CS6005 – DEEP LEARNING

ASSIGNMENT - 4

MINI PROJECT IN NLP

Github: https://github.com/SuryaaSeran/Emotion-prediction-from-text-using-Bi-LSTM

By V.S. Suryaa

2018103610

P-Batch

Emotion prediction from text using Bi-LSTM

Problem statement:

We try to classify the text by understanding the emotions behind it. The emotions considered are -sadness, anger, love, surprise, fear and happy. We use Bidirectional LSTM with a dense layer and an embedding layer with and without the pretrained GloVe weights. We analyse the performance of both the models and their learning process.

Bi-Directional LSTM:

A Bidirectional LSTM is a sequence processing model that consists of two LSTMs: one taking the input in a forward direction, and the other in a backwards direction. LSTM helps preserves information from inputs that has already passed through it using the hidden state. Unidirectional LSTM only preserves information of the past because the only inputs it has seen are from the past. Using bidirectional will run your inputs in two ways, one from past to future and one from future to past and what differs this approach from unidirectional is that in the LSTM that runs backwards you preserve information from the future and using the two hidden states combined you are able in any point in time to preserve information from both past and future.

Global Vector Representation (GloVe):

GloVe is a model for distributed word representation. The model is an unsupervised learning algorithm for obtaining vector representations for words. This is achieved by mapping words into a meaningful space where the distance between words is related to semantic similarity.

Dataset:

The csv file contains two columns, Text and Emotions. The Emotions column has various categories ranging from happiness to sadness to love and fear. The text mood corresponds to the emotions. There are total of 21,405 text rows with corresponding emotions portrayed.

Text	Emotion
i didnt feel humiliated	sadness
i can go from feeling so hopeless to so damned hopeful just from being around someone who cares and is awake	sadness
im grabbing a minute to post i feel greedy wrong	anger
i am ever feeling nostalgic about the fireplace i will know that it is still on the property	love
i am feeling grouchy	anger
ive been feeling a little burdened lately wasnt sure why that was	sadness
ive been taking or milligrams or times recommended amount and ive fallen asleep a lot faster but i also feel like so funny	surprise
i feel as confused about life as a teenager or as jaded as a year old man	fear
i have been with petronas for years i feel that petronas has performed well and made a huge profit	happy
i feel romantic too	love
i feel like i have to make the suffering i m seeing mean something	sadness
i do feel that running is a divine experience and that i can expect to have some type of spiritual encounter	happy
i think it s the easiest time of year to feel dissatisfied	anger
i feel low energy i m just thirsty	sadness
i have immense sympathy with the general point but as a possible proto writer trying to find time to write in the corners of life and with no sign of an agent let alone a publishing contract this feels a little precious	happy
i do not feel reassured anxiety is on each side	happy
i didnt really feel that embarrassed	sadness
i feel pretty pathetic most of the time	sadness
i started feeling sentimental about dolls i had as a child and so began a collection of vintage barbie dolls from the sixties	sadness
i now feel compromised and skeptical of the value of every unit of work i put in	fear
i feel irritated and rejected without anyone doing anything or saying anything	anger
i am feeling completely overwhelmed i have two strategies that help me to feel grounded pour my heart out in my journal in the form of a letter to god and then end with a list of five things i am most grateful for	fear
i have the feeling she was amused and delighted	happy
i was able to help chai lifeline with your support and encouragement is a great feeling and i am so glad you were able to help me	happy
i already feel like i fucked up though because i dont usually eat at all in the morning	anger
i still love my so and wish the best for him i can no longer tolerate the effect that bm has on our lives and the fact that is has turned my so into a bitter angry person who is not always particularly kind to the people around him when	sadness
i feel so inhibited in someone elses kitchen like im painting on someone elses picture	sadness
i become overwhelmed and feel defeated	sadness

Module Wise Explanation:

1. **Importing the appropriate libraries:**
    - Tensorflow – For building the keras model
    - Pandas – reading the data
    - Numpy – process the input for training
    - Nltk – to process the data and remove stopwords
    - Sklearn – Calculate results and split input-output
    - Seaborn and matplotlib – for visualization of result
2. **Read data:**

    The data is read from the CSV file to be used for the training and prediction. In the case of GloVe, the text data containing the vector representation (in our case 100 dimension) is read.
3. **Process data:**

    Dropping the columns with null values and resetting the index. Using NLTK, pre-processing the text by getting rid of special characters, numbers, converting the tweet to lowercase, stemming the words, removing stop-words and one-hot encoding the sentences.
4. **Embedding layer:**

    In the case of GloVe, we try to identify the GloVe pretrained vectors of words in our vocabulary from the text file and assign the missing vectors to a randomly created vector with values close to the mean and standard deviation calculated from all the vectors in the file. We choose the vocabulary size as 10000 and the feature vector size as 100. So, we consider only maximum of 10000 words irrespective of the number of words in our vocabulary.

In the case of not using GloVe, we skip this part and randomly give weights to the embedding layer.

5. **Prepare input-output:**

   We first convert the sentences as vectors with one hot representation and pad them with the appropriate length for uniformity in input data. We then process the data into numpy array for providing it as input and label encode the target output. Sklearn is used to split the data into train, test and validation set.

6. **Create and train model:**

   We then create a model with an embedding layer, bidirectional lstm, dense layer and followed by a softmax dense layer for classifying the output. We train the model with a set of parameters and use it for prediction.

Parameter	**Value**
Loss	Sparse categorical cross entropy
Optimizer	Adam
Learning rate	0.01
Batch size	64
Epochs	40
Padded input	35

7. **Analysis and visualization of result:**

   The result is analysed using plotting the training accuracy and loss vs epochs graph. The accuracy score is calculated with classification report which contains information regarding the precision, recall and f1-score. Then the confusion matrix is plotted for understanding the test data prediction.

8. **Using own example:**

   Raw input is processed to check if the model can handle the text and predict accurately for unseen data.

Code:

Importing the appropriate libraries:

```
import numpy as np
import pandas as pd
import tensorflow as tf
import nltk
import seaborn as sns
import re
import matplotlib.pyplot as plt

from tensorflow.keras.layers import Embedding, Dense, LSTM, Dropout, Bidirectional
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import one_hot, Tokenizer
from tensorflow.keras.callbacks import ModelCheckpoint
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

Read data:

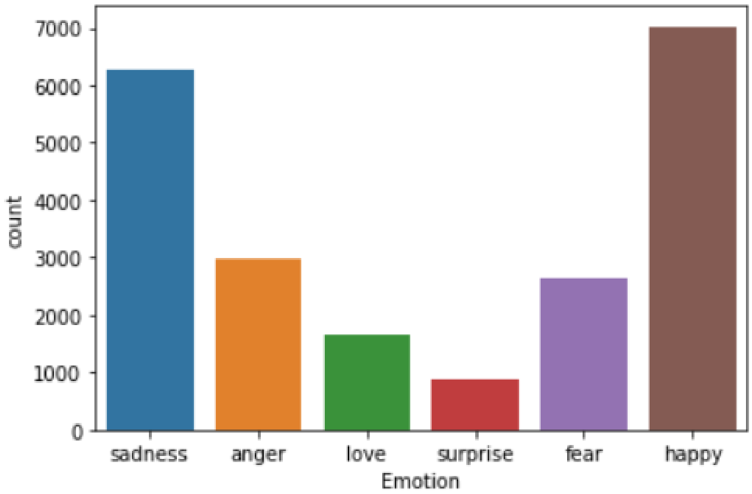
```
df=pd.read_csv('../input/emotions-in-text/Emotion_final.csv') #Text data
EMBEDDING_FILE= f'../input/glove6b100dtxt/glove.6B.100d.txt' #GloVe file path
df.head()
```

	Text	Emotion
0	i didnt feel humiliated	sadness
1	i can go from feeling so hopeless to so damned...	sadness
2	im grabbing a minute to post i feel greedy wrong	anger
3	i am ever feeling nostalgic about the fireplac...	love
4	i am feeling grouchy	anger

```
#Target Classes
sns.countplot(df['Emotion'])
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/_decorators.py:4
ata`, and passing other arguments without an explicit keyword w
  FutureWarning
```

```
<AxesSubplot:xlabel='Emotion', ylabel='count'>
```



Process data:

```
df=df.dropna() #Drop columns with NA values
X=df.drop('Emotion',axis=1) #Input
y=df['Emotion'] #Output
```

```
messages=X.copy()
messages.reset_index(inplace=True) #Drop NA may cause inconsistency in index
```

```
nltk.download('stopwords')
ps = PorterStemmer()
corpus = []
for i in range(0, len(messages)):
    review = re.sub('[^a-zA-Z]', ' ', messages['Text'][i]) #Remove Special Characters
    review = review.lower() #Lower case
    review = review.split()
    review = [ps.stem(word) for word in review if not word in stopwords.words('english')] #Remove stopwords
    review = ' '.join(review)
    corpus.append(review)
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
corpus[:10]
```

```
['didnt feel humili',
 'go feel hopeless damn hope around someon care awak',
 'im grab minut post feel greedi wrong',
 'ever feel nostalg fireplac know still properti',
 'feel grouchi',
 'ive feel littl burden late wasnt sure',
 'ive take milligram time recommend amount ive fallen asleep lot faster also feel like funni',
 'feel confus life teenag jade year old man',
 'petrona year feel petrona perform well made huge profit',
 'feel romant']
```

Embedding layer:

```
#Creating the dictionary with word as key and pretrained-value array as value
def get_coefs(word,*arr): return word, np.asarray(arr, dtype='float32')
embeddings_index = dict(get_coefs(*o.strip().split()) for o in open(EMBEDDING_FILE))

#Calculate mean and std for the pretrained weights
all_embs = np.stack(embeddings_index.values())
emb_mean,emb_std = all_embs.mean(), all_embs.std()
print(emb_mean,emb_std)
```

```
/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3357: FutureWarning: a
terables such as generators is deprecated as of NumPy 1.16 and will raise an error in the futu
  if (await self.run_code(code, result,  async_=asy)):
0.004451992 0.4081574
```

```
voc_size=10000 # Vocabulary size
embed_size=100 #word vector size

tokenizer = Tokenizer(num_words=voc_size)
tokenizer.fit_on_texts(list(corpus))
word_index = tokenizer.word_index #Total words in the corpus
nb_words = min(voc_size, len(word_index))

#Initialize weight matrix for embedding layer
embedding_matrix = np.random.normal(emb_mean, emb_std, (nb_words, embed_size))

for word, i in word_index.items():
    if i >= voc_size: continue #Skip the words if vocab size is reached
    embedding_vector = embeddings_index.get(word) #Extract the pretrained values from GloVe
    if embedding_vector is not None: embedding_matrix[i] = embedding_vector
```

```
#Contains the pretrained GloVe weights for the words
len(embedding_matrix)
```

```
10000
```

Prepare input-output:

```
#One hot representation for input
onehot_repr=[one_hot(words,voc_size)for words in corpus]

#Finding max words
l = 0
for x in corpus:
    l = max(l,len(x.split(' ')))

#Padding the sequences for input
sent_length= l
embedded_docs=pad_sequences(onehot_repr,padding='pre',maxlen=sent_length)
print(embedded_docs)
```

```
[[   0    0    0 ... 5427   99 1689]
 [   0    0    0 ... 4782 1495 8980]
 [   0    0    0 ...   99 3316 3628]
 ...
 [   0    0    0 ... 5535 7908 8318]
 [   0    0    0 ... 6279 8318 7579]
 [   0    0    0 ...    0 8318 9956]]
```

```
#Encoding the target outputs to integers
label_encoder = preprocessing.LabelEncoder()

X_final=np.array(embedded_docs) #input to array
y = label_encoder.fit_transform(y)
y_final=np.array(y)
print(y_final)
```

```
[4 4 0 ... 1 1 1]
```

```
X_final.shape,y_final.shape
```

```
((21459, 35), (21459,))
```

```
#Train-Test split
X_train, X_test, y_train, y_test = train_test_split(X_final, y_final,
                                                    test_size=0.2, random_state=42)
#Train-Validation split
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
                                                  test_size=0.1, random_state=21)
```

Create and train model:

```
# Creating model
model=Sequential()
model.add(Embedding(voc_size, embed_size, weights=[embedding_matrix]))
model.add(Dropout(0.3))
model.add(Bidirectional(LSTM(64)))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu',kernel_regularizer=tf.keras.regularizers.l1(0.01))) #L1 regularization
model.add(Dropout(0.3))
model.add(Dense(6,activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy',optimizer= tf.keras.optimizers.Adam(learning_rate=0.001),
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 100)         1000000
_________________________________________________________________
dropout (Dropout)            (None, None, 100)         0
_________________________________________________________________
bidirectional (Bidirectional (None, 128)               84480
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 64)                8256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 390
=================================================================
Total params: 1,093,126
Trainable params: 1,093,126
Non-trainable params: 0
_________________________________________________________________
```

```
model_save = ModelCheckpoint('weights.h5', save_best_only = True, save_weights_only = True, monitor = 'val_loss',
                             mode = 'min', verbose = 1)
history = model.fit(X_train,y_train,validation_data=(X_val,y_val),epochs=40,batch_size=64,callbacks = [model_save])
```

```
Epoch 1/40
242/242 [==============================] - 9s 20ms/step - loss: 5.6914 - accuracy: 0.3203 - val_loss: 1.6684 - val_accuracy: 0.3151

Epoch 00001: val_loss improved from inf to 1.66838, saving model to weights.h5
Epoch 2/40
242/242 [==============================] - 4s 17ms/step - loss: 1.6392 - accuracy: 0.3198 - val_loss: 1.6282 - val_accuracy: 0.3151

Epoch 00002: val_loss improved from 1.66838 to 1.62817, saving model to weights.h5
Epoch 3/40
242/242 [==============================] - 4s 15ms/step - loss: 1.6089 - accuracy: 0.3263 - val_loss: 1.5664 - val_accuracy: 0.3564

Epoch 00003: val_loss improved from 1.62817 to 1.56639, saving model to weights.h5
Epoch 4/40
242/242 [==============================] - 4s 16ms/step - loss: 1.5008 - accuracy: 0.3679 - val_loss: 1.3715 - val_accuracy: 0.4228

Epoch 00004: val_loss improved from 1.56639 to 1.37153, saving model to weights.h5
Epoch 5/40
242/242 [==============================] - 4s 15ms/step - loss: 1.3491 - accuracy: 0.4604 - val_loss: 1.2622 - val_accuracy: 0.5131

Epoch 00005: val_loss improved from 1.37153 to 1.26223, saving model to weights.h5
Epoch 6/40
242/242 [==============================] - 4s 15ms/step - loss: 1.2095 - accuracy: 0.5613 - val_loss: 1.0985 - val_accuracy: 0.6243

Epoch 00006: val_loss improved from 1.26223 to 1.09848, saving model to weights.h5
Epoch 7/40
242/242 [==============================] - 4s 16ms/step - loss: 1.0562 - accuracy: 0.6488 - val_loss: 0.9872 - val_accuracy: 0.6570

Epoch 00007: val_loss improved from 1.09848 to 0.98720, saving model to weights.h5
Epoch 8/40
242/242 [==============================] - 4s 15ms/step - loss: 0.9595 - accuracy: 0.6759 - val_loss: 0.9145 - val_accuracy: 0.6855
```

Analysis and visualization of result:

```
print(history.history.keys())
#  "Accuracy"
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# "Loss"
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
#Load the best weights
model.load_weights('weights.h5')
```

```
y_pred=model.predict_classes(X_test)
print(y_pred)
```

```
/opt/conda/lib/python3.7/site-packages/tensorflow/python/ke
tead:* `np.argmax(model.predict(x), axis=-1)`,   if your mo
if your model does binary classification   (e.g. if it uses
  warnings.warn('`model.predict_classes()` is deprecated an
[4 0 4 ... 0 0 2]
```

```
#Accuracy score
print(accuracy_score(y_test,y_pred))
```

```
#Classification report
print(classification_report(y_test, y_pred, digits=5))
```

```
#Confusion Matrix
print('Confusion Matrix')
print(sns.heatmap(confusion_matrix(y_test, y_pred),annot=True,fmt="d"))
```

Using own example:

```
#Mapping of target classes using label-encoder
le_name_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))
print(le_name_mapping)
```

```
{'anger': 0, 'fear': 1, 'happy': 2, 'love': 3, 'sadness': 4, 'surprise': 5}
```

```
#Example
def predict_emotion(stri):
    review = re.sub('[^a-zA-Z]', ' ', stri)
    review = review.lower()
    review = review.split()
    review = [ps.stem(word) for word in review if not word in stopwords.words('english')]
    review = ' '.join(review)
    onehot_repr = [one_hot(review,voc_size)]
    embed = pad_sequences(onehot_repr,padding='pre',maxlen=sent_length)
    predicti = model.predict(embed)
    return label_encoder.classes_[np.argmax(predicti)]
```

```
predict_emotion('I am very happy and joyful today')
```

```
'happy'
```

```
predict_emotion('He is an arrogant and rude person')
```

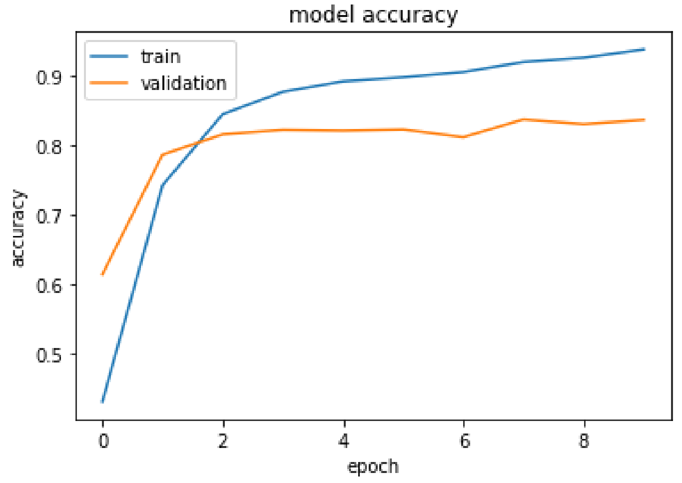
```
'anger'
```

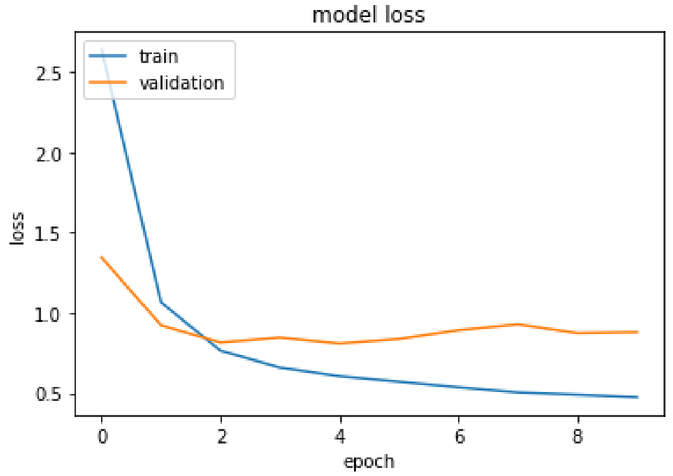
```
predict_emotion('The teacher is intimidating and scary')
```

```
'fear'
```

Result Analysis:

Embedding layer without GloVe:



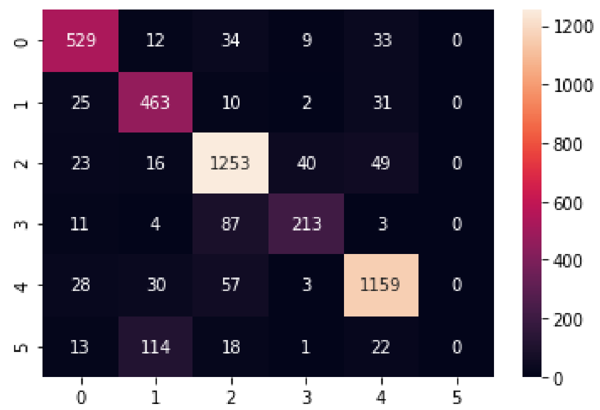



```
print(accuracy_score(y_test,y_pred))
```

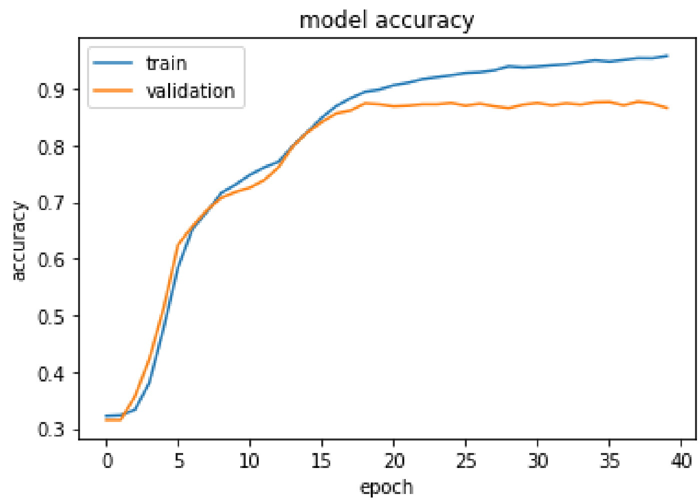
```
0.8427306616961789
```

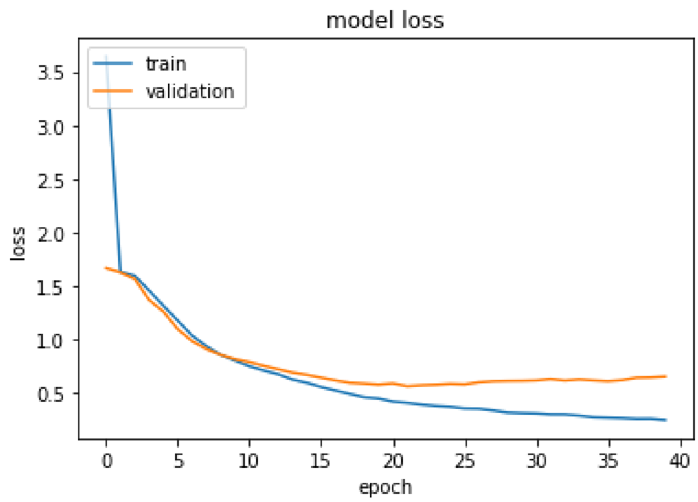
```
print(classification_report(y_test, y_pred, digits=5))
```

```
              precision    recall  f1-score   support

           0    0.84102   0.85737   0.84912       617
           1    0.72457   0.87194   0.79145       531
           2    0.85881   0.90731   0.88239      1381
           3    0.79478   0.66981   0.72696       318
           4    0.89360   0.90760   0.90054      1277
           5    0.00000   0.00000   0.00000       168

    accuracy                        0.84273      4292
   macro avg    0.68546   0.70234   0.69175      4292
weighted avg    0.81163   0.84273   0.82570      4292
```



Embedding layer with GloVe:





```
#Accuracy score
print(accuracy_score(y_test,y_pred))
```

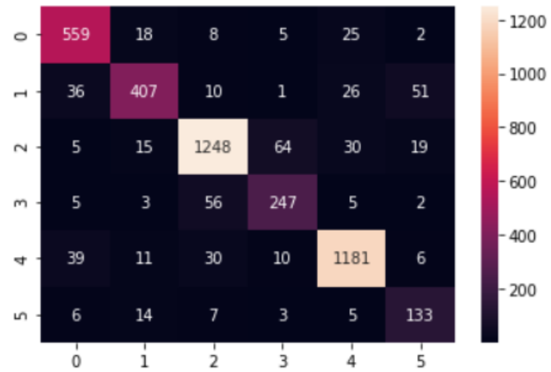
```
0.8795433364398881
```

```
#Classification report
print(classification_report(y_test, y_pred, digits=5))
```

```
              precision    recall  f1-score   support

           0    0.86000   0.90600   0.88240       617
           1    0.86966   0.76648   0.81481       531
           2    0.91832   0.90369   0.91095      1381
           3    0.74848   0.77673   0.76235       318
           4    0.92846   0.92482   0.92664      1277
           5    0.62441   0.79167   0.69816       168

    accuracy                        0.87954      4292
   macro avg    0.82489   0.84490   0.83255      4292
weighted avg    0.88285   0.87954   0.88028      4292
```

```
#Confusion Matrix
print('Confusion Matrix')
print(sns.heatmap(confusion_matrix(y_test, y_pred),annot=True,fmt="d"))
```

```
Confusion Matrix
AxesSubplot(0.125,0.125;0.62x0.755)
```



Comparison:

Model	Accuracy	Precision	Recall	F1-score
Without GloVe	84.27	81.16	84.27	82.57
With GloVe	87.95	88.28	87.95	88.02

Conclusion:

Emotion prediction from text using deep learning proves to be successful with a 6-way classification yielding 88% accuracy with significant amount of data in test. We can see the improvement in performance while we use GloVe pretrained layers.

References:

1. GloVe: https://nlp.stanford.edu/projects/glove/
2. https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010
3. Dataset: https://www.kaggle.com/ishantjuyal/emotions-in-text