Shreyas Kapoor | Lab Assignment 3

Impact of RNN Architecture

Method

The first step was to import all the necessary libraries. For this experiment, I chose the "SalehAhmad/Spam-Ham" dataset from huggingface library. The data consisted of 5566 items with one column being the "sms_text" and other being the label classified as "ham" (no spam) or "spam". After importing the data, I changed the string ham/spam to integer 0/1. After that, I split the data into 70% training data and 30% test data. The next step was to tokenize the sentences and do the padding of the input sentences. For that, I used the Tokenizer and pad_sequences libraries from keras. After this, out data was ready to be fed into the models

I created 3 models, using SimpleRNN, LSTM and GRU libraries. The hyperparameters what were consistent across the models were

Table 1: Hyper parameters

No of words used as features	10000
Max length of sentences	500
No of epochs	20
Output dimension from Embedding layer	64
Output dimension from SimpleRNN, LSTM, GRU layers:	64
Optimizer	Rmsprop
Loss function	Binary_crossentropy
Batch size while training	60
Activation function of Dense Layer	sigmoid

Model 1: SimpleRNN

The first step was to add the sequential layer. After that an embedding layer was added with max_features=1000 and output dimension as 64. After that, a SimpleRNN layer was added with output dimension = 64. After that, a dense layer was added with activation function as sigmoid.
The model had 648,321 parameters.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, None, 64)          640000

 simple_rnn_1 (SimpleRNN)    (None, 64)                8256

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 648,321
Trainable params: 648,321
Non-trainable params: 0
_________________________________________________________________
```

Model 2: LSTM

The first step was to add the sequential layer. After that an embedding layer was added with max_features=1000 and output dimension as 64. After that, a LSTM layer was added with output dimension = 64. After that, a dense layer was added with activation function as sigmoid.
The model had 673,089 parameters.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, None, 64)          640000

 lstm (LSTM)                 (None, 64)                33024

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 673,089
Trainable params: 673,089
Non-trainable params: 0
_________________________________________________________________
```

Model 3: GRU

The first step was to add the sequential layer. After that an embedding layer was added with max_features=1000 and output dimension as 64. After that, a GRU layer was added with output dimension = 64. After that, a dense layer was added with activation function as sigmoid.
The model had 665,025 parameters.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, None, 64)          640000

 gru (GRU)                   (None, 64)                24960

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 665,025
Trainable params: 665,025
Non-trainable params: 0
_________________________________________________________________
```

After all the models were trained, test sentences were used to predict the models and precision and recall was calculated.

The next part of this experiment was to break the test sentences into three parts according to the length of the sentences. For this, I first calculated the length of the longest sentence. This was 96. After that, I created three lists for small_input, medium_input and large_input. I ran a loop and appended the small_input list if the length of the sentence was less than 32, appended medium_input if length if the sentence was between 32 and 64 and appended the large_input if the length of the sentence was larger than 64. Of the 1670 sentences what were there 1586 were of small_input, 77 were of medium input, and 7 were of large input. Finally, I tested these sentences an all the three models and calculated the precision and recall.

Results

Table 2: Precision of the models

SimpleRNN	50.73
LSTM	50.78
GRU	51.22

Table 3: Recall of the models

SimpleRNN	50.95
LSTM	51.02
GRU	51.45

Table 4: Precision of small input

SimpleRNN	49.31
LSTM	50.09
GRU	50.09

Table 5: Recall of small input

SimpleRNN	49.11
LSTM	50.12
GRU	50.11

Table 6: Precision of medium input

SimpleRNN	45.38
LSTM	44.53
GRU	73.87

Table 7: Recall of medium input

SimpleRNN	45.59
LSTM	46.57
GRU	55.43

Table 8: Precision of large input

SimpleRNN	42.86
LSTM	42.86
GRU	42.86

Table 9: Recall of large input

SimpleRNN	0.5
LSTM	0.5
GRU	0.5

Analysis

Did a certain network perform better?

The precision and recall for all my models were almost similar. One reason for this can be that my dataset was heavy on non spam messages. I did analysis on my data and found out that just 5% of the data was spam message. However, even though similar, precision and recall were highest for GRU, then for LSTM and lowest for SimpleRNN. This was expected as GRU and LSTM are better models than the simpleRNN because they overcome the shortcoming of the simpleRNN of not being able to remember the earlier outputs.

Impact of small, medium and large input

Bothe the precision and recall decreases as the size of the input is increased. This is consistent across all the models. The reason for this can be that the number if samples for small input is much larger than the number of samples for large input (1586 vs 7). Since the original dataset was already biased on non spam messages, all the long messages were also non spam. Because of this, the precision and recall decreased for the long input messages.

Impact of Pretrained Word Embedding

Method

The first step was to import all the necessary libraries. For this experiment, I chose the "SalehAhmad/Spam-Ham" dataset from huggingface library. The data consisted of 5566 items with one column being the "sms_text" and other being the label classified as "ham" (no spam) or "spam". After importing the data, I changed the string ham/spam to integer 0/1. After that, I split the data into 70% training data and 30% test data. The next step was to create a text vectorizer to index our vocabulary and convert the string training samples into integer numpy array. After that, I downloaded the GloVe word embedding and created the embedding layer and embedding matrix. After that, I created the LSTM and models using the GloVe embedding layer for model 1 and Word2Vec for model 2.

The hyperparameters that were consistent across the models were

Table 5: Hyperparameters common across models

Output Dimension of LSTM and GRU layers	20
Output Dimension of Dense layer	2
Activation function in Dense layer	sigmoid
No of Epochs	20
Batch size during training	128
Optimizer	Adam
Loss function	Sparse_categorical_crossentropy

Model 1: LSTM with GloVe

The first step was to define the input shape to be fed into the model. Next was to create the embedding sequence from the embedding layer. Next, a bidirectional layer of LSTM was added with output dimension as 20. Another bidirectional layer of LSTM was added after that with same output dimension. Finally, a dense layer was added with 2 output units and sigmoid as activation function. This model had 809,402 parameters of which 29, 202 were trainable.

```
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, None)]            0

 embedding (Embedding)       (None, None, 100)         780200

 bidirectional_2 (Bidirectio  (None, None, 40)         19360
 nal)

 bidirectional_3 (Bidirectio  (None, 40)               9760
 nal)

 dense_1 (Dense)             (None, 2)                 82

=================================================================
Total params: 809,402
Trainable params: 29,202
Non-trainable params: 780,200
```

Model 2: LSTM with word2vec

The first step was to define the input shape to be fed into the model. Next was to create the embedding sequence from the embedding layer. Next, a bidirectional layer of LSTM was added with output dimension as 20. Another bidirectional layer of GRU was added after that with same output dimension. Finally, a dense layer was added with 2 output units and sigmoid as activation function. This model had 809,402 parameters of which 29,202 were trainable.

```
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, None)]            0

 embedding (Embedding)       (None, None, 100)         780200

 bidirectional_4 (Bidirectio  (None, None, 40)         14640
 nal)

 bidirectional_5 (Bidirectio  (None, 40)               7440
 nal)

 dense_2 (Dense)             (None, 2)                 82

=================================================================
Total params: 802,362
Trainable params: 22,162
Non-trainable params: 780,200
```

After the models were, trained, the test data set was tested on the models and confusion matrix, precision and recall were calculated.

Results

Confusion matrix

Table 6: Confusion matrix for LSTM with Glove

		Predicted	
		0	1
Actual	0	1445	19
	1	8	198

Table 7: Confusion matrix for GRU

		Predicted	
		0	1
Actual	0	1426	51
	1	27	166

Precision

Table 8: Precision

Precision for LSTM	97.41
Precision for GRU	95.53

Recall

Table 9: Recall

Recall for LSTM	91.28
Recall for GRU	87.32

Analysis

Did a certain type of word embedding perform better? From the results we can see that GloVe performed better than Word2Vec. This can be because Glove is based on global word to word co-occurrence whereas word2vec is based on local word to word co-occurrence.

Comparison with Baseline Models – Both these models fared extremely well when compared to the base models created in part 1 of this assignment. This is because the Embedding layer is highly trained using the glove or word2vec models. While in part 1, the embedding layer was trained using the dataset which was not much.

```python
!pip install datasets
from datasets import load_dataset
import numpy as np
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import tensorflow as tf
from keras.layers import LSTM, GRU, SimpleRNN, Dense, Embedding
from keras.models import Sequential
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score
dataset = load_dataset("SalehAhmad/Spam-Ham")
X = dataset['train']['sms_text']
Y = dataset['train']['label']
texts = []
labels = []
for i, label in enumerate(dataset['train']['label']):
    texts.append(dataset['train']['sms_text'][i])
    if label == 'ham':
        labels.append(0)
    else:
        labels.append(1)

texts = np.asarray(texts)
labels = np.asarray(labels)


print("number of texts :" , len(texts))
print("number of labels: ", len(labels))
X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.3, random_state=1)
# number of words used as features
max_features = 10000
# cut off the words after seeing 500 words in each document(email)
maxlen = 500
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)
sequences_train = tokenizer.texts_to_sequences(X_train)
word_index = tokenizer.word_index
print("Found {0} unique words: ".format(len(word_index)))
data_train = pad_sequences(sequences_train, maxlen=maxlen)
print(data_train.shape)
#SimpleRNN

model = Sequential()
model.add(Embedding(max_features, 64))
model.add(SimpleRNN(64))
model.add(Dense(1, activation='sigmoid'))
model.summary()

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history_rnn = model.fit(data_train, y_train, epochs=20, batch_size=60)
from sklearn.metrics import confusion_matrix
# number of words used as features
max_features = 10000
# cut off the words after seeing 500 words in each document(email)
maxlen = 500
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_test)
sequences_test = tokenizer.texts_to_sequences(X_test)
word_index = tokenizer.word_index
print("Found {0} unique words: ".format(len(word_index)))
data_test = pad_sequences(sequences_test, maxlen=maxlen)
print(data_test.shape)
pred = model.predict(data_test)
pred_label =[]
for i in pred:
  if i >0.5:
    pred_label.append(1)
  else:
    pred_label.append(0)
print(confusion_matrix(pred_label, y_test))
print('Precision for SimpleRNN: %.4f'% precision_score(y_test, pred_label,average='macro'))
print('Recall for SimpleRNN: %.4f' % recall_score(y_test, pred_label, average='macro'))
# LSTM
```

```
modelLSTM = Sequential()
modelLSTM.add(Embedding(max_features, 64))
modelLSTM.add(LSTM(64))
modelLSTM.add(Dense(1, activation='sigmoid'))
modelLSTM.summary()
modelLSTM.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history_ltsm = modelLSTM.fit(data_train, y_train, epochs=10, batch_size=60)
predLSTM = modelLSTM.predict(data_test)
predLSTM_label =[]
for i in predLSTM:
  if i >0.5:
    predLSTM_label.append(1)
  else:
    predLSTM_label.append(0)
print(confusion_matrix(predLSTM_label, y_test))
print('Precision for LSTM: %.4f'% precision_score(y_test, predLSTM_label,average='macro'))
print('Recall for LSTM: %.4f' % recall_score(y_test, predLSTM_label, average='macro'))
#GRU

modelGRU = Sequential()
modelGRU.add(Embedding(max_features, 64))
modelGRU.add(GRU(64))
modelGRU.add(Dense(1, activation='sigmoid'))
modelGRU.summary()
modelGRU.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history_gru = modelGRU.fit(data_train, y_train, epochs=10, batch_size=60)
predGRU = modelGRU.predict(data_test)
predGRU_label =[]
for i in predGRU:
  if i >0.5:
    predGRU_label.append(1)
  else:
    predGRU_label.append(0)

print(confusion_matrix(predGRU_label, y_test))
print('Precision for LSTM: %.4f'% precision_score(y_test, predGRU_label,average='macro'))
print('Recall for LSTM: %.4f' % recall_score(y_test, predGRU_label, average='macro'))
wordsCount=[]
for i in X_test:
  noOfWords = len(i.split())
  wordsCount.append(noOfWords)

print(len(wordsCount))
wordsCount.sort()
wordsCount[1669]
small_input = []
medium_input =[]
large_input=[]

small_input_label = []
medium_input_label =[]
large_input_label=[]

for i,c in enumerate(X_test):
  noOfWords = len(c.split())
  if noOfWords <= 32:
    small_input.append(c)
    small_input_label.append(y_test[i])
  elif noOfWords >32 and noOfWords <=64:
    medium_input.append(c)
    medium_input_label.append(y_test[i])
  elif noOfWords > 64:
    large_input.append(c)
    large_input_label.append(y_test[i])

print(len(small_input))
print(len(small_input_label))
print("----------------")
print(len(medium_input))
print(len(medium_input_label))
print("----------------")
print(len(large_input))
print(len(large_input_label))
small_input_label = np.array(small_input_label)
medium_input_label = np.array(medium_input_label)
```

```python
large_input_label = np.array(large_input_label)
#Short input

tokenizer = Tokenizer()
tokenizer.fit_on_texts(small_input)
sequences_short = tokenizer.texts_to_sequences(small_input)
data_short = pad_sequences(sequences_short, maxlen=maxlen)


pred_short = model.predict(data_short)
pred_short_label =[]
for i in pred_short:
  if i >0.5:
    pred_short_label.append(1)
  else:
    pred_short_label.append(0)

print(confusion_matrix(pred_short_label, small_input_label))
print('Precision for SimpleRNN: %.4f'% precision_score(small_input_label, pred_short_label,average='macro'))
print('Recall for SimpleRNN: %.4f' % recall_score(small_input_label, pred_short_label, average='macro'))

pred_shortLSTM = modelLSTM.predict(data_short)
pred_shortLSTM_label =[]
for i in pred_shortLSTM:
  if i >0.5:
    pred_shortLSTM_label.append(1)
  else:
    pred_shortLSTM_label.append(0)

print(confusion_matrix(pred_shortLSTM_label, small_input_label))
print('Precision for LSTM: %.4f'% precision_score(small_input_label, pred_shortLSTM_label,average='macro'))
print('Recall for LSTM: %.4f' % recall_score(small_input_label, pred_shortLSTM_label, average='macro'))

pred_shortGRU = modelGRU.predict(data_short)
pred_shortGRU_label =[]
for i in pred_shortGRU:
  if i >0.5:
    pred_shortGRU_label.append(1)
  else:
    pred_shortGRU_label.append(0)

print(confusion_matrix(pred_shortGRU_label, small_input_label))
print('Precision for GRU: %.4f'% precision_score(small_input_label, pred_shortGRU_label,average='macro'))
print('Recall for GRU: %.4f' % recall_score(small_input_label, pred_shortGRU_label, average='macro'))
#Medium input

tokenizer = Tokenizer()
tokenizer.fit_on_texts(medium_input)
sequences_medium = tokenizer.texts_to_sequences(medium_input)
data_medium = pad_sequences(sequences_medium, maxlen=maxlen)


pred_medium = model.predict(data_medium)
pred_medium_label =[]
for i in pred_medium:
  if i >0.5:
    pred_medium_label.append(1)
  else:
    pred_medium_label.append(0)

print(confusion_matrix(pred_medium_label, medium_input_label))
print('Precision for SimpleRNN: %.4f'% precision_score(medium_input_label, pred_medium_label,average='macro'))
print('Recall for SimpleRNN: %.4f' % recall_score(medium_input_label, pred_medium_label, average='macro'))

pred_mediumLSTM = modelLSTM.predict(data_medium)
pred_mediumLSTM_label =[]
for i in pred_mediumLSTM:
  if i >0.5:
    pred_mediumLSTM_label.append(1)
  else:
    pred_mediumLSTM_label.append(0)

print(confusion_matrix(pred_mediumLSTM_label, medium_input_label))
print('Precision for LSTM: %.4f'% precision_score(medium_input_label, pred_mediumLSTM_label,average='macro'))
print('Recall for LSTM: %.4f' % recall_score(medium_input_label, pred_mediumLSTM_label, average='macro'))
```

```
pred_mediumGRU = modelGRU.predict(data_medium)
pred_mediumGRU_label =[]
for i in pred_mediumGRU:
  if i >0.5:
    pred_mediumGRU_label.append(1)
  else:
    pred_mediumGRU_label.append(0)

print(confusion_matrix(pred_mediumGRU_label, medium_input_label))
print('Precision for GRU: %.4f'% precision_score(medium_input_label, pred_mediumGRU_label,average='macro'))
print('Recall for GRU: %.4f' % recall_score(medium_input_label, pred_mediumGRU_label, average='macro'))
#Lerge input

tokenizer = Tokenizer()
tokenizer.fit_on_texts(large_input)
sequences_large = tokenizer.texts_to_sequences(large_input)
data_large = pad_sequences(sequences_large, maxlen=maxlen)


pred_large = model.predict(data_large)
pred_large_label =[]
for i in pred_large:
  if i >0.5:
    pred_large_label.append(1)
  else:
    pred_large_label.append(0)

print(confusion_matrix(pred_large_label, large_input_label))
print('Precision for SimpleRNN: %.4f'% precision_score(large_input_label, pred_large_label,average='macro'))
print('Recall for SimpleRNN: %.4f' % recall_score(large_input_label, pred_large_label, average='macro'))

pred_largeLSTM = modelLSTM.predict(data_large)
pred_largeLSTM_label =[]
for i in pred_largeLSTM:
  if i >0.5:
    pred_largeLSTM_label.append(1)
  else:
    pred_largeLSTM_label.append(0)

print(confusion_matrix(pred_largeLSTM_label, large_input_label))
print('Precision for LSTM: %.4f'% precision_score(large_input_label, pred_largeLSTM_label,average='macro'))
print('Recall for LSTM: %.4f' % recall_score(large_input_label, pred_largeLSTM_label, average='macro'))

pred_largeGRU = modelGRU.predict(data_large)
pred_largeGRU_label =[]
for i in pred_largeGRU:
  if i >0.5:
    pred_largeGRU_label.append(1)
  else:
    pred_largeGRU_label.append(0)

print(confusion_matrix(pred_largeGRU_label, large_input_label))
print('Precision for GRU: %.4f'% precision_score(large_input_label, pred_largeGRU_label,average='macro'))
print('Recall for GRU: %.4f' % recall_score(large_input_label, pred_largeGRU_label, average='macro'))
```

```
!pip install datasets
from datasets import load_dataset
import numpy as np
from sklearn.model_selection import train_test_split
from keras.layers import TextVectorization
import tensorflow as tf
import os
from keras.layers import Embedding
from keras.initializers import Constant
from keras import layers, Input, Model
dataset = load_dataset("SalehAhmad/Spam-Ham")
X = dataset['train']['sms_text']
Y = dataset['train']['label']

texts = []
labels = []
for i, label in enumerate(dataset['train']['label']):
    texts.append(dataset['train']['sms_text'][i])
    if label == 'ham':
        labels.append(0)
    else:
        labels.append(1)


print("number of texts :" , len(texts))
print("number of labels: ", len(labels))
texts_train, texts_test, y_train, y_test = train_test_split(texts, labels, test_size=0.3, random_state=1)
vectorizer = TextVectorization(max_tokens=10000, output_sequence_length=100)
text_ds = tf.data.Dataset.from_tensor_slices(texts_train).batch(128) ## Read batches of 128 samples
vectorizer.adapt(text_ds)
print(len(vectorizer.get_vocabulary())) ## We set max_tokens=10000
vectorizer.get_vocabulary()[:5]
output = vectorizer([["I feel good today"]])
output.numpy()[0, :4]
voc = vectorizer.get_vocabulary()
word_index = dict(zip(voc, range(len(voc))))

## print the unique list of integers for the same string using the new map "Word_index"
test = ["i", "feel", "good", "today"]
[word_index[w] for w in test]
x_train = vectorizer(np.array([[s] for s in texts_train])).numpy()
y_train = np.array(y_train)
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip
path_to_glove_file = "glove.6B.100d.txt"

embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

print("Found %s word vectors." % len(embeddings_index))
num_tokens = len(voc)
embedding_dim = 100 ## 100 dimensions
hits = 0 ## number of words that were found in the pretrained model
misses = 0 ## number of words that were missing in the pretrained model

# Prepare embedding matrix for our word list
embedding_matrix = np.zeros((num_tokens, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in embedding index will be all-zeros.
        # This includes the representation for "padding" and "OOV"
        embedding_matrix[i] = embedding_vector
        hits += 1
    else:
        misses += 1
print("Converted %d words (%d misses)" % (hits, misses))
embedding_layer = Embedding(num_tokens, embedding_dim,
                            embeddings_initializer= Constant(embedding_matrix),
                            trainable=False,
)
int_sequences_input = Input(shape=(None,), dtype="int64")
embedded_sequences = embedding_layer(int_sequences_input)
```

```
x = layers.Bidirectional(layers.LSTM(20, return_sequences=True))(embedded_sequences)
x = layers.Bidirectional(layers.LSTM(20))(x)
preds = layers.Dense(2, activation="softmax")(x)
model = Model(int_sequences_input, preds)
model.summary()
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['acc'])
model.fit(x_train, y_train, batch_size=128, epochs=20)
string_input = Input(shape=(1,), dtype="string")
x = vectorizer(string_input)
preds = model(x)
end_to_end_model = Model(string_input, preds)
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score

predLSTM = end_to_end_model.predict(texts_test)
pred_label = []
for i in predLSTM:
    pred_label.append(np.argmax(i))
print(confusion_matrix(pred_label, y_test))
print('Precision for LSTM: %.4f'% precision_score(y_test, pred_label,average='macro'))
print('Recall for LSTM: %.4f' % recall_score(y_test, pred_label, average='macro'))
!pip install --upgrade gensim
import gensim
w2v_model = gensim.models.Word2Vec(vector_size=embedding_dim, window=3, min_count=5, workers=8)
documents=[text.split() for text in texts_train]
print(len(documents))
w2v_model.build_vocab(documents)
from gensim.models import KeyedVectors
embedding_matrix2 = np.zeros((num_tokens, embedding_dim))
for word, i in word_index.items():
  if word in w2v_model.wv:
    embedding_matrix2[i] = w2v_model.wv[word]
embedding_layer = Embedding(num_tokens, embedding_dim, weights=[embedding_matrix2], input_length=100, trainable=False)
int_sequences_input2 = Input(shape=(None,), dtype="int64")
embedded_sequences2 = embedding_layer(int_sequences_input2)
x2 = layers.Bidirectional(layers.LSTM(20, return_sequences=True))(embedded_sequences2)
x2 = layers.Bidirectional(layers.LSTM(20))(x2)
preds2 = layers.Dense(2, activation="softmax")(x2)
model2 = Model(int_sequences_input2, preds2)
model2.summary()
model2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['acc'])
model2.fit(x_train, y_train, batch_size=128, epochs=20)
string_input2 = Input(shape=(1,), dtype="string")
x = vectorizer(string_input2)
preds2 = model2(x)
end_to_end_model = Model(string_input2, preds2)
predLSTM = end_to_end_model.predict(texts_test)
pred_label = []
for i in predLSTM:
    pred_label.append(np.argmax(i))
print(confusion_matrix(pred_label, y_test))
print('Precision for LSTM: %.4f'% precision_score(y_test, pred_label,average='macro'))
print('Recall for LSTM: %.4f' % recall_score(y_test, pred_label, average='macro'))
```