Milestone 4 - LSTM and GloveVector word embeddings

Ali Wehbe - Dina Younes - Shaza Fakih - Reeda El Saintbai - Youssef Jaafar

```
from google.colab import drive
drive.mount("/content/drive")

     Mounted at /content/drive
```

Note:

- make sure to upload the "embedding_matrix.pkl" and "iseardataset.csv" file in the same directory as this ipynb file.

Downloading GloveVector

Note:

- make sure you create a folder named "embeddings" in the same directory of this .ipynb file before running the code below

```
import zipfile
!wget http://nlp.stanford.edu/data/glove.840B.300d.zip      #needed for using the word embedd
zip_file = zipfile.ZipFile('glove.840B.300d.zip')
zip_file.extractall('./embeddings/')

     --2020-12-03 17:42:04--  http://nlp.stanford.edu/data/glove.840B.300d.zip
     Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
     Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
     HTTP request sent, awaiting response... 302 Found
     Location: https://nlp.stanford.edu/data/glove.840B.300d.zip [following]
     --2020-12-03 17:42:04--  https://nlp.stanford.edu/data/glove.840B.300d.zip
     Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
     HTTP request sent, awaiting response... 301 Moved Permanently
     Location: http://downloads.cs.stanford.edu/nlp/data/glove.840B.300d.zip [following]
     --2020-12-03 17:42:05--  http://downloads.cs.stanford.edu/nlp/data/glove.840B.300d.zip
     Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
     Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:80...
     HTTP request sent, awaiting response... 200 OK
     Length: 2176768927 (2.0G) [application/zip]
     Saving to: ‘glove.840B.300d.zip.1’

     glove.840B.300d.zip 100%[===================>]   2.03G  1.96MB/s    in 16m 55s

     2020-12-03 17:59:00 (2.05 MB/s) - ‘glove.840B.300d.zip.1’ saved [2176768927/2176768927]
```

```
    str = re.sub(r"\'ll", " \'ll", str)           #ex: "I'll" becomes "I 'll"
    str = re.sub(r",", " , ", str)        # here we split a word from punctuations: ex: "hell
    str = re.sub(r"!", " ! ", str)
    str = re.sub(r"\(", " \( ", str)
    str = re.sub(r"\)", " \) ", str)      #ex: "(for)" -> "\\( for \\)"
    str = re.sub(r"\?", " \? ", str)      #ex: "done?" -> "done \\?"
    str = re.sub(r"\s{2,}", " ", str)
    return str.strip().lower()           #converting to lower case

#example
str = " I've broken my leg skiing the previous winter— first time down the hill— I suffered a
clean_words(str)


################################################################################
#Loading Data

import pandas as pd
from keras.utils.np_utils import to_categorical

DIR_DATA = '/content/drive/My Drive/Colab Notebooks/'
filename = 'iseardataset.csv'

df = pd.read_csv(DIR_DATA + filename)
needed = ['label', 'text']
not_needed = list(set(df.columns) - set(needed))
df = df.drop(not_needed, axis=1)
df = df.dropna(axis=0, how='any', subset=needed)
y_labels = sorted(list(set(df[needed[0]].tolist())))
dict.fromkeys(set(df[needed[0]].tolist()))
labels_dictionary = {}
for i in range(len(y_labels)):
  labels_dictionary[y_labels[i]] = i

x_train = df[needed[1]].apply(lambda x: clean_words(x)).tolist()      #cleaning sentences
y_train = df[needed[0]].apply(lambda y: labels_dictionary[y]).tolist()
y_train = to_categorical(np.asarray(y_train))

sentences = x_train
y_labels = y_train


################################################################################
# using GloveVector  word embeddings

import os
import pandas as pd
np.random.seed(7)
filename2 = 'glove.840B.300d.txt'
DIR_GLOVE = './embeddings/'

embeddings = {}
embeddings = {}
file1 = open(os.path.join(DIR_GLOVE, filename2), encoding='utf-8')
i = 0
for line in file1:
    values = line.split()
    word = values[0]
    try:
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings[word] = coefs
    except ValueError:
        i += 1
file1.close()



##################################################################################
#Creating Vocab and Data

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

MAX_NB_WORDS = 20000
MAX_SEQUENCE_LENGTH = 100

tokenizer1 = Tokenizer(num_words=MAX_NB_WORDS)               #tokenizing
tokenizer1.fit_on_texts(sentences)
sequences = tokenizer1.texts_to_sequences(sentences)
vocab = tokenizer1.word_index
data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)


##################################################################################
#creating embedding matrix

MAX_NB_WORDS = 20000
EMBEDDING_DIM = 300     #created a 300-dim embedding

word_index = vocab
embeddings_index = embeddings

number_of_words = min(MAX_NB_WORDS, len(word_index))
embedding_matrix = np.zeros((number_of_words + 1, EMBEDDING_DIM))  #created a 300-dim embeddi
for word, i in word_index.items():
    if i > MAX_NB_WORDS:
        continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

embedding_mat = embedding_matrix
#the embedding word vectors of the embedding matrix are then fed to the NN model
```

Function used for cleaning words

```
import re
def clean_words(str):                  #cleaning strings (removing excessive spaces, punctuat
    str = re.sub(r"[^A-Za-z0-9(),!?\'\`]", " ", str)
    str = re.sub(r"\'s", " \'s", str)             #ex: "my friend's father" becomes "my frien
    str = re.sub(r"\'ve", " \'ve", str)           #ex: "I've" -> "I 've"
    str = re.sub(r"n\'t", " n\'t", str)           #ex: "don't" -> "do n't"
    str = re.sub(r"\'re", " \'re", str)           #ex: "they're" -> "they 're"
    str = re.sub(r"\'d", " \'d", str)             #ex: "I'd" becomes "I 'd"
    str = re.sub(r"\'ll", " \'ll", str)           #ex: "I'll" becomes "I 'll"
    str = re.sub(r",", " , ", str)        # here we split a word from punctuations: ex: "hell
    str = re.sub(r"!", " ! ", str)
    str = re.sub(r"\(", " \( ", str)
    str = re.sub(r"\)", " \) ", str)      #ex: "(for)" -> "\\( for \\)"
    str = re.sub(r"\?", " \? ", str)      #ex: "done?" -> "done \\?"
    str = re.sub(r"\s{2,}", " ", str)
    return str.strip().lower()           #converting to lower case

#example
str = " I've broken my leg, while skiing the previous winter— "
print("example sentence1:")
print(str)
print("cleaned sentence1:")
print(clean_words(str))

str = "I suffered a lot!! Don't you want to help? (Please) "
print("example sentence2:")
print(str)
print("cleaned sentence2:")
clean_words(str)
```

```
example sentence1:
 I've broken my leg, while skiing the previous winter—
cleaned sentence1:
i 've broken my leg , while skiing the previous winter
example sentence2:
I suffered a lot!! Don't you want to help? (Please)
cleaned sentence2:
'i suffered a lot ! ! do n't you want to help \\? \\( please \\)'
```

```
import re
def clean_words(str):                  #cleaning strings (removing excessive spaces, punctuat
    str = re.sub(r"[^A-Za-z0-9(),!?\'\`]", " ", str)
    str = re.sub(r"\'s", " \'s", str)             #ex: "my friend's father" becomes "my frien
    str = re.sub(r"\'ve", " \'ve", str)           #ex: "I've" becomes "I 've"
    str = re.sub(r"n\'t", " n\'t", str)           #ex: "don't" becomes "do n't"
    str = re.sub(r"\'re", " \'re", str)           #ex: "they're" -> "they 're"
    str = re.sub(r"\'d", " \'d", str)             #ex: "I'd" becomes "I 'd"
```

```
################################################################################
#creating the LSTM model

from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers.recurrent import LSTM
from keras.layers.core import Dense, Activation
from keras.layers.embeddings import Embedding

pickle.dump([data, y_labels, embedding_mat], open('/content/drive/My Drive/Colab Notebooks/em
print ("Data created")

print("Train Test split")
X_train, X_test, y_train, y_test = train_test_split(data, y_labels, test_size=TEST_SPLIT, ran

TEST_SPLIT = 0.1        #best after trying 0.1, 0.2, and 0.3
VALIDATION_SPLIT = 0.1  #validation split fixed at 10% for validation dataset and 90% for tra
epoch = 40                          #specifying the number of epoch to be 40 for the number o
embedding_matrix = embedding_mat


lstmmodel = Sequential()
n, embedding_dims = embedding_matrix.shape

lstmmodel.add(Embedding(n, embedding_dims, weights=[embedding_matrix], input_length=MAX_SEQUE
lstmmodel.add(LSTM(128, dropout=0.6, recurrent_dropout=0.6))     #specifying a dropout probab
lstmmodel.add(Dense(7))                                          # we have 7 different output
lstmmodel.add(Activation('softmax'))                            #Dense is used for classifica
                                        #softmax used to have an output as a 7-dim vector

lstmmodel.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(lstmmodel.summary())

lstmmodel.fit(X_train, y_train, validation_split=VALIDATION_SPLIT, epochs=epoch, batch_size=1
lstmmodel.save_weights('/content/drive/My Drive/Colab Notebooks/text_lstm_weights.h5')

scores= lstmmodel.evaluate(X_test, y_test, verbose=0)
print("%s: %.2f%%" % (lstmmodel.metrics_names[1], scores[1] * 100))
```

```
48/48 [==============================] - 16s 330ms/step - loss: 1.0689 - accuracy: 0.
Epoch 13/40
48/48 [==============================] - 16s 331ms/step - loss: 1.0434 - accuracy: 0.
Epoch 14/40
48/48 [==============================] - 16s 330ms/step - loss: 1.0110 - accuracy: 0.
Epoch 15/40
48/48 [==============================] - 16s 328ms/step - loss: 1.0004 - accuracy: 0.
Epoch 16/40
48/48 [==============================] - 15s 323ms/step - loss: 0.9833 - accuracy: 0.
Epoch 17/40
48/48 [==============================] - 16s 330ms/step - loss: 0.9598 - accuracy: 0.
Epoch 18/40
```

```
48/48 [==============================] - 16s 328ms/step - loss: 0.9527 - accuracy: 0.
Epoch 19/40
48/48 [==============================] - 16s 337ms/step - loss: 0.9318 - accuracy: 0.
Epoch 20/40
48/48 [==============================] - 16s 326ms/step - loss: 0.9142 - accuracy: 0.
Epoch 21/40
48/48 [==============================] - 16s 328ms/step - loss: 0.8960 - accuracy: 0.
Epoch 22/40
48/48 [==============================] - 16s 333ms/step - loss: 0.8748 - accuracy: 0.
Epoch 23/40
48/48 [==============================] - 15s 322ms/step - loss: 0.8556 - accuracy: 0.
Epoch 24/40

48/48 [==============================] - 16s 331ms/step - loss: 0.8439 - accuracy: 0.
Epoch 25/40
48/48 [==============================] - 15s 321ms/step - loss: 0.8203 - accuracy: 0.
Epoch 26/40
48/48 [==============================] - 15s 317ms/step - loss: 0.8142 - accuracy: 0.
Epoch 27/40
48/48 [==============================] - 16s 326ms/step - loss: 0.7918 - accuracy: 0.
Epoch 28/40
48/48 [==============================] - 16s 326ms/step - loss: 0.7665 - accuracy: 0.
Epoch 29/40
48/48 [==============================] - 15s 321ms/step - loss: 0.7638 - accuracy: 0.
Epoch 30/40
48/48 [==============================] - 16s 330ms/step - loss: 0.7495 - accuracy: 0.
Epoch 31/40
48/48 [==============================] - 16s 326ms/step - loss: 0.7477 - accuracy: 0.
Epoch 32/40
48/48 [==============================] - 16s 327ms/step - loss: 0.7284 - accuracy: 0.
Epoch 33/40
48/48 [==============================] - 16s 324ms/step - loss: 0.7217 - accuracy: 0.
Epoch 34/40
48/48 [==============================] - 16s 323ms/step - loss: 0.7019 - accuracy: 0.
Epoch 35/40
48/48 [==============================] - 15s 322ms/step - loss: 0.6746 - accuracy: 0.
Epoch 36/40
48/48 [==============================] - 16s 323ms/step - loss: 0.6848 - accuracy: 0.
Epoch 37/40
48/48 [==============================] - 16s 333ms/step - loss: 0.6666 - accuracy: 0.
Epoch 38/40
48/48 [==============================] - 15s 321ms/step - loss: 0.6552 - accuracy: 0.
Epoch 39/40
48/48 [==============================] - 16s 325ms/step - loss: 0.6459 - accuracy: 0.
Epoch 40/40
48/48 [==============================] - 15s 321ms/step - loss: 0.6212 - accuracy: 0.
accuracy: 63.56%
```