every row is a vector representation from the vocabulary indexed by the tokenizer index.

   embedding\_matrix=np.zeros((vocab\_size,100))

  #  for  hj in range(0,len(tokenized\_paded\_documents)):

  #        for tv in range(0,len(tokenized\_paded\_documents[hj])):

  #             if tokenized\_paded\_documents[hj][tv] in tokenizer.word\_index.items

   for word,i in tokenizer.word\_index.items():

      if med[word] is not None:

         embedding\_vector = med[word]

    #print(embedding\_vector)

      if embedding\_vector is not None:

          embedding\_matrix[i] = embedding\_vector

   document\_embeddings=np.zeros((len(tokenized\_paded\_documents),100))

   dict\_keys=list(tokenizer.word\_index.keys())

   dict\_values=list(tokenizer.word\_index.values())

   for i in range(len(tokenized\_paded\_documents)):

      joh = np.zeros((1,100),  dtype = np.float32)

      for j in range(len(tokenized\_paded\_documents[0])):

            print(tokenized\_paded\_documents[i][j])

            if tokenized\_paded\_documents[i][j] != 0:

               val\_index=dict\_values.index(tokenized\_paded\_documents[i][j])

               print("Associated key is:")

               myKey=dict\_keys[val\_index]

               print(myKey)

               if med[myKey] is not None:

                  surbhi12[myKey]= med[myKey]

              # print("test devesh",embeddings\_index.get(myKey))

               #print(embeddings\_index.get(myKey).dtype)

                  joh += med[myKey]

      document\_embeddings[i]=joh

      document\_embeddings.shape

   for i in range(0,len(document\_embeddings)):

     for j in range(0,len(document\_embeddings[i])):

           if(math.isnan(document\_embeddings[i][j])):

                      document\_embeddings[i][j]=0;

   X\_train, X\_test, y\_train, y\_test = train\_test\_split(document\_embeddings, label1, stratify=label1,

                                                     random\_state=1)

   if num\_layers==1:

     clf = MLPClassifier(hidden\_layer\_sizes=(1), activation='logistic', solver='adam', max\_iter=2000)

   elif num\_layers==2:

     clf = MLPClassifier(hidden\_layer\_sizes=(9,1), activation='logistic', solver='adam', max\_iter=2000)

   elif num\_layers==3:

     clf = MLPClassifier(hidden\_layer\_sizes=(9,12, 1), activation='logistic', solver='adam', max\_iter=2000)

   elif num\_layers==4:

      clf = MLPClassifier(hidden\_layer\_sizes=(9,12, 15,1), activation='logistic', solver='adam', max\_iter=2000)

   hn = clf.fit(X\_train, y\_train)

   print(hn.predict\_proba(X\_test))

   predicted\_values = hn.predict(X\_test)

   print(confusion\_matrix(y\_test,predicted\_values))

   print(classification\_report(y\_test,predicted\_values))

   return hn

iu = train\_MLP\_model("/content/olid-training-v1.0.tsv",  wv, num\_layers = 2)

**Results:**

**Result for Two layer:**

**Pretrained Results:**

**Predicted probabilities for test data out of training data:**

[[0.58463613 0.41536387]

[0.94721926 0.05278074]

[0.53609484 0.46390516]

...

[0.86768698 0.13231302]

[0.95292953 0.04707047]

[0.07924145 0.92075855]]

**Confusion Matrix**

[[1792 418]

[ 473 627]]

precision recall f1-score support

0 0.79 0.81 0.80 2210

1 0.60 0.57 0.58 1100

accuracy 0.73 3310

macro avg 0.70 0.69 0.69 3310

weighted avg 0.73 0.73 0.73 3310

**Fine Tuned Results:**

**Results from fine tuned on test set**

**Predicted Probabilities**

[0.6639977 0.3360023 ]

[0.66399746 0.33600254]

[0.66401175 0.33598825]

...

[0.66705955 0.33294045]

[0.66399759 0.33600241]

[0.66400125 0.33599875]]

**Confusion matrix**

[[2210 0]

[1100 0]]

precision recall f1-score support

0 0.67 1.00 0.80 2210

1 0.00 0.00 0.00 1100

accuracy 0.67 3310

macro avg 0.33 0.50 0.40 3310

weighted avg 0.45 0.67 0.53 3310

**OBSERVATION:**

**Pretrained models have shown greater accuracy as compared to fine tuned embeddings.**

**I think pretrained embeddings have been made on large vocab rather than fine tuned embedding on smaller corpus of dataset might result in accuracy drop to 67 % from 73.**

**The above comparision has been done only for two hidden layers.**

**Links to the model Saved and keyvectored:**

**For wordkeyvectored:**

[**https://drive.google.com/file/d/1ir\_4MPkBkuK4N0C8G\_51llakJHBIyYQI/view?usp=sharing**](https://drive.google.com/file/d/1ir_4MPkBkuK4N0C8G_51llakJHBIyYQI/view?usp=sharing)

**For the model saved:**

**https://drive.google.com/file/d/1wUnm\_rogZPS0yNRbNI8EmS8PJor68vWe/view?usp=sharing**