

DEEP LEARNING ASSIGNMENT 2

Rishabh Jain - 19231092

Question 1 : Describe the data and any pre-processing steps you undertook (10)

The dataset is collected from Large Movie Review Dataset. The dataset contains 50,000 review evenly splitting into 25000 training and 25000 testing datasets. The overall distribution is balanced between 25,000 positive and 25,000 negative sets. There are an additional 50,000 unlabelled documents. a negative review has a score ≤ 4 out of 10, and a positive review has a score ≥ 7 out of 10. Thus, reviews with more neutral ratings are not included in the train/test sets. In the unsupervised set, reviews of any rating are included and there are an even number of reviews > 5 and ≤ 5 . Data contains two directories /test and /train corresponding to training and testing sets. Each of them has a /pos and /neg directory containing the positive and negative review. Reviews are stored in these directories in form a .txt file with rating of review in the range of 1-10. In addition to review text files, data also contains already-tokenized bag of words features that can be used in making our model. The data is stored in the .feat file. Each .feat file is in LIBSVM format. There is also an additional [imdbEr.txt] computed by (Potts, 2011) which contains the expected rating for each token in [imdb.vocab]. (This data classification is given in the readme file of the dataset).

I am using Spyder IDE for my assignment to run my deep learning tasks. I am taking a dataset from Kaggle for my assignment which consist the data in form of positive and negative reviews. The task is to label the reviews as **negative** or **positive**. The file is called **IMDB Dataset.csv**.

	A	B
1	review	sentiment
2	One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is exactly what happened with me. 	positive
3	A wonderful little production. The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomforting, sense of realism to the entire piece. The realism is very well done, and the script is very well written. The film is very well made, and the acting is very good. The film is very well made, and the acting is very good. The film is very well made, and the acting is very good.	positive
4	I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater and watching a light-hearted comedy. The plot is simple, but the film is very well made, and the acting is very good. The film is very well made, and the acting is very good. The film is very well made, and the acting is very good.	positive
5	Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fighting all the time. This movie is slower than a snail's pace, but it's very well made, and the acting is very good. The film is very well made, and the acting is very good. The film is very well made, and the acting is very good.	negative
6	Petter Mattei's "Love in the Time of Money" is a visually stunning film to watch. Mr. Mattei offers us a vivid portrait about human relations. This is a movie that positive	positive
7	Probably my all-time favorite movie, a story of selflessness, sacrifice and dedication to a noble cause, but it's not preachy or boring. It just never gets old, despite the age of the film. The film is very well made, and the acting is very good. The film is very well made, and the acting is very good. The film is very well made, and the acting is very good.	positive
8	I sure would like to see a resurrection of a up dated Seahunt series with the tech they have today it would bring back the kid excitement in me. I grew up on big screen movies, and I would love to see a new series. The film is very well made, and the acting is very good. The film is very well made, and the acting is very good. The film is very well made, and the acting is very good.	positive
9	This show was an amazing, fresh & innovative idea in the 70's when it first aired. The first 7 or 8 years were brilliant, but things dropped off after that. By 1990, the show was over, and I was disappointed. The film is very well made, and the acting is very good. The film is very well made, and the acting is very good. The film is very well made, and the acting is very good.	negative
10	Encouraged by the positive comments about this film on here I was looking forward to watching this film. Bad mistake. I've seen 950+ films and this is truly one of the worst I have ever seen. The film is very well made, and the acting is very good. The film is very well made, and the acting is very good. The film is very well made, and the acting is very good.	negative

#Load the data

```
Movie_df=pd.read_csv('IMDB Dataset.csv')
print('Shape of dataset::',Movie_df.shape)
Movie_df.head(10)
```

When we load the dataset, you will notice that there are lot of `
` tags which will likely create a problem during training because it doesn't give us any indication is the dataset if positive or negative. There is also a lot of noise in the data which is useless data. We can remove stopwords which provide little context to a sentence. We remove these words to remove the noise from the sentences. We can make a list of stopwords like a,an,the...etc. and use that list to remove the stopwords from our sentences. We can also make use of regular expressions to manipulate our data. We can replace `
` with an empty string. We can also replace character that are not lowercase letter for cleaning our data. Those are the pre-processing steps I will take.

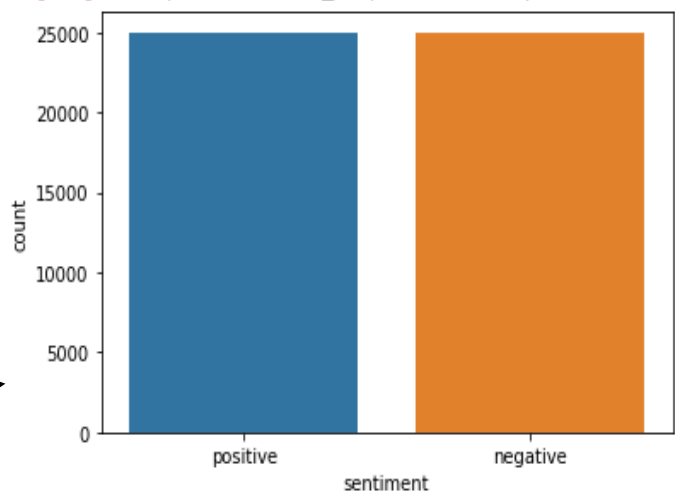
We can see that there are equal number of positive and negative reviews

```
import seaborn as sns
sns.countplot(x='sentiment', data=Movie_df)
```

```
In [102]: print('Shape of dataset::',Movie_df.shape)
...: Movie_df.head(10)
Shape of dataset:: (50000, 2)
Out[102]:
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
5	Probably my all-time favorite movie, a story o...	positive
6	I sure would like to see a resurrection of a u...	positive
7	This show was an amazing, fresh & innovative i...	negative
8	Encouraged by the positive comments about this...	negative
9	If you like original gut wrenching laughter yo...	positive

```
In [103]: sns.countplot(x='sentiment', data=Movie_df)
Out[103]: <matplotlib.axes._subplots.AxesSubplot at 0x1d2630ce748>
```



Let's look at the stats of our data

#Stats of data

```
print("General stats:")
print(Movie_df.info())
print("Summary stats:\n")
print(Movie_df.describe())
```

```
General stats::
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
review      50000 non-null object
sentiment    50000 non-null object
dtypes: object(2)
memory usage: 781.3+ KB
None
Summary stats::
```

#Number of positive & negative reviews

```
Movie_df.sentiment.value_counts()
```

```
count      review sentiment
unique      50000      50000
top         49582         2
freq    Loved today's show!!! It was a variety and not...  negative
                                     5      25000
```

Now, for the pre-processing:

#Encode our target labels

```
lb=preprocessing.LabelBinarizer()
```

#Encode 1 for positive label & 0 for Negative label

```
train_sentiment=lb.fit_transform(train_sentiment)
```

```
test_sentiment=lb.transform(test_sentiment)
```

#Reshape the array

```
train_sentiment=train_sentiment.ravel()
```

```
test_sentiment=test_sentiment.ravel()
```

#Convert categorical to numeric ones

```
train_sentiment=train_sentiment.astype('int64')
```

```
test_sentiment=test_sentiment.astype('int64')
```

```
In [106]: Movie_df.sentiment.value_counts()
Out[106]:
negative    25000
positive    25000
Name: sentiment, dtype: int64
```

#Let's explore our data before normalization

```
train_reviews[0]
```

```
test_reviews[30001]
```

```
In [107]: train_reviews[0]
Out[107]: "One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is exactly what happened with me.<br /><br />The first thing that struck me about Oz was its brutality and unflinching scenes of violence, which set in right from the word GO. Trust me, this is not a show for the faint hearted or timid. This show pulls no punches with regards to drugs, sex or violence. Its is hardcore, in the classic use of the word.<br /><br />It is called Oz as that is the nickname given to the Oswald Maximum Security State Penitentiary. It focuses mainly on Emerald City, an experimental section of the prison where all the cells have glass fronts and face inwards, so privacy is not high on the agenda. Em City is home to many..Aryans, Muslims, gangstas, Latinos, Christians, Italians, Irish and more....so scuffles, death stares, dodgy dealings and shady agreements are never far away.<br /><br />I would say the main appeal of the show is due to the fact that it goes where other shows wouldn't dare. Forget pretty pictures painted for mainstream audiences, forget charm, forget romance...Oz doesn't mess around. The first episode I ever saw struck me as so nasty it was surreal, I couldn't say I was ready for it, but as I watched more, I developed a taste for Oz, and got accustomed to the high levels of graphic violence. Not just violence, but injustice (crooked guards who'll be sold out for a nickel, inmates who'll kill on order and get away with it, well mannered, middle class inmates being turned into prison bitches due to their lack of street skills or prison experience) Watching Oz, you may become comfortable with what is uncomfortable viewing....thats if you can get in touch with your darker side."
```

```
In [108]: test_reviews[30001]
Out[108]: 'This is one of the silliest movies I have ever had the misfortune to watch! I should have expected it, after seeing the first two, but I keep getting suckered into these types of movies with the idea of "Maybe they did it right this time". Nope - not even close.<br /><br />Where do I begin? How about with the special effects... To give you an idea of what passes for SFX in this movie, at one point a soldier is shooting at a "Raptor" as it runs down a hallway. Even with less than a second of screen time, the viewer can easily see that it is just a man with a tail apparently taped to him running around. Bad bad bad bad.<br /><br />How about the acting? If that's what you can call it. There is one character who, I suppose, is supposed to be from the south. However, after living in the south for six years now, I have never heard this way of talking. Perhaps he has some sort of weird disability - the inability to talk normally. I find it fascinating that the character does nothing that requires him to have that accent - therefore there was no reason for the actor to try to do one.<br /><br />How about the plot? It's pretty basic - Raptors escape, people with guns must hunt them down. I'm starting to wonder why the dinosaurs in these movies always seem to run into the nearest system of tunnels... wouldn't they stay outside to hunt prey? Oh well, at least they have the good sense to appear very very little in the movie which supposedly revolves around them.<br /><br />Other things - Let's say you are in a building and you know that there are man eating raptors running around in it. Would you decide to take time out to have an argument about who is better - Army or Marine? And then decide to have an arm wrestling contest to settle it? How about the idiotic idea that they have to track down the raptors - Split up into groups of two. Didn't they ever watch any horror movies (Or at least an episode of Scooby Doo)? In short, this is one of the dumber movies out there. Miss it unless you want to groan your way through a movie.'
```

In above paragraphs, we can observe stop words, html tags, special characters & numbers, which are not required for sentiment analysis. So, we need to remove those by normalizing the review data to reduce dimensionality & noise in the data.

Let's normalize our data to remove stopwords, html tags and so on. The main steps include exclusion of html tags, conversion to lower case, tokenizing the review data, removing stop words, and applying stemming.

```
ps=PorterStemmer()
stopwords=set(stopwords.words('english'))
# Define function for data mining
def normalize_reviews(review):
    #Excluding html tags
    data_tags=re.sub(r'<[^>]+>','',review)
    #Remove special characters/whitespaces
    data_special=re.sub(r'[^a-zA-Z0-9\s]','',data_tags)
    #converting to lower case
    data_lowercase=data_special.lower()
    #tokenize review data
    data_split=data_lowercase.split()
    #Removing stop words
    meaningful_words=[w for w in data_split if not w in stopwords]
    #Apply stemming
    text= ' '.join([ps.stem(word) for word in meaningful_words])
    return text
```

#Normalize the train & test data

```
norm_train_reviews=train_reviews.apply(normalize_reviews)
```

```
In [110]: norm_train_reviews[0]
Out[110]: 'one review mention watch 1 oz episod youll hook right exactli happen first thing struck oz
brutal unflinch scene violenc set right word go trust show faint heart timid show pull punch regard
drug sex violenc hardcor classic use word call oz nicknam given oswald maximum secur state
penitentari focus mainli emerald citi experiment section prison cell glass front face inward privaci
high agenda em citi home manyaryan muslim gangsta latino christian italian irish moreso scuffl death
stare dodgi deal shadi agreement never far away would say main appeal show due fact goe show wouldnt
dare forget pretti pictur paint mainstream audienc forget charm forget romanceoz doesnt mess around
first episod ever saw struck nasti surreal couldnt say readi watch develop tast oz got accustom high
level graphic violenc violenc injustic crook guard wholl sold nickel inmat wholl kill order get away
well manner middl class inmat turn prison bitch due lack street skill prison experi watch oz may
becom comfort uncomfot viewingthat get touch darker side'
```

```
norm_test_reviews=test_reviews.apply(normalize_reviews)
```

```
In [111]: norm_test_reviews[30001]
Out[111]: 'one silliest movi ever misfortun watch expect see first two keep get sucker type movi idea
mayb right time nope even close begin special effect give idea pass sfx movi one point soldier shoot
raptor run hallway even less second screen time viewer easili see man tail appar tape run around bad
bad bad act that call one charact suppos suppos south howev live south six year never heard way
talk perhap sort weird disabl inabl talk normal find fascin charact noth requir accent therefor
reason actor tri one plot pretti basic raptor escap peopl gun must hunt im start wonder dinosaur movi
alway seem run nearest system tunnel wouldnt stay outsid hunt prey oh well least good sens appear
littl movi supposedli revolv around thing let say build know man eat raptor run around would decid
take time argument better armi marin decid arm wrestl contest settl idiot idea track raptor split
group two didnt ever watch horror movi least episod scoobi doo short one dumber movi miss unless want
groan way movi'
```

#Let's create features using bag of words model

```
cv=CountVectorizer(ngram_range=(1,2))
train_cv=cv.fit_transform(norm_train_reviews)
test_cv =cv.transform(norm_test_reviews)
print('Shape of train_cv::',train_cv.shape)
print('Shape of test_cv::',test_cv.shape)
```

```
In [112]: print('Shape of train_cv::',train_cv.shape)
...: print('Shape of test_cv::',test_cv.shape)
Shape of train_cv:: (30000, 1929440)
Shape of test_cv:: (20000, 1929440)
```

Question 2 : Choose a suitable NN architecture and justify your choice. (20)

I am using recurrent neural network architecture for my assignment. I am also adding additional LSTM layer to my model. So, I am using recurrent neural network with LSTM. Recurrent neural networks are very popular for solving text classification tasks. It is a neural network that is intentionally run multiple times, where parts of each run feed into the next run. Hidden layer from the previous run provides input to the same hidden layer in the next run. Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. In RNNs, the decisions are influenced by what it learnt in the past. I am using RNN's because it can process input of any length, the size of the model does not increase with the size of input, the weights are shared across time, and it takes into account the historical information when computation takes place. The RNNs can be viewed in the figure 1 below:

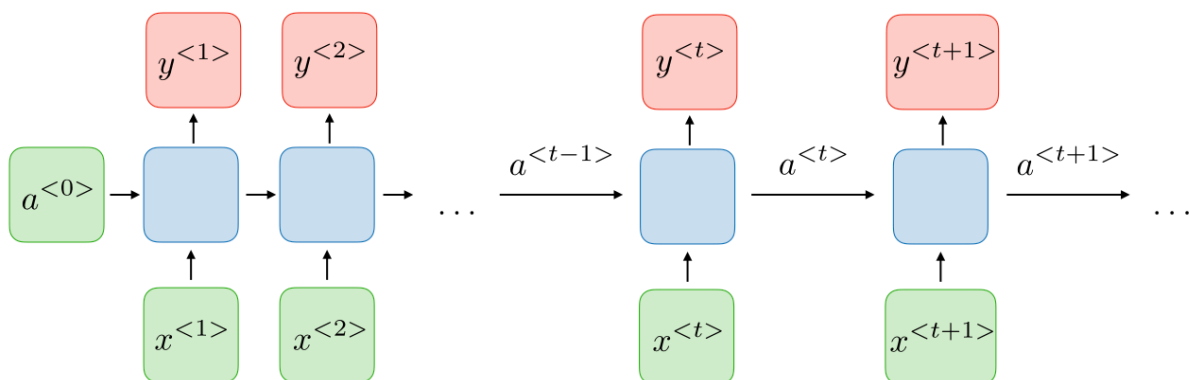


Figure 1

For each timestep t , the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad \text{and} \quad y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions.

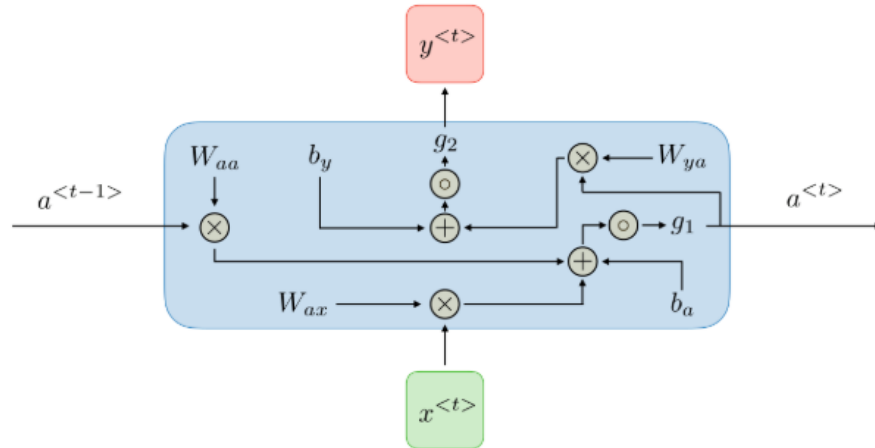


Figure 2

For example,

Suppose our RNN runs four times. When RNN runs the first time, the values learn in hidden player from the first run will be used as an input to the hidden layer in the second run. Similarly, the values learn in hidden player from the second run will be used as an input to the hidden layer in the third run. This process continues for training and predicting the neural network. This can be illustrated from the diagram below (figure 3) :

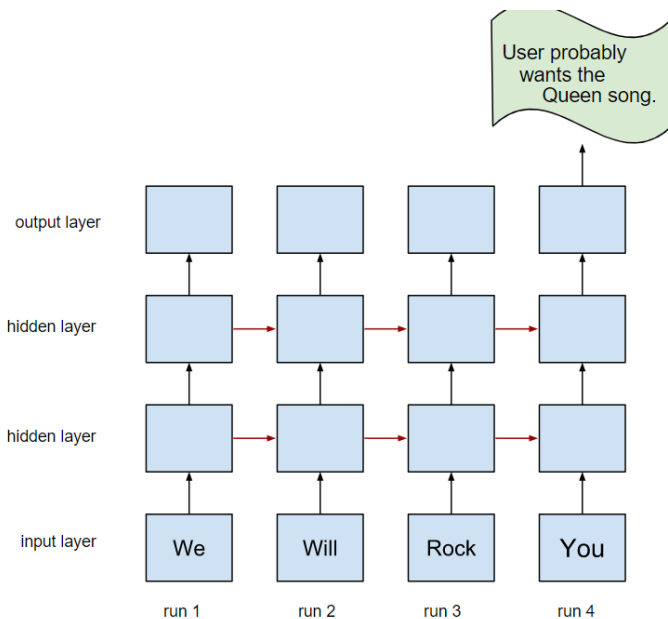


Figure 3

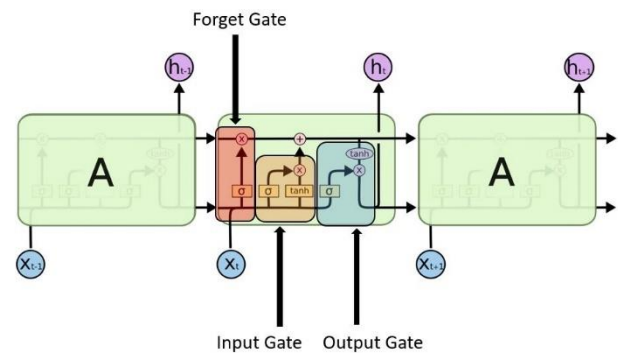


Figure 4

I am then adding LSTM layer to my RNN. Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks. LSTM helps in remembering the past data in memory. It makes the process easier. It trains the model by back propagation which makes it easier to remember past data in memory. LSTM can allow the neural network to forget and remember the information selectively and introduces the gates to regulate the flow of information i.e. adding and removing information. LSTM is a three-step process. Every LSTM module will have 3 gates named as Forget gate, Input gate, Output gate. You can see that in the figure 4. RNN and LSTM can be used together in conjunction. RNN might create a problem to identify the relationship between inputs that are far apart from each other. LSTM can be used to solve this problem as they can remember for longer periods of time.

Question 3 : Use TensorFlow to train on the data set and document your results. (40)

I am using code available at tensor flow website and Kaggle website as a reference code for my problem. I am using this code because I have read the book **Deep learning with Python** for learning keras and tensor flow and the author **François Chollet** has given permission to use this code free of charge for learning. Also, the author on Kaggle has given the permission to use the code for learning. CODE is stored in the **Assignment2_19231092.py** file. I am using tensor flow library to create my recurrent neural network. I am also using adding a LSTM (Long Short-Term Memory network) layer to my RNN model, which is also a variant of RNN to solve sentiment classification problem.

I am creating two models here:

a) Random forest classifier ML model (for comparing my model in the Question 4 (ii))

```
#Random Forest model
#Training the classifier
rfc=RandomForestClassifier(n_estimators=20,random_state=42)
rfc=rfc.fit(train_cv,train_sentiment)
score=rfc.score(train_cv,train_sentiment)
print('Accuracy of trained model is ::',score)

#Making predictions
rfc_predict=rfc.predict(test_cv)

#Accuracy and confusion matrix
cm=confusion_matrix
(test_sentiment,rfc_predict)

#plot our confusion matrix
skplt.metrics.plot_confusion_matrix
(test_sentiment,rfc_predict,
normalize=False,figsize=(12,8))
plt.show()

#print classification report for performance
metrics
cr=classification_report
(test_sentiment,rfc_predict)
print('Classification report is::\n',cr)

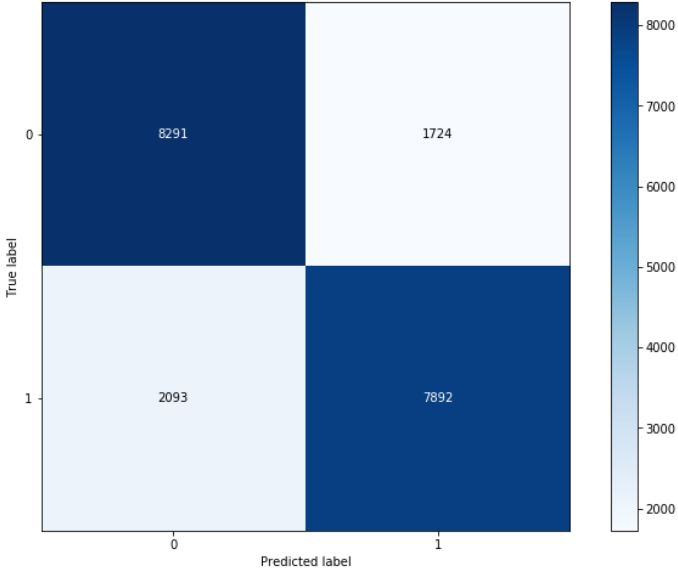
#ROC curve for Random Forest Classifier
fpr_rf,tpr_rf,threshold_rf=roc_curve(test_sentiment,rfc_predict)

#Area under curve (AUC) score, fpr-False
Positive rate, tpr-True Positive rate
auc_rf=auc(fpr_rf,tpr_rf)
print('AUC score for Random Forest
classifier::',np.round(auc_rf,3))
```

In [78]: rfc=RandomForestClassifier(n_estimators=20,random_state=42)
...: rfc=rfc.fit(train_cv,train_sentiment)
...: score=rfc.score(train_cv,train_sentiment)
...: print('Accuracy of trained model is ::',score)
Accuracy of trained model is :: 0.9996

In [80]: rfc_predict
Out[80]: array([1, 0, 1, ..., 0, 1, 0], dtype=int64)

Confusion Matrix



	0	1
0	8291	1724
1	2093	7892

Classification report is::\n

	precision	recall	f1-score	support
0	0.80	0.83	0.81	10015
1	0.82	0.79	0.81	9985
accuracy			0.81	20000
macro avg	0.81	0.81	0.81	20000
weighted avg	0.81	0.81	0.81	20000

...: print('AUC score for Random Forest classifier::',np.round(auc_rf,3))
AUC score for Random Forest classifier:: 0.809

b) Recurrent Neural network with LSTM (for my assignment problem)

```
#Recurrent neural network (RNN) with LSTM (Long Short-Term Memory) model
#Train dataset
X_train=train_cv
X_train=[str(x[0]) for x in X_train]
y_train=train_sentiment
# Test dataset
```



```
X_test=test_cv
X_test=[str(x[0]) for x in X_test]
y_test=test_sentiment
```

Tokenize the train & test dataset

```
Max_Review_length=500
tokenizer=Tokenizer(num_words=Max_Review_length,lower=False)
tokenizer.fit_on_texts(X_train)
```

#tokenizing train data

```
X_train_token=tokenizer.texts_to_sequences(X_train)
```

#tokenizing test data

```
X_test_token=tokenizer.texts_to_sequences(X_test)
```

#Truncate or pad the dataset for a length of 500 words for each review

```
X_train=pad_sequences(X_train_token,maxlen=Max_Review_length)
X_test=pad_sequences(X_test_token,maxlen=Max_Review_length)
```

```
print('Shape of X_train dataset after
padding:',X_train.shape)
print('Shape of X_test dataset after
padding:',X_test.shape)
```

```
In [163]: print('Shape of X_train dataset after padding:',X_train.shape)
...: print('Shape of X_test dataset after padding:',X_test.shape)
Shape of X_train dataset after padding: (30000, 500)
Shape of X_test dataset after padding: (20000, 500)
```

Most popular words found in the dataset

```
vocabulary_size=5000
embedding_size=64
model=Sequential()
model.add(Embedding(vocabulary_size,embedding_size,input_length=Max_Review_length))
model.add(LSTM(30))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

#Compile our model

```
model.compile(loss='binary_crossentropy',
optimizer='adam',metrics=['accuracy'])
```

#Train our model

```
batch_size=128
num_epochs=6
X_valid,y_valid=X_train[:batch_size],
train_sentiment[:batch_size]
X_train1,y_train1=X_train[batch_size:],
train_sentiment[batch_size:]
```

Fit the model

```
history =
model.fit(X_train1,y_train1,validation_data=(X_valid,y_valid),validation_split=0.2,
batch_size=batch_size,epochs=num_epochs, verbose=1,shuffle=True)
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
embedding_8 (Embedding)	(None, 500, 64)	320000
bidirectional_4 (Bidirection	(None, 128)	66048
dense_10 (Dense)	(None, 64)	8256
dense_11 (Dense)	(None, 1)	65
Total params: 394,369		
Trainable params: 394,369		
Non-trainable params: 0		

```
In [197]: batch_size=128
...: num_epochs=6
...: X_valid,y_valid=X_train[:batch_size],train_sentiment[:batch_size]
...: X_train1,y_train1=X_train[batch_size:],train_sentiment[batch_size:]
...: # Fit the model
...: history = model.fit(X_train1,y_train1,validation_data=(X_valid,y_valid),validation_split=0.2,
...: batch_size=batch_size,epochs=num_epochs, verbose=1,shuffle=True)
C:\Users\Rishabh Jain\Anaconda3\lib\site-packages\tensorflow_core\python\framework\indexed_slices.py:433: UserWarning: Converting sparse IndexedSlices to a dense Tensor of unknown shape. This may consume a large amount of memory.
"Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
Train on 29872 samples, validate on 128 samples
Epoch 1/6
29872/29872 [=====] - 963s 32ms/step - loss: 0.6074 - accuracy: 0.6470 - val_loss: 0.4747 - val_accuracy: 0.7422
Epoch 2/6
29872/29872 [=====] - 1028s 34ms/step - loss: 0.4906 - accuracy: 0.7665 - val_loss: 0.4754 - val_accuracy: 0.7578
Epoch 3/6
29872/29872 [=====] - 983s 33ms/step - loss: 0.4758 - accuracy: 0.7764 - val_loss: 0.4330 - val_accuracy: 0.8203
Epoch 4/6
29872/29872 [=====] - 679s 23ms/step - loss: 0.4659 - accuracy: 0.7820 - val_loss: 0.4176 - val_accuracy: 0.8281
Epoch 5/6
29872/29872 [=====] - 669s 22ms/step - loss: 0.4572 - accuracy: 0.7844 - val_loss: 0.4209 - val_accuracy: 0.8203
Epoch 6/6
29872/29872 [=====] - 742s 25ms/step - loss: 0.4533 - accuracy: 0.7868 - val_loss: 0.4356 - val_accuracy: 0.7812
```

Predictions

```
y_predict_rnn=model.predict(X_test)
```

#Changing the shape of y_predict to 1-Dimensional

```
y_predict_rnn1=y_predict_rnn.ravel()
y_predict_rnn1=(y_predict_rnn1>0.5)
y_predict_rnn1[0:10]
```

```
In [200]: y_predict_rnn1[0:10]
Out[200]:
array([False, False, True, True, False, True, True, True, True,
       False])
```

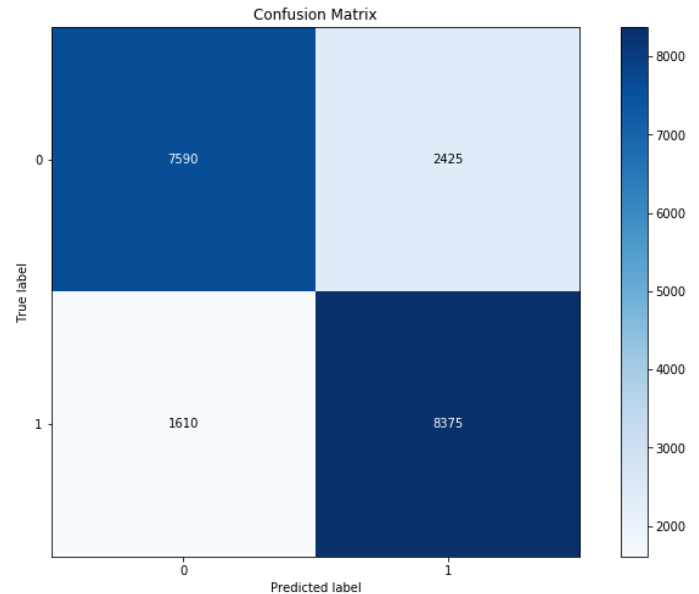
#Accuracy

```
score = model.evaluate(X_test,y_test)
print("Test Score:", score[0])
print("Test Accuracy:", score[1] )
```

```
In [201]: score = model.evaluate(X_test,y_test)
...: print("Test Score:", score[0])
...: print("Test Accuracy:", score[1])
20000/20000 [=====] - 164s 8ms/step
Test Score: 0.44174904391765596
Test Accuracy: 0.7982500195503235
```

#Confusion matrix for RNN with LSTM

```
cm_rnn=confusion_matrix(y_test,
y_predict_rnn1)
#plot our confusion matrix
skplt.metrics.plot_confusion_matrix
(y_test,y_predict_rnn1,
normalize=False,figsize=(12,8))
plt.show()
```



#Classification report for performance metrics

```
cr_rnn=classification_report
(y_test,y_predict_rnn1)
print('The Classification report
is::\n',cr_rnn)
```

```
In [203]: cr_rnn=classification_report(y_test,y_predict_rnn1)
...: print('The Classification report is::\n',cr_rnn)
The Classification report is::

```

	precision	recall	f1-score	support
0	0.82	0.76	0.79	10015
1	0.78	0.84	0.81	9985
accuracy			0.80	20000
macro avg	0.80	0.80	0.80	20000
weighted avg	0.80	0.80	0.80	20000

#ROC curve for RNN with LSTM

```
fpr_rnn,tpr_rnn,threshold_rnn=
roc_curve(y_test,y_predict_rnn)
#AUC score for RNN
auc_rnn=auc(fpr_rnn,tpr_rnn)
print('AUC score for RNN with LSTM
::',np.round(auc_rnn,3))
```

```
In [204]: fpr_rnn,tpr_rnn,threshold_rnn=roc_curve(y_test,y_predict_rnn)
...: #AUC score for RNN
...: auc_rnn=auc(fpr_rnn,tpr_rnn)
...: print('AUC score for RNN with LSTM ::',np.round(auc_rnn,3))
AUC score for RNN with LSTM :: 0.879
```

Question 4 : Discuss the performance of your model. (30)

This could be achieved through:

i) Accuracy achieved – plots/tables of performance

Accuracy of model is coming to be 79.8%.

#Accuracy

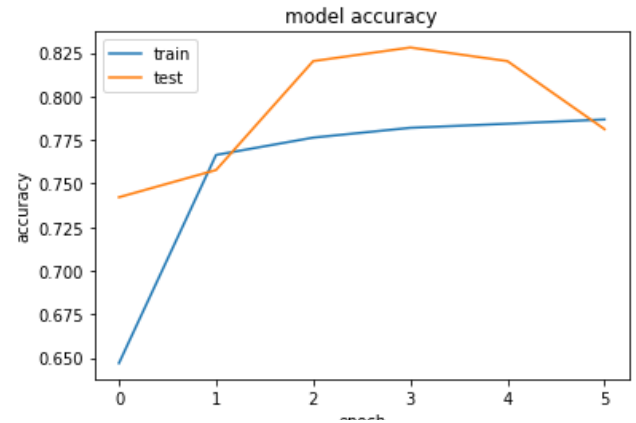
```
score = model.evaluate(X_test,y_test)
print("Test Score:", score[0])
print("Test Accuracy:", score[1] )
```

```
In [201]: score = model.evaluate(X_test,y_test)
...: print("Test Score:", score[0])
...: print("Test Accuracy:", score[1])
20000/20000 [=====] - 164s 8ms/step
Test Score: 0.44174904391765596
Test Accuracy: 0.7982500195503235
```

```
#PLOT for accuracy and loss
import matplotlib.pyplot as plt
```

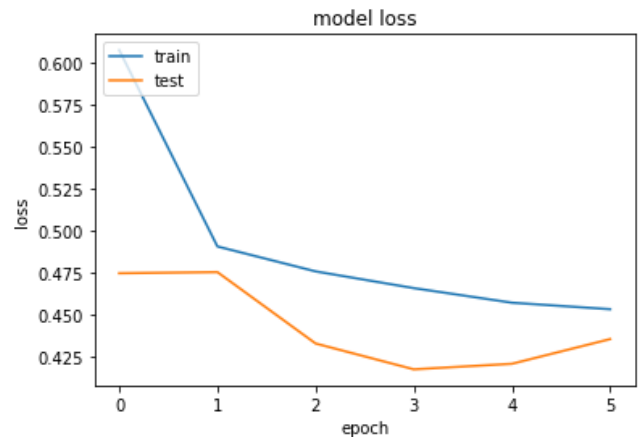
#Accuracy plot

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



#Loss plot

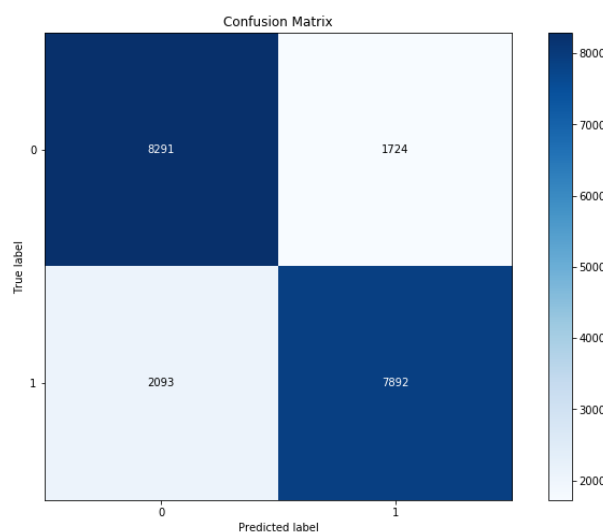
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



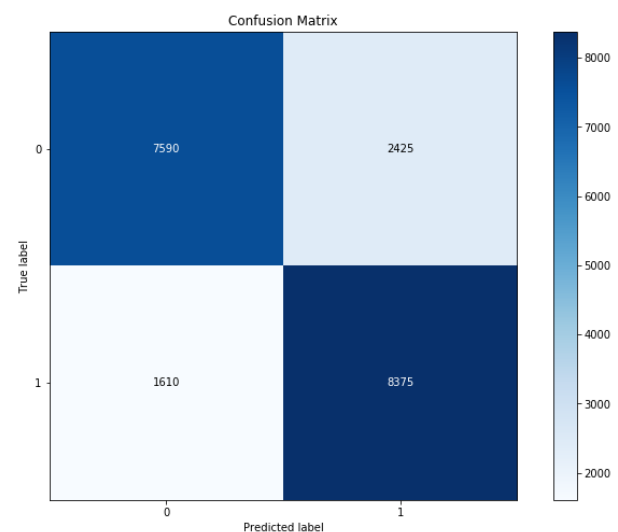
The difference between accuracy of training and test set is decreasing and coming closer with every epoch. The difference between loss values is also negligible. We can conclude that RNN with LSTM is a good model for our assignment problem. Model is overfitting initially as there is a vast difference between the training and test accuracy. But with increasing epoch, the overfitting seems to be decreasing.

ii) Comparing your performance to an alternative model (you may choose a naïve approach to show the improvement obtained with your model from 2).

I am choosing a random forest classification machine learning model as an alternative model for comparing the performance with my RNN with LSTM model.



Confusion matrix for random forest classifier



Confusion matrix for RNN with LSTM

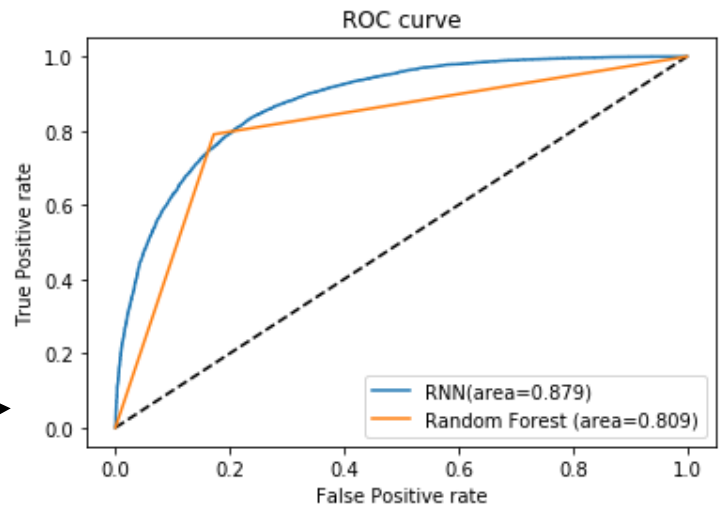
Here, 0 is Negative class and 1 is Positive class.

From the confusion matrix plot, it is concluded that, the Random Forest classifier with 20 decision trees classified the 81% of the reviews (16183 reviews) correctly & remaining 19% of reviews (3817 reviews) incorrectly.

The confusion matrix plot for the RNN with LSTM model classified 79% of reviews (15965 reviews) correctly & remaining 21% of reviews (4035 reviews) incorrectly.

Confusion matrix is almost same for both the plot. Let's have a look at ROC Curve.

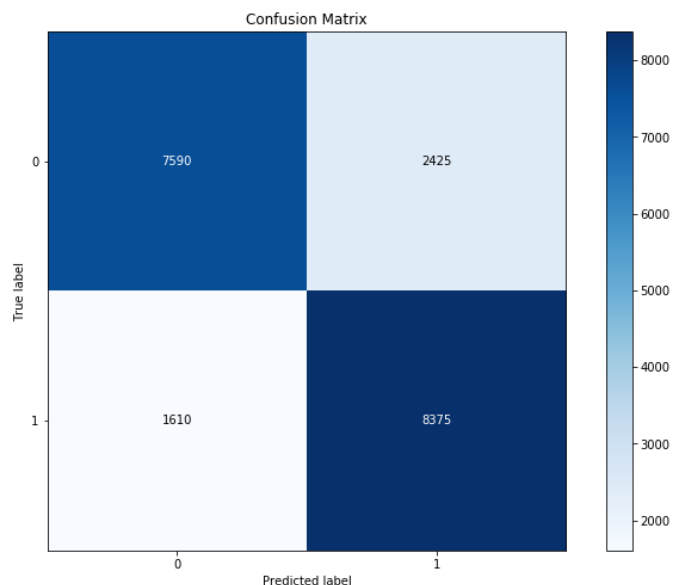
```
#Receiver Operating Characteristic(ROC)
Curve for Model Evaluation
#Now, let's plot the ROC for both Random
Forest Classifier & RNN with LSTM
plt.figure(1)
plt.plot([0,1],[0,1], 'k--')
plt.plot(fpr_rnn,tpr_rnn,label=
'RNN(area={:.3f})'.format(auc_rnn))
plt.plot(fpr_rf,tpr_rf,label='Random
Forest (area={:.3f})'.format(auc_rf))
plt.xlabel('False Positive rate')
plt.ylabel('True Positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```



The f1-score for Random forest classifier is higher than for RNN with LSTM model & the roc_auc score for Random forest classifier is lower than for RNN with LSTM model. From the scores above, we can say that the Random forest classifier is better than RNN with LSTM. This is because Random forest classifier is comparatively less computationally expensive and works well on small and large datasets.

iii) Illustrating cases where it gives the incorrect result for unseen cases.

The confusion matrix plot for the RNN with LSTM model classified 79% of reviews (15965 reviews) correctly & remaining 21% of reviews (4035 reviews) incorrectly. We can use this data to say that the accuracy for model to give incorrect result for unseen cases is 21%. We can perform model evaluation on unseen dataset by comparing it with Random forest classifier. We can use the f1 score to see the balance between precision and recall. Precision is the number of True Positives divided by the number of True Positives and False Positives. Precision can be thought of as a measure of a classifier's exactness. Recall is the number of True Positives divided by the number of True Positives and the number of False Negatives. Recall can be thought of as a measure of a classifier's completeness. We can use f1 score to measure the accuracy of the test data. Clearly low f1 score suggests incorrect results for unseen cases. You can see the precision and recall measures in the classification report. F1 score is 79% for negative reviews and 81% for positive reviews. Accuracy is given to be 80%.



```
In [203]: cr_rnn=classification_report(y_test,y_predict_rnn1)
...: print('The Classification report is::\n',cr_rnn)
The Classification report is::
```

	precision	recall	f1-score	support
0	0.82	0.76	0.79	10015
1	0.78	0.84	0.81	9985
accuracy			0.80	20000
macro avg	0.80	0.80	0.80	20000
weighted avg	0.80	0.80	0.80	20000

#Model Evaluation on unseen dataset

```
Model_evaluation=pd.DataFrame({'Model':['Random Forest Classifier','RNN with LSTM'],  
                               'f1_score':[0.81,0.79],  
                               'roc_auc_score':[np.round(auc_rf,3),np.round(auc_rnn,3)]})  
Model_evaluation
```

Out[208]:

	Model	f1_score	roc_auc_score
0	Random Forest Classifier	0.81	0.809
1	RNN with LSTM	0.79	0.879

REFERENCES AND ACKNOWLEDGEMENT:

- [1] https://www.tensorflow.org/datasets/api_docs/python/tfds/features/text/SubwordTextEncoder
- [2] https://developers.google.com/machine-learning/glossary/#recurrent_neural_network
- [3] <https://stackabuse.com/python-for-nlp-movie-sentiment-analysis-using-deep-learning-in-keras/>
- [4] <https://www.kaggle.com/ramanchandra/sentiment-analysis-on-imdb-movie-reviews/notebook>
- [5] https://www.tensorflow.org/datasets/catalog/imdb_reviews
- [6] <https://tfhub.dev/google/nnlm-en-dim128/1>
- [7] <https://machinelearningmastery.com/predict-sentiment-movie-reviews-using-deep-learning/http://irejournals.com/formatedpaper/1700620.pdf>
- [8] https://www.tensorflow.org/hub/tutorials/text_classification_with_tf_hub
- [9] <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>
- [10] <https://www.kaggle.com/ramanchandra/sentiment-analysis-on-imdb-movie-reviews/notebook>
- [11] <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
- [12] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). [Learning Word Vectors for Sentiment Analysis](#). The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).