IST 664 – Natural Language Processing

Under guidance of Prof. Lu Xiao

Twitter Emotion Recognition using RNN

Group 2:

Chaithanya Chikkannaswamy

Mars Sun

Nishitha Maniganahalli Venkatesh

Qinwei Huang

Table of Contents

1. Abstract

Twitter is an American microblogging and social networking service on which users post and interact using “tweets”. Twitter has gotten progressively popular with academics, as well as students, policymakers, legislators and overall population. It allows users to express their opinion, seek feedback about one’s work, etc. Each tweet a user tweets contains single or multiple emotions.

The goal of our project is to apply a deep learning approach to predict the emotions in various tweets. We used the BiLSTM model to predict the emotions. Finally, we generated a confusion matrix to compare the predicted emotions against the test emotions.

2. Introduction

Feelings are communicated in nuanced ways, which differs by group or individual encounters, information, and convictions. Hence, to understand emotions conveyed through texts, a robust system fit for capturing and demonstrating diverse etymological subtleties and wonders is required.

The amount of user created content on the web is growing quickly, because of the emergence of social networks, microblogging sites and other platforms that enable users to share their personal content. User generated content is abundant in emotions, feelings and opinions. These online expressions have been used to predict stock movie’s financial success., market fluctuations and book sales.

Emotion recognition is the way toward recognizing human emotions. Each individual has a different accuracy in identifying others’ emotions. Use of technology for recognizing emotions is an emerging research field. Various studies are being carried out in the field of text mining because of the simplicity in sourcing for information and the huge advantages its deliverable offers.

Data from various sources like Twitter have been often used for sentiment and emotion analysis. These provide greater insights about the users’ feelings and emotions towards a particular topic or in general.

3. Data Description

We are using the default English Twitter messages dataset from the HuggingFace Hub 🤗. The dataset has two fields i.e., text and label. The label contains six basic emotions: anger, fear, joy, love, sadness, and surprise. We have a total of 20000 data points.

The dataset is already divided into train, validation and test sets. The test set contains 16000 entries (80%), validation set contains 2000 entries (10%) and test set contains the remaining 2000 entries (10%).

Let us take a closer look into the quality of data. First, we need to check if there are any missing values. After examinations on these datasets, it is obvious to see that there are no missing values within this dataset.

```
label    0
text     0
dtype: int64
```

For each record in three datasets, they have already been labeled, and prepressed by applying lower case, and removing hashtags and URLs.

4. Exploratory Data Analysis

The plot below shows the length of tweets in our dataset. The average length of tweets is around 19, and median length is 17. It is obvious to see that this is a right-skewed distribution plot.

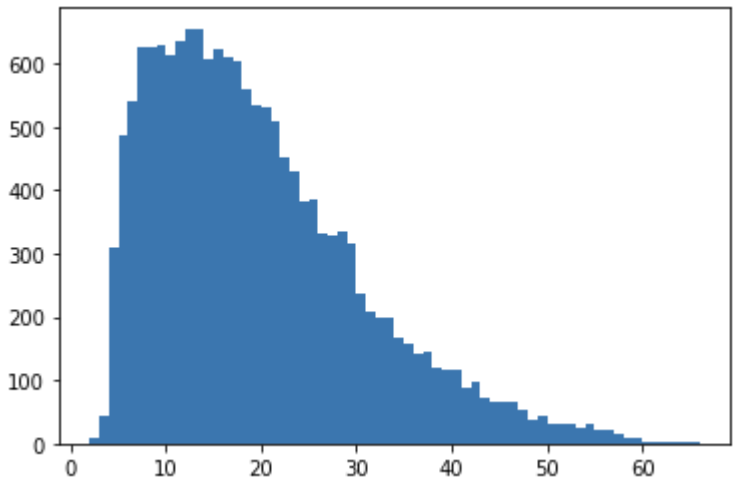

Figure 1. Length of Tweets

Then we want to check the number of tweets with different labels. From the following chart, we can find that joy tweets occupied the greatest number of tweets among the dataset which is over 5000. Then the second greatest amount of label is sadness tweets, which count around 4500. By comparison, the lowest number of emotions is surprise, which number is lower than 500. Overall, the number of emotion labels in this dataset are imbalanced. Therefore, as for surprise emotion, it may be hard to capture during the prediction.

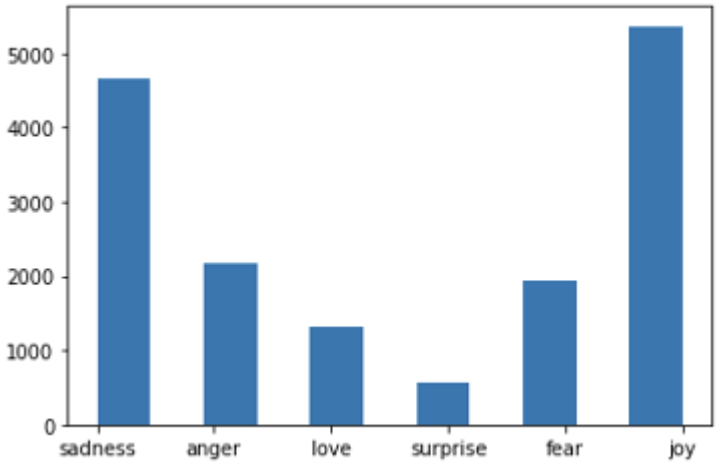

Figure 2. Frequency of labels

5. Data preprocessing

5.1 Tokenizing

First, we need to create a corpus of all the words that exist in the dataset, then give each unique word a unique corresponding token. We can set how many frequently used words to be tokenized, and less commonly used words are ignored.

The arguments are saying that we will only keep 10000 most frequent words to be tokenized, otherwise will be represented as <UNK>, and for this string will also have a unique number to represent.

```
In [9]: from tensorflow.keras.preprocessing.text import Tokenizer

In [10]: tokenizer = Tokenizer(num_words=10000, oov_token='<UNK>')
         tokenizer.fit_on_texts(tweets)

In [11]: tokenizer.texts_to_sequences([tweets[0]])
Out[11]: [[2, 139, 3, 679]]

In [12]: tweets[0]
Out[12]: 'i didnt feel humiliated'
```

5.2 Padding and truncating sequences

The input of the model requires fixed shape. Therefore, we need to set max-lengths argument to a number. The words of tweets exceeding this number will be chopped off at the end. For words of tweets within this number will be padded with zeros.

```
maxlen = 50
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
def get_sequences(tokenizer, tweets):
  sequences = tokenizer.texts_to_sequences(tweets)
  padded = pad_sequences(sequences, truncating='post', padding='post', maxlen=maxlen)
  return padded
```

```
padded_train_seq = get_sequences(tokenizer, tweets)
```

```
padded_train_seq[0]
```

```
array([  2, 139,   3, 679,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0], dtype=int32)
```

5.3 Preparing labels

Preparing labels is the last step of data preprocessing. Before this we have already generated the sequence, or tweets vector. Then we also need to embed their label which needs to be numeric. Therefore, we transform the labels into int values.

```
In [20]: class_to_index = dict((c, i) for i, c in enumerate(classes))
         index_to_class = dict((v, k) for k, v in class_to_index.items())

In [21]: class_to_index
Out[21]: {'anger': 0, 'fear': 5, 'joy': 1, 'love': 4, 'sadness': 2, 'surprise': 3}

In [22]: index_to_class
Out[22]: {0: 'anger', 1: 'joy', 2: 'sadness', 3: 'surprise', 4: 'love', 5: 'fear'}

In [23]: names_to_ids = lambda labels: np.array([class_to_index.get(x) for x in labels])

In [24]: train_labels = names_to_ids(labels)
         print(train_labels[0])
         2
```

5.4 Prepare training testing sets

5.4.1 Creating our train, validation, and test sets

In this project, we plan to utilize the trained model to obtain the internal emotion of actual tweets. The aim of our project is to train a model which can distinguish the different emotions in real tweets as precise as possible. Or, from the opposite aspect, we are trying to build a classifier that generates less wrong prediction results on real tweets. We call the error of the classifier, when applying to real situations, a generalization error, and then we should try to lower the generalization error of our trained model.

Since we are going to use a neural network model to train a classifier which could predict the emotion included in the input sentence, we need to split the dataset we got into 3 subsets rather than 2 simple training and testing sets used by traditional machine learning models.

1. Training set: this set would be used as the basic training data inputted in the network to let the model learn the features of each sentence.

2. Validation set: this set is usually used as the model selection. The essence of model selection is to evaluate the generalization error of the expected model, and then select the model with the smallest generalization error to train the sample to prevent the model from overfitting. In our project, the validation set is mainly used to determine the selection of Hyperparameter.

3. Testing set: this set works more like the test set we are familiar with. It is used to give an evaluation of our current model like the accuracy of the classifier.

The dataset we used (default English Twitter messages dataset from the HuggingFace Hub) has already been divided into 3 sets. There are totally 20,000 sentences tagged with emotions in the dataset.

Name of set	**Training set**	**Validation set**	**Testing set**
Number of data	16,000	2,000	2,000

5.4.2 Potential improvement

In the dataset we used, it has already split the data into fixed 3 sets we mentioned above. Therefore, we think that if we can mix the 3 sets up and divide them by ourselves. Experimental results may be more general than previous one. Because the fixed dataset might have the problem of not being random collected. So, in order to fix this problem, we can shuffle the dataset. Then, we can divide the whole set into 10 equal subsets. Next, in each subset, we can choose the first 80% of data as training data, 10% of next data as validation set, rest of them as testing set. In this way, we might get the result with more universality.

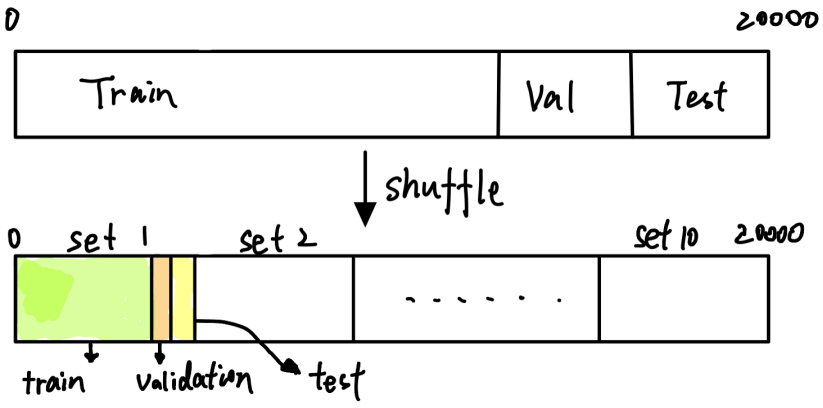

Figure 3. Split subsets

6. Implementation

6.1 Recurrent Neural Network (RNN)

Recurrent Neural Network is one of the most popular neural networks which is especially suitable for sequence models like dealing with context. Thus, RNN is widely used in the NLP area.

Like Convolutional Neural Network, most algorithms would require the input and output to be corresponding. However, in some scenarios, one input would not be enough. We need a flow of information to generate the output. In this situation, RNN becomes useful.

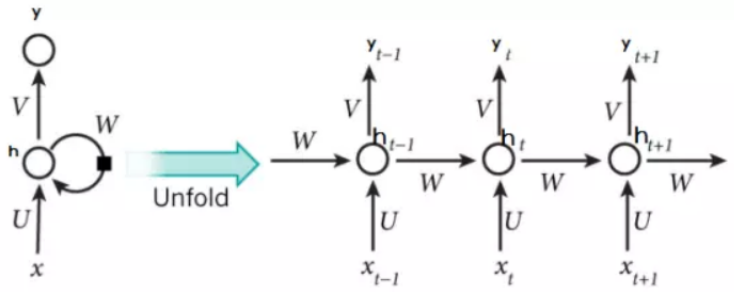

Figure 4. RNN architecture

Most significant difference between RNN and traditional neural networks is that RNN will transform the former output to the current hidden layer to train with the new input together. In figure 4, every x_t is an input word and after the operation of h_t (or cell), we can get the output y_t. Then, in the next hidden layer, it will utilize the last output y_t in the current hidden layer h_{t+1}. So, the information contained in the previous input can be passed to the next input and cause the influence. Finally, we can input a sequence of words into the model to predict the possible last word.

However, RNN has its own drawback. When RNN transforms the previous information to the next stage. It will always forget part of the more recent information and replace that part with new information. This action would lead to a result of the latest information and would cause larger influence on later stages. Also, this structure determines that RNN cannot take long context for it would lose the information at the beginning. In order to eliminate this kind of negative influence caused by RNN, we should make some changes on RNN.

6.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory network is a branch of RNN which overcomes the drawback we mentioned above. It has the ability to learn long-term dependence.

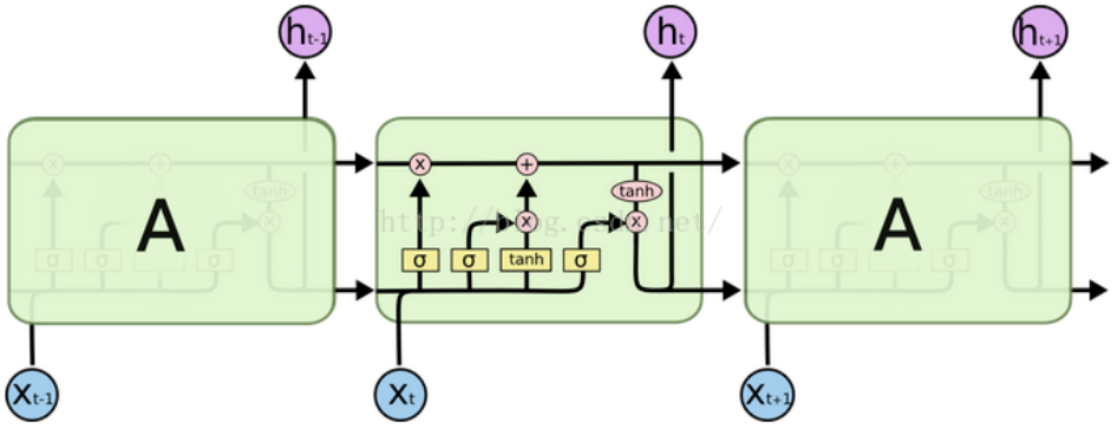

Figure 5. LSTM architecture

In figure 5, green squares mean a cell, pink dots mean the gate (pointwise operation) and the yellow squares mean neural network layer. Unlike RNN, in each cell rather than one, LSTM has 4 neural network layers. The key idea of LSTM is the cell state which is the black line in the upper area of the cell in figure 5. The cell state can be understood as a message transformer. It will pass all the cells in the model with just a few linear interactions with the messages. Selected messages that are chosen to be passed to the later part can freely run through in a constant way. Then, LSTM will use the gate structure to select whether to increase or decrease the amount of information for cell state. Every gate makes up with a sigmoid or dot product operation. These two operators will determine how much information could be passed to the next cell. To be specific, when the gate value is 0, no information could be passed. When the gate value is 1, all information should be transformed.

6.3 Bi-LSTM

LSTM solves the problem of long-term memory. Nevertheless, it only utilizes the information coming from the front of context sequence. The information contained in the back of the context sequence is not considered. For more accurate prediction and information collecting, we decided to use Bi-LSTM model in our project to build the neural network model. Bi-LSTM takes the inputs from both forward direction and backward direction.

In this way, Bi-LSTM provides a more effective model to extract the hidden information in the whole sentence. By using this model, we can get a more precise prediction of which emotion is included in the input sentence.

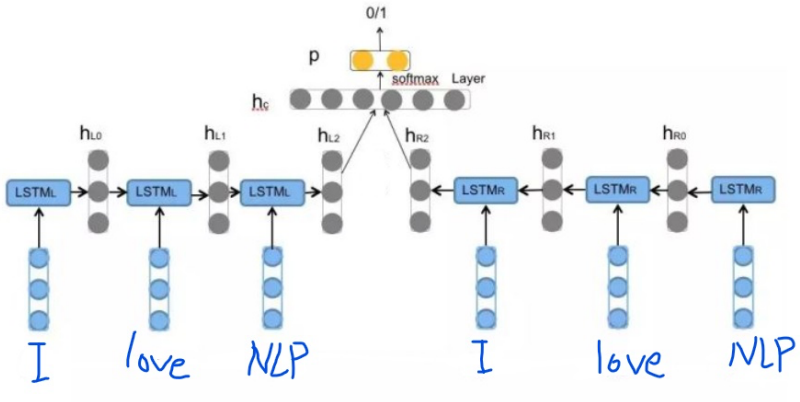

Figure 6. Bi-LSTM

6.4 Using Tensorflow to build network

We utilized the Tensorflow package to build and train our model. Our model consists of 4 layers:

1. Embedding layer: input dimension: 10,000, output dimension: (16, 50)

2. First Bidirectional LSTM layer: 20 cells with return sequence set to True. (Return an output in every timestamp or every cell)

3. Second Bidirectional LSTM layer: 20 cells

4. Output layer: 6 emotion classes, using activation function softmax.

Our neural network contains 175,926 trainable parameters.

```
In [26]: model = tf.keras.models.Sequential([
             tf.keras.layers.Embedding(10000, 16, input_length=maxlen),
             tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(20, return_sequences=True)),
             tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(20)),
             tf.keras.layers.Dense(6, activation='softmax')
         ])
         model.compile(

             loss='sparse_categorical_crossentropy',
             optimizer='adam',
             metrics=['accuracy']
         )

In [27]: model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 50, 16)            160000
_________________________________________________________________
bidirectional_2 (Bidirection (None, 50, 40)            5920
_________________________________________________________________
bidirectional_3 (Bidirection (None, 40)                9760
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 246
=================================================================
Total params: 175,926
Trainable params: 175,926
Non-trainable params: 0
_________________________________________________________________
```

Figure 7. Architecture of our neural network and training

Then, we can start training our model with training and validation sets.

```
h = model.fit(
    padded_train_seq, train_labels,
    validation_data=(val_seq, val_labels),
    epochs=20,
    callbacks=[
                tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=2)
    ]
)
```

```
Epoch 1/20
500/500 [==============================] - 8s 16ms/step - loss: 1.2523 - accuracy: 0.5146 - val_loss: 0.7588 - val_accuracy: 0.7525
Epoch 2/20
500/500 [==============================] - 7s 14ms/step - loss: 0.5273 - accuracy: 0.8192 - val_loss: 0.5209 - val_accuracy: 0.8225
Epoch 3/20
500/500 [==============================] - 7s 14ms/step - loss: 0.3163 - accuracy: 0.8994 - val_loss: 0.4604 - val_accuracy: 0.8590
Epoch 4/20
500/500 [==============================] - 7s 14ms/step - loss: 0.2143 - accuracy: 0.9344 - val_loss: 0.4074 - val_accuracy: 0.8740
Epoch 5/20
500/500 [==============================] - 7s 15ms/step - loss: 0.1545 - accuracy: 0.9515 - val_loss: 0.5065 - val_accuracy: 0.8650
Epoch 6/20
500/500 [==============================] - 7s 15ms/step - loss: 0.1322 - accuracy: 0.9581 - val_loss: 0.3652 - val_accuracy: 0.8845
Epoch 7/20
500/500 [==============================] - 7s 14ms/step - loss: 0.0985 - accuracy: 0.9676 - val_loss: 0.3308 - val_accuracy: 0.8965
Epoch 8/20
500/500 [==============================] - 7s 15ms/step - loss: 0.0791 - accuracy: 0.9736 - val_loss: 0.3988 - val_accuracy: 0.8905
Epoch 9/20
500/500 [==============================] - 7s 14ms/step - loss: 0.0712 - accuracy: 0.9766 - val_loss: 0.3707 - val_accuracy: 0.9005
Epoch 10/20
500/500 [==============================] - 7s 14ms/step - loss: 0.0634 - accuracy: 0.9786 - val_loss: 0.4216 - val_accuracy: 0.8950
Epoch 11/20
500/500 [==============================] - 7s 14ms/step - loss: 0.0563 - accuracy: 0.9814 - val_loss: 0.3674 - val_accuracy: 0.9015
Epoch 12/20
500/500 [==============================] - 7s 14ms/step - loss: 0.0474 - accuracy: 0.9848 - val_loss: 0.4575 - val_accuracy: 0.8920
Epoch 13/20
500/500 [==============================] - 7s 14ms/step - loss: 0.0498 - accuracy: 0.9840 - val_loss: 0.4050 - val_accuracy: 0.8955
```

Figure 8. Training model

7. Result

The training and the validation data set is represented through a graph to analyze the accuracy and loss of the epochs using the matplotlib library in python. The below is the graphical representation of the same.

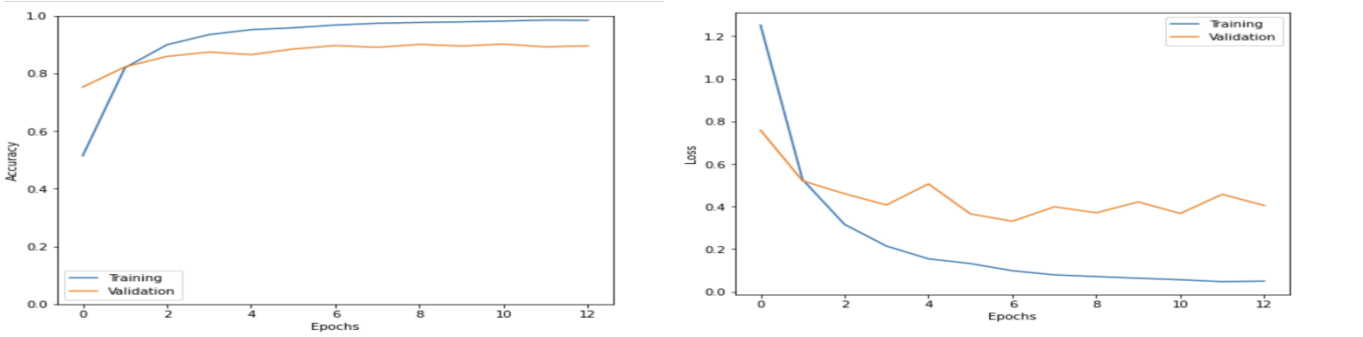

Figure 9. (1) Graph representing the accuracy of training and validation dataset, (2) Graph representing the loss of training and validation dataset

The RNN model used in our project for the text-based emotion recognition of the tweets resulted in **89% accuracy** and **40% loss.**

Below is the attached screenshot of the accuracy and loss:

```
test_tweets, test_labels = get_tweet(test)
test_seq = get_sequences(tokenizer, test_tweets)
test_labels = names_to_ids(test_labels)
```

```
_=model.evaluate(test_seq, test_labels)
```

```
63/63 [==============================] - 0s 6ms/step - loss: 0.4029 - accuracy: 0.8985
```

We were able to successfully train our data and predict the emotions of the tweets. Below screenshot is a sample of the tweet for which we have predicted the emotion. Emotion here is the truth value i.e., the emotion given in the data set and the predicted emotion is the result of the model.

```
i = random.randint(0, len(test_labels) - 1)

print('Sequence:', test_tweets[i])
print('Emotion:', index_to_class[test_labels[i]])

p = model.predict(np.expand_dims(test_seq[i], axis=0))[0]
pred_class = index_to_class[np.argmax(p).astype('uint8')]

print('Predicted Emotion:', pred_class)
```

```
Sequence: i can t help feeling lucky little do i know
Emotion: joy
Predicted Emotion: joy
```

Figure 10. Sample output of the project

To better understand the result of our prediction model, we decided to plot the confusion matrix. Confusion matrix often describes the performance of a classifier or a classification model on a given data set for which the truth values are known. It basically compares the predicted output of our machine learning model with the actual target values. In our project, we have used the sklearn library to plot the confusion matrix.

In the below diagram, x-axis represents the truth value i.e., the emotions given in the data set and y-axis represents the predicted emotions i.e., the result of our model. Classes are the set of labels that contain 6 different labels as mentioned in the dataset i.e., anger, joy, sadness, surprise, love and fear.

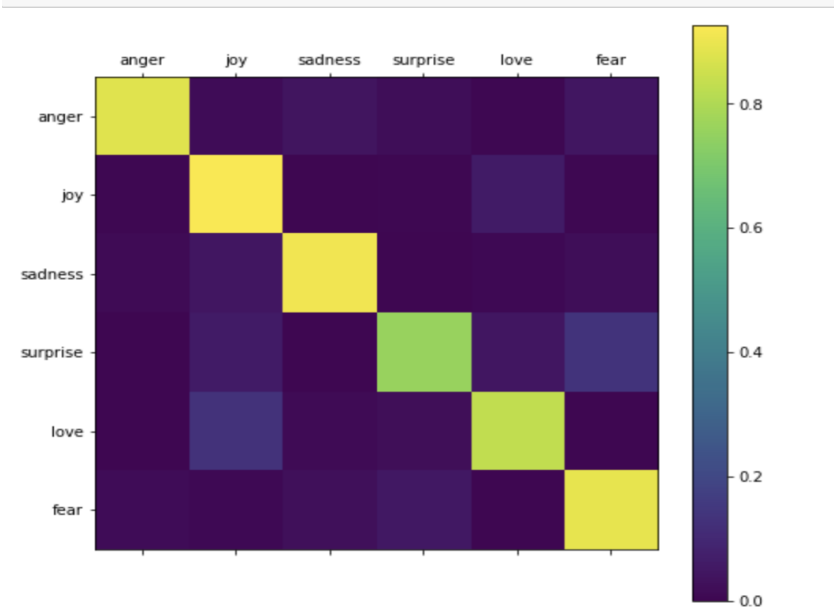

Figure 11. Confusion matrix representing the performance evaluation of model

8. Conclusion

The usage of BiLSTM model for training the data set gave us better results as BiLSTM model contains 2 LSTM, one taking the input in forward and other in backward direction. This improved the performance of our sequence classification model. The Recurrent Neural Network is used for modeling the sequence data. Here the output of the previous step is fed as input to the next step which helped for the better predictions for text-based emotion recognition resulting in more accurate predictions and we were able to get high prediction accuracy.

9. Future Work

After completing and analyzing the text-based emotion recognition using the RNN model, we came up with ideas we can still implement in our project as stated below:

1. To learn the performance difference between different NN models, we can implement a CNN to replace the LSTM model. Then we can compare the performance of these two models.
2. We can mix the given test, validation and training data sets to create these sets by ourselves by changing the percentage of these three sets to see the difference of model's performance.

10. References

[1] Elvis Saravia, Hsien- Chi Toby Liu, Yen-Hao Huang, Junlin Wu, Yi-Shin Chen, “Contextualized Affect Representations for Emotion Recognition”

[2] J. Li and L. Xiao, "Emotions in online debates: Tales from 4Forums and ConvinceMe," Proceedings of the Association for Information Science and Technology, vol. 57, (1), pp. n/a, 2020