

Assignment - 2 Report

S.No	Name	Roll Number
1	Anurag Saraswat	M20CS066
2	Parsa Revanth	M20CS058

Colab Links:

1. **Question 1(part a and part b),**
<https://colab.research.google.com/drive/1OM0HI4PvJ4AwdSUWlGm9ZHyiA3XCrfIB?usp=sharing>
2. **Question 1(remaining parts),**
<https://colab.research.google.com/drive/1mkxcjIaao6oHyzBddShFt49cEXdEFiPK?usp=sharing>
3. **Question 2,** we did it in the laptop, so adding the github link which contains all the required files, <https://github.com/anurag-saraswat/RASA-Chatbot>
4. **Question 3,**
https://colab.research.google.com/drive/1HhBjx-vL9tdhyLi-YtdXzMFO_iwVkjmn?usp=sharing

Question - 1

Part *a* and Part *b*

DATASET:

- Firstly, we downloaded the 20newsgroups dataset from the sklearn datasets
- Secondly, we created a decontract function in order to convert the selected short forms to full forms like 're -> are, won't -> will not, can't -> can not, 'd -> would, 's -> is, 't -> not, 'll -> will, 'm -> am, 've -> have
- Thirdly, we pass it to the pipeline function to remove punctuation, numbers, curly braces, square brackets, and also remove the stop words
- Finally, we have the cleaned data which will be used for creating NER tags using the stanford NER and Spacy NER

Stanford NER:

- Firstly, in order to calculate the NER tags with stanford NER we should download the stanford NER related files in the current repository and initialise the stanford ner model
- Secondly, we combined the whole corpus into one string and divided it into ten strings. We did this because if we pass the combined string the colab gets crashed after passing more than 10 lakh tags to the stanford NER tag() function
- Thirdly, we created the word tokens using the nltk library
- Fourthly, we passed the word tokens for each sequence separately into the tag() function of stanford NER and generated the tags
- Fifthly, we created two lists corresponding to both LOC and PERSON tags separately
- Seventhly, we created a frequency function which takes input a list and returns a dictionary which is sorted in the decreasing order of the number of occurrences
- Sixthly, we used frequency function to create the top 100 LOC and top 100 PERSON tags in the decreasing order of their number of occurrences in the 20newsgroup dataset
- Finally, we created a dataframe to display the top 100 LOC and top 100 PERSON tags in the structured format.
- Words with top 100 LOC and PERSON tags which are generated using stanford NER are in the below table
- The 2nd column contains the location and 3rd column contains the number of occurrences of location entity, the 4th column contains the person entity, the 5th column contains the number of occurrences of person entity

	<i>Location</i>	<i>Location Frequency</i>	<i>Person</i>	<i>Person Frequency</i>
0	<i>Israel</i>	809	<i>Jesus</i>	1347
1	<i>US</i>	804	<i>John</i>	1058
2	<i>Earth</i>	338	<i>David</i>	996
3	<i>Turkey</i>	324	<i>Paul</i>	653
4	<i>Canada</i>	262	<i>Mike</i>	586
...
95	<i>England</i>	37	<i>Sam</i>	86
96	<i>Austria</i>	37	<i>Roy</i>	84
97	<i>IL</i>	36	<i>Adams</i>	84
98	<i>Cambridge</i>	36	<i>Dale</i>	84
99	<i>Istanbul</i>	36	<i>JFIF</i>	82

Spacy NER:

- Firstly, we loaded the spacy library and passed each text from the cleaned dataset
- Secondly, we created the two lists with one containing the entity and the other list containing the corresponding labels
- Thirdly, we created the two lists with one containing the LOC and the other list containing the PERSON tags
- Fourthly, we created a frequency function which takes input a list and returns a dictionary which is sorted in the decreasing order of the number of occurrences
- Fifthly, we used frequency function to create the top 100 LOC and top 100 PERSON tags in the decreasing order of their number of occurrences in the 20newsgroup dataset
- Finally, we created a dataframe to display the top 100 LOC and top 100 PERSON tags in the structured format.
- Words with top 100 LOC and PERSON tags which are generated using spacy NER are in the below table,

	<i>Location</i>	<i>Location Frequency</i>	<i>Person</i>	<i>Person Frequency</i>
0	<i>Israel</i>	848	<i>Jesus</i>	1190
1	<i>US</i>	811	<i>David</i>	889
2	<i>Armenia</i>	409	<i>John</i>	851
3	<i>New</i>	406	<i>Paul</i>	598
4	<i>United</i>	380	<i>Clinton</i>	457
...
95	<i>Norway</i>	41	<i>Dick</i>	84
96	<i>London</i>	40	<i>Carl</i>	83
97	<i>Houston</i>	40	<i>Sam</i>	83
98	<i>Pluto</i>	40	<i>Tommy</i>	82
99	<i>Ohio</i>	40	<i>Randy</i>	81

Degree of correlation:

- Degree of correlation is the number of words matched between the top 100 words generated from both stanford NER and spacy NER divided by 100
- Degree of correlation between LOC tags from stanford NER and Spacy NER is **0.71**
- The matched words for location entity are,

	<i>Location Entity</i>	<i>Frequency_Stanford_NER</i>	<i>Frequency_Spacy_NER</i>
0	<i>Israel</i>	848	809
1	<i>US</i>	811	804
2	<i>Earth</i>	409	338
3	<i>Turkey</i>	406	324
4	<i>Canada</i>	380	262
...
66	<i>Italy</i>	44	39
67	<i>Cyprus</i>	43	38
68	<i>England</i>	41	37
69	<i>Cambridge</i>	40	36
70	<i>Istanbul</i>	40	36

- Degree of correlation between PERSON tags from stanford NER and Spacy NER is **0.75**
- The matched words for person entity are,

	<i>PERSON Entity</i>	<i>Frequency_Stanford_NER</i>	<i>Frequency_Spacy_NER</i>
0	<i>Jesus</i>	1190	1347
1	<i>John</i>	889	1058
2	<i>David</i>	851	996
3	<i>Paul</i>	598	653
4	<i>Mike</i>	457	586
...
70	<i>Larson</i>	86	87
71	<i>Bruce</i>	85	87
72	<i>Clayton</i>	84	86
73	<i>Sam</i>	84	86
74	<i>Adams</i>	83	84

Question - 1

Part c and Part d

Annotated dataset creation:

- In order to generate the annotated documents we used spacy NER and created the tags
- Later we created annotated dataset as list of lists with words of the sequences and corresponding labels in another list of lists
- We took a fixed length of 128 for each sequence in order to pass into the LSTM
- If a sequence in the annotated data doesn't have 128 words padding has been done and if it has more than 128 words then the remaining words are dropped
- We have 18 unique tags in the dataset.
- Now the problem is a multiclass classification problem with input as a sequence to LSTM.
- Output predicting each word in the sequence is a 18 dimensional vector and with each dimension giving the probability of the word belonging to that tag.
- The output of the LSTM will be a probability distribution among the 18 classes. Since we kept the maximum length to be 128. Therefore, for a single input we have 128x18 output i.e for every word in sequence we have probability that it belongs to 18 of NER classes.

Implementation:

1. Annotate data using Spacy NER.
2. Tokenize data and create vocabulary.
3. Fine Tune Embedding mentioned in question(if required)
4. Generate Embedding Matrix of using vocabulary dictionary for each word using Embedding model. Now, we have embedding matrix which have dimension

$$\text{dim(Embedding Matrix)} = (\text{vocab size} \times \text{Embedding Vector size})$$

5. Now, prepare a LSTM model having the first layer as an Embedding layer with the same size as that of Embedding Matrix. Feed Embedding Matrix as weights of Embedding Layer. Here, we are also learning/changing Embedding Vectors for each word in vocab.
6. Final Classification layer of the model is the Time distributed dense layer with softmax as activation function.
7. Train Model and Evaluate model for different metrics.

Word2vec embeddings:

- Word2vec represents each distinct word with a particular list of numbers called a vector.
- For training purposes we used a gensim module.
- Here we used CBOW model of word2vec to generate the embeddings

GloVe embeddings:

- GloVe is an unsupervised learning algorithm for obtaining vector representations for words.
- Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.
- Firstly, we loaded the pre-trained embeddings downloaded
- Secondly, we create the list of words which are not in the vocabulary
- Thirdly, we create the countvectorizer for the out of the word vocabulary
- Fourthly, co occurrence matrix for out of the vocabulary vector
- Fifthly, we used the Mittens library and passed out of the word vocabulary embeddings, pre-trained embeddings and co occurrence matrix as parameters to the mittens model to get new finetunes embeddings. Hyper parameter 500 iterations to get vector rep of 50
- Sixthly, we use this fine tuned embeddings accordingly

FastText embeddings:

- FastText is a library for learning of word embeddings and text classification
- The model allows one to create an unsupervised learning or supervised learning algorithm for obtaining vector representations for words.

BERT embeddings:

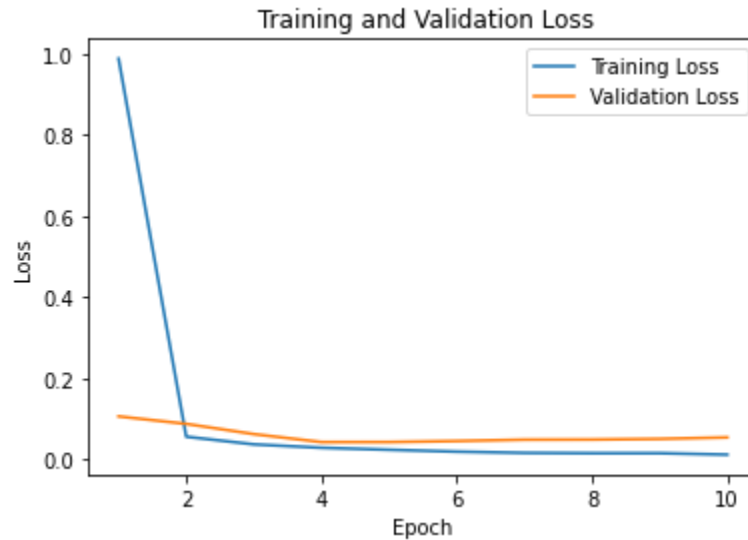
- It contains three embeddings for each word which are, token embeddings, sentence embeddings, and positional embeddings

LSTM Architecture:

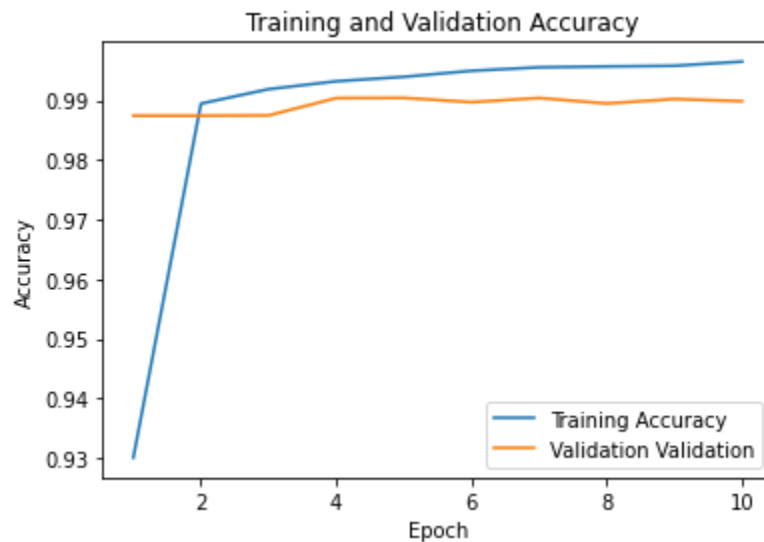
Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 128)]	0
embedding_1 (Embedding)	(None, 128, 50)	533350
lstm_1 (LSTM)	(None, 128, 128)	91648
dropout (Dropout)	(None, 128, 128)	0
batch_normalization (BatchNo	(None, 128, 128)	512
time_distributed_1 (TimeDist	(None, 128, 19)	2451

Analysis:

1. LSTM with Word2vec embeddings:



- The above graph shows the training and the validation loss curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding loss for both training and validation
- We achieved a training loss of 0.012 and validation loss of 0.054

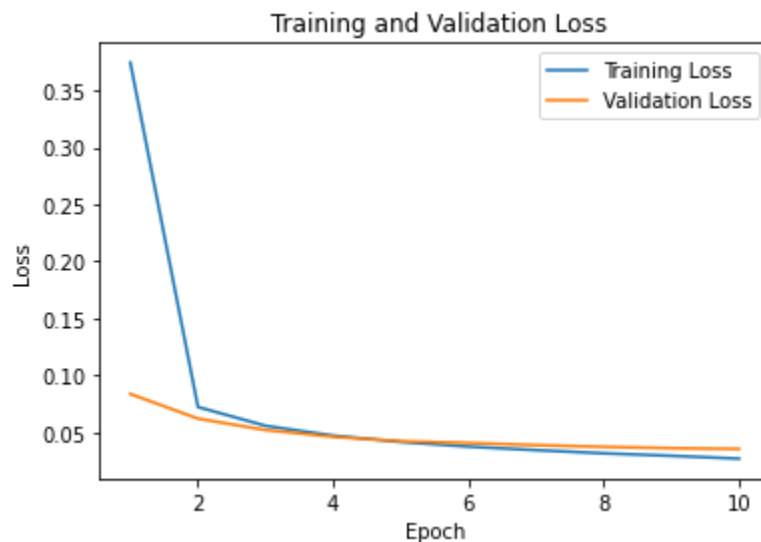


- The above graph shows the training and the validation accuracy curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding accuracy for both training and validation
- We can see that we reached 99.5% of training accuracy and 98.8% of validation accuracy

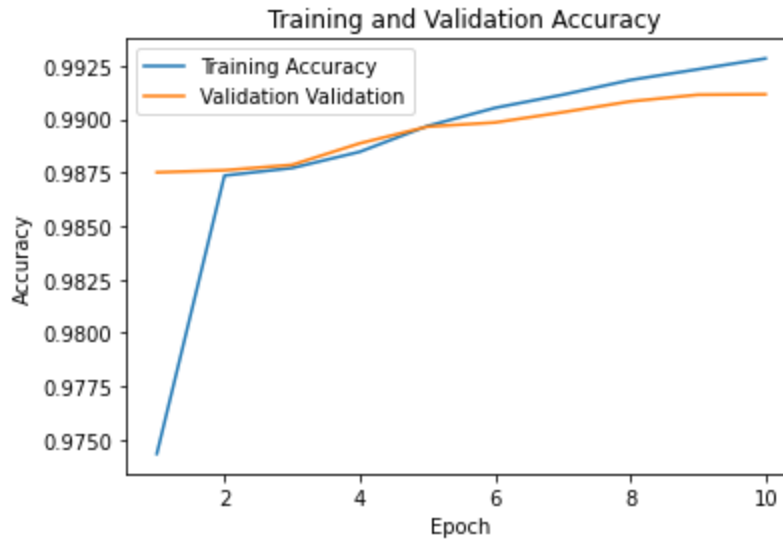
	precision	recall	f1-score	support
CARDINAL	0.67	0.56	0.61	177
DATE	0.72	0.31	0.44	249
EVENT	0.00	0.00	0.00	19
FAC	0.27	0.08	0.12	52
GPE	0.65	0.47	0.55	289
LANGUAGE	0.00	0.00	0.00	0
LAW	0.44	0.19	0.26	37
LOC	0.65	0.50	0.57	110
MONEY	0.00	0.00	0.00	11
NORP	0.77	0.64	0.69	181
O	0.95	0.98	0.96	33941
ORDINAL	0.73	0.88	0.80	58
ORG	0.43	0.32	0.37	1074
PERSON	0.73	0.40	0.52	836
PRODUCT	0.18	0.02	0.03	156
QUANTITY	0.00	0.00	0.00	21
TIME	0.52	0.32	0.39	47
WORK_OF_ART	0.56	0.16	0.25	57
accuracy			0.93	37315
macro avg	0.46	0.32	0.36	37315
weighted avg	0.91	0.93	0.92	37315

- The above figure is the classification report
- We can infer that the accuracy is 93%

2. LSTM with GloVe embeddings:



- The above graph shows the training and the validation loss curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding loss for both training and validation
- We achieved a training loss of 0.026 and validation loss of 0.035

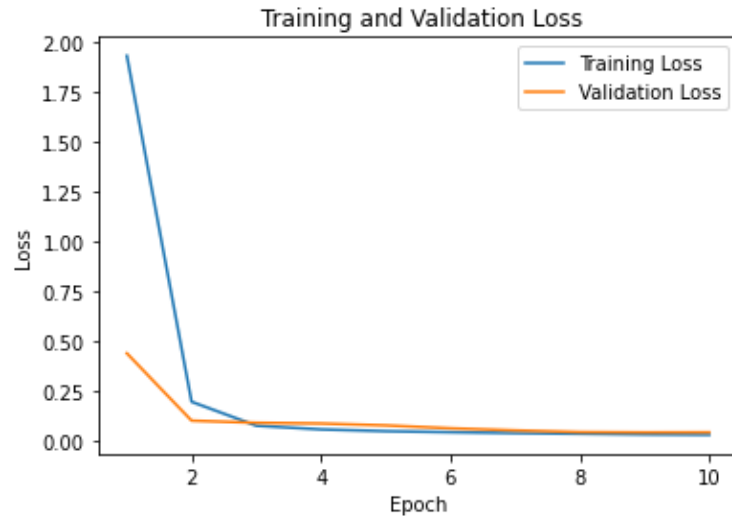


- The above graph shows the training and the validation accuracy curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding accuracy for both training and validation
- We can see that we reached 99.28% of training accuracy and 99.12% of validation accuracy

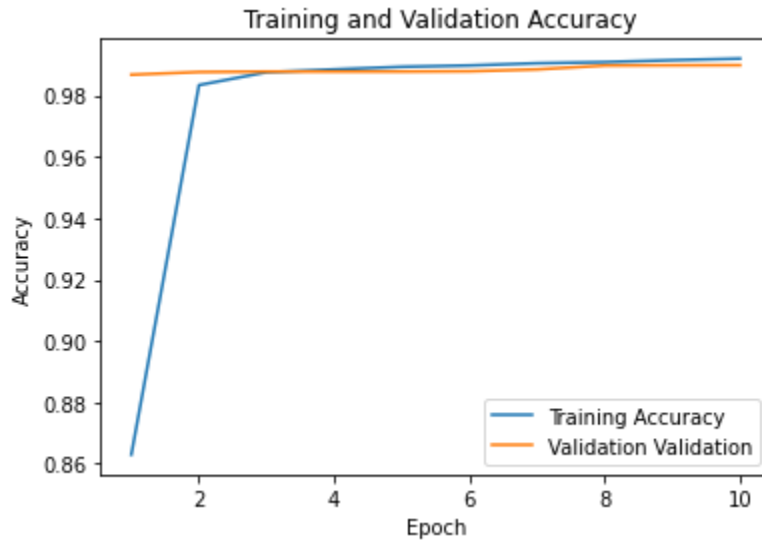
	precision	recall	f1-score	support
CARDINAL	0.68	0.63	0.66	95
DATE	0.64	0.53	0.58	136
EVENT	0.00	0.00	0.00	15
FAC	0.00	0.00	0.00	20
GPE	0.63	0.61	0.62	122
LANGUAGE	0.00	0.00	0.00	1
LAW	0.00	0.00	0.00	17
LOC	0.68	0.55	0.61	58
MONEY	0.00	0.00	0.00	9
NORP	0.77	0.82	0.79	68
O	0.96	0.98	0.97	15067
ORDINAL	0.89	0.89	0.89	37
ORG	0.50	0.36	0.42	513
PERCENT	0.00	0.00	0.00	0
PERSON	0.62	0.56	0.59	328
PRODUCT	0.00	0.00	0.00	21
QUANTITY	0.00	0.00	0.00	5
TIME	0.00	0.00	0.00	35
WORK_OF_ART	0.00	0.00	0.00	30
accuracy			0.93	16577
macro avg	0.34	0.31	0.32	16577
weighted avg	0.92	0.93	0.93	16577

- The above figure is the classification report
- We can infer that the accuracy is 93%

3. LSTM with FastText embeddings:



- The above graph shows the training and the validation loss curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding loss for both training and validation
- We achieved a training loss of 0.032 and validation loss of 0.044

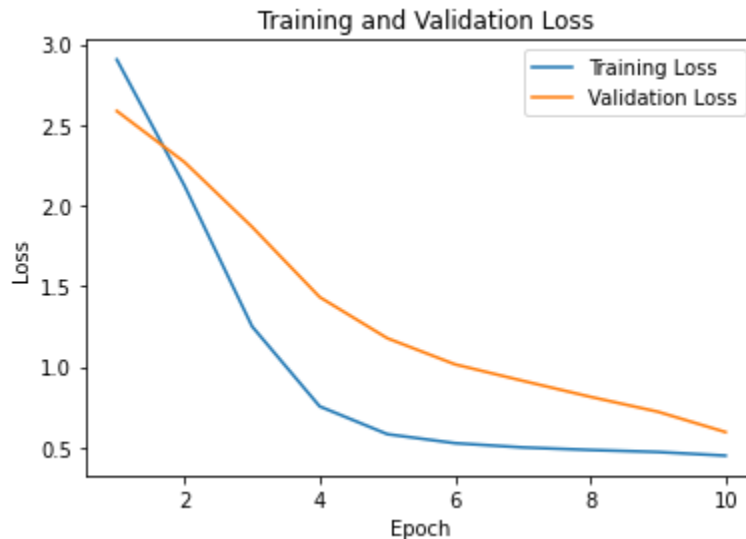


- The above graph shows the training and the validation accuracy curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding accuracy for both training and validation
- We can see that we reached 99.20% of training accuracy and 98.99% of validation accuracy

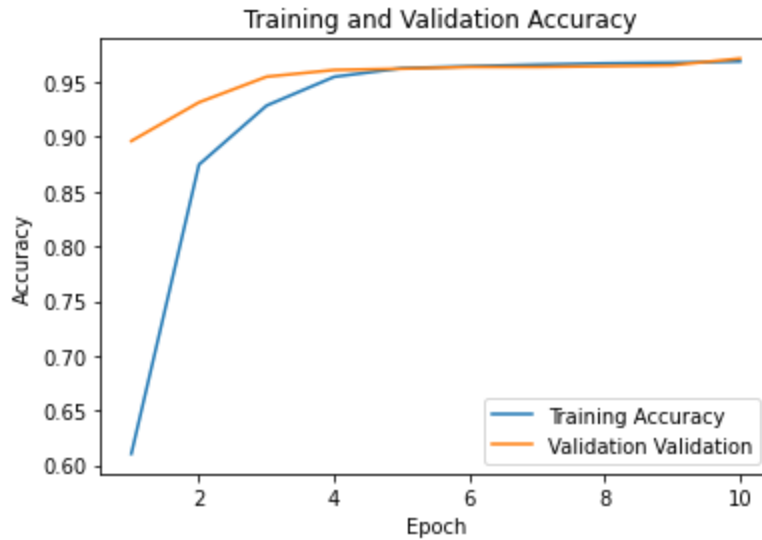
	precision	recall	f1-score	support
CARDINAL	0.70	0.32	0.44	94
DATE	0.73	0.37	0.49	189
EVENT	1.00	0.20	0.33	10
FAC	0.00	0.00	0.00	27
GPE	0.73	0.45	0.55	101
LANGUAGE	0.00	0.00	0.00	2
LAW	0.00	0.00	0.00	5
LOC	0.98	0.56	0.71	73
MONEY	0.00	0.00	0.00	11
NORP	0.82	0.54	0.65	78
O	0.94	0.99	0.97	15719
ORDINAL	0.83	0.73	0.78	26
ORG	0.46	0.16	0.23	463
PERSON	0.59	0.35	0.44	382
PRODUCT	0.00	0.00	0.00	27
QUANTITY	0.00	0.00	0.00	6
TIME	0.73	0.24	0.36	34
WORK_OF_ART	0.67	0.25	0.36	8
accuracy			0.93	17255
macro avg	0.51	0.29	0.35	17255
weighted avg	0.91	0.93	0.92	17255

- The above figure is the classification report
- We can infer that the accuracy is 93%

4. LSTM with BERT embeddings:



- The above graph shows the training and the validation loss curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding loss for both training and validation
- We achieved a training loss of 0.449 and validation loss of 0.595



- The above graph shows the training and the validation accuracy curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding accuracy for both training and validation
- We can see that we reached 96.8% of training accuracy and 97.17% of validation accuracy

	precision	recall	f1-score	support
CARDINAL	0.42	0.50	0.46	16
DATE	0.25	0.48	0.32	52
EVENT	0.08	0.40	0.14	5
GPE	0.40	0.68	0.50	31
LOC	0.60	0.62	0.61	40
MONEY	0.00	0.00	0.00	4
NORP	0.24	0.67	0.35	9
O	0.96	0.82	0.88	3150
ORDINAL	0.80	0.80	0.80	5
ORG	0.11	0.40	0.17	80
PERSON	0.26	0.59	0.36	56
PRODUCT	0.00	0.00	0.00	6
TIME	0.21	0.43	0.29	7
WORK_OF_ART	0.00	0.00	0.00	3
accuracy			0.79	3464
macro avg	0.31	0.46	0.35	3464
weighted avg	0.89	0.79	0.83	3464

- The above figure is the classification report
- We can infer that the accuracy is 79%

Comparison:

- The below table shows the comparison of different embedding methods on the LSTM designed for this question to generate the NERs,

S.NO	DL Architecture	Embeddings	Accuracy
1	LSTM	Word2vec	93%
2	LSTM	GloVe	93%
3	LSTM	Fasttext	93%
4	LSTM	BERT embeddings	79%

- The three embeddings Word2vec, GloVe, and Fasttext will generate the embeddings of only one vector for each word
- BERT will generate 3 different embedding vectors, but we used only the token embeddings for passing to LSTM architecture
- We can infer that instead of using the LSTM architecture for BERT embedding, use of BERT will give more accuracy

Question - 2

We used the Rasa framework to design the chatbot. Rasa internally uses Tensorflow
In the Rasa framework we have a NLU and a Rasa Core.

1. Rasa NLU:

- It is a library for natural language understanding with intent classification and entity extraction. This helps the chatbot to understand what the user is saying.
- This is the place where rasa tries to understand User messages to detect Intent and Entity in your message.
- It has different components for recognizing intents and entities, most of which have some additional dependencies.

2. Rasa Core:

- It is a chatbot framework with machine learning-based dialogue management that predicts the next best action based on the input from NLU, the conversation history, and the training data.
- This is the place where Rasa tries to help you with contextual message flow. Based on User message, it can predict dialogue as a reply and can trigger Rasa Action Server.

Rasa Configuration:

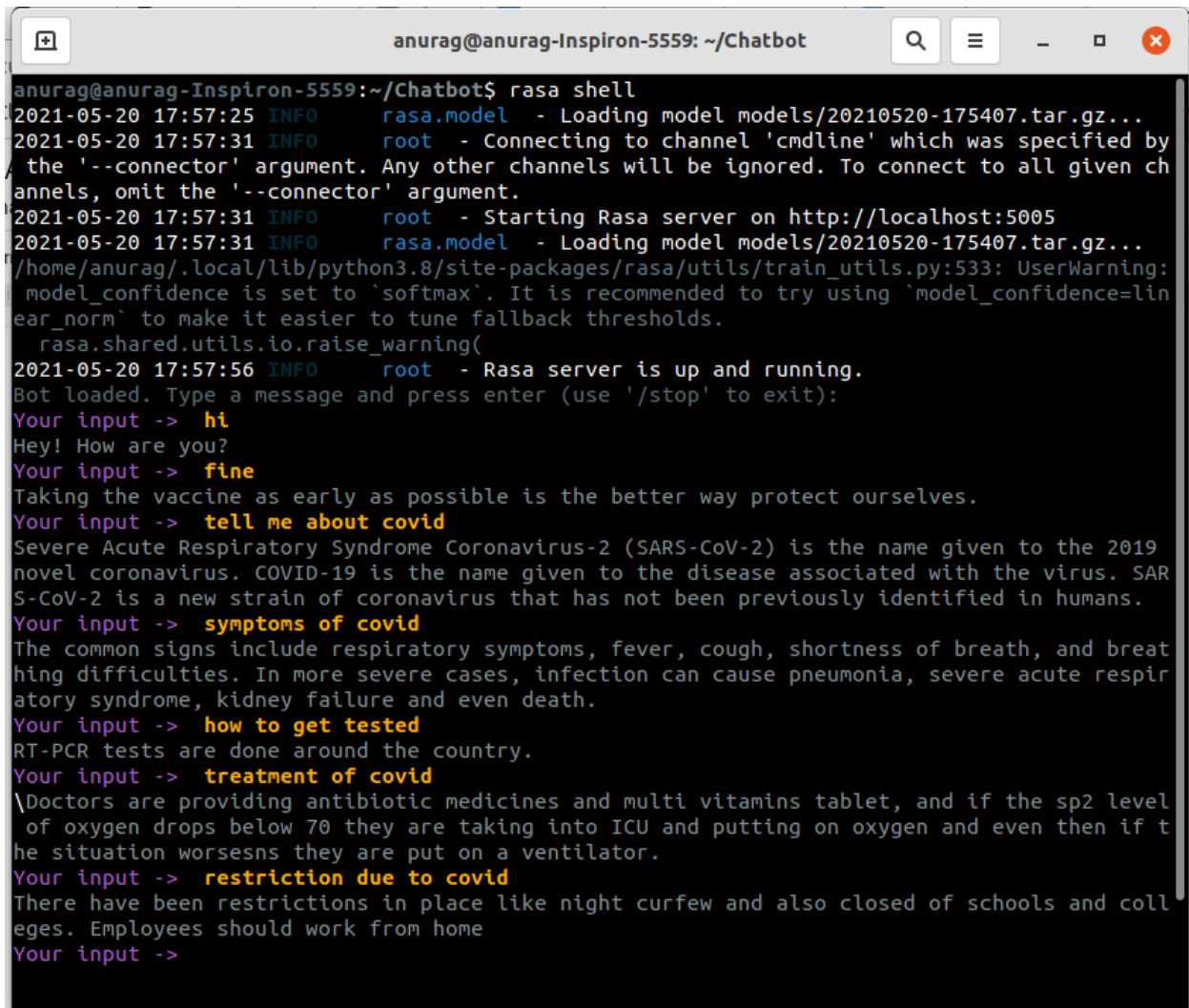
1. **actions.py** file is generated to code for your custom actions. In-case you want Rasa to call an external server via REST API or API call, you can define your Custom Actions here. Remember you can create multiple Python Script for Rasa Custom Action.
2. **nlu.yml** file is generated for your NLU training data. Here you can define Intent. Like Order Pizza or Book Uber. You need to add related Sentences for that Intent. Remember if you are using Rasa-X, your training Intent and Data will be added automatically.
3. **stories.yml** file is generated to make your stories. This is required for Rasa Core. There is something called “Dialog Flow in Rasa” where Rasa Core controls the flow of the conversation between you and the chatbot, so for that flow, you need to train the chatbot using these stories. So in case you want your chatbot to be very perfect in different contexts (stories) you can add those stories here.
4. **domain.yml** file is generated for your assistant’s domain. This file combines Different Intent which a chatbot can detect and a list of Bot replies. Remember you can define your Custom Action Server Python method name here (in underscore format), so that Rasa will call that python method for you.
5. **credentials.yml** file is generated to get the details for connecting to other services. In case you want to build a Bot on Facebook Messenger, Microsoft Bot Framework, you can maintain such credentials and tokens here. So basically you just need to add Facebook, slack and Bot framework related configuration, rasa will automatically do rest for you.

6. **config.yml** file is generated for configuration of your NLU and Core models. In-case you are dealing with Tensorflow or Spacy, you need to define such a pipeline here. To handle this file, you show knowledge about Machine Learning and Deep Learning.

The Rasa NLU was trained for 100 epochs and we got the training loss of 2.44 and accuracy of 92.3%. The loss returned is the perplexity, which is the exponential of cross entropy loss. The lower the perplexity the better the model is.

Some examples of Q&A with Rasa:

Example 1:



```
anurag@anurag-Inspiron-5559: ~/Chatbot
anurag@anurag-Inspiron-5559:~/Chatbot$ rasa shell
2021-05-20 17:57:25 INFO     rasa.model - Loading model models/20210520-175407.tar.gz...
2021-05-20 17:57:31 INFO     root - Connecting to channel 'cmdline' which was specified by
the '--connector' argument. Any other channels will be ignored. To connect to all given ch
annels, omit the '--connector' argument.
2021-05-20 17:57:31 INFO     root - Starting Rasa server on http://localhost:5005
2021-05-20 17:57:31 INFO     rasa.model - Loading model models/20210520-175407.tar.gz...
/home/anurag/.local/lib/python3.8/site-packages/rasa/utils/train_utils.py:533: UserWarning:
model_confidence is set to 'softmax'. It is recommended to try using 'model_confidence=lin
ear_norm' to make it easier to tune fallback thresholds.
  rasa.shared.utils.io.raise_warning(
2021-05-20 17:57:56 INFO     root - Rasa server is up and running.
Bot loaded. Type a message and press enter (use '/stop' to exit):
Your input -> hi
Hey! How are you?
Your input -> fine
Taking the vaccine as early as possible is the better way protect ourselves.
Your input -> tell me about covid
Severe Acute Respiratory Syndrome Coronavirus-2 (SARS-CoV-2) is the name given to the 2019
novel coronavirus. COVID-19 is the name given to the disease associated with the virus. SAR
S-CoV-2 is a new strain of coronavirus that has not been previously identified in humans.
Your input -> symptoms of covid
The common signs include respiratory symptoms, fever, cough, shortness of breath, and breat
hing difficulties. In more severe cases, infection can cause pneumonia, severe acute respir
atory syndrome, kidney failure and even death.
Your input -> how to get tested
RT-PCR tests are done around the country.
Your input -> treatment of covid
Doctors are providing antibiotic medicines and multi vitamins tablet, and if the sp2 level
of oxygen drops below 70 they are taking into ICU and putting on oxygen and even then if t
he situation worsens they are put on a ventilator.
Your input -> restriction due to covid
There have been restrictions in place like night curfew and also closed of schools and coll
eges. Employees should work from home
Your input ->
```

- Here, by analysing the conversation we depict that since chatbot is trained on COVID data therefore chatbot replies are biased towards providing covid information.
- Example in above conversation when we say 'hi'. Chatbot replies 'how are you?' and we replied fine, then the chatbot advised us to take the vaccine as early as possible.

Example 2:

```
anurag@anurag-Inspiron-5559: ~/Chatbot
anurag@anurag-Inspiron-5559:~/Chatbot$ rasa shell
2021-05-20 19:00:30 INFO     rasa.model - Loading model models/20210520-185338.tar.gz...
2021-05-20 19:00:34 INFO     root - Connecting to channel 'cmdline' which was specified by
the '--connector' argument. Any other channels will be ignored. To connect to all given ch
annels, omit the '--connector' argument.
2021-05-20 19:00:34 INFO     root - Starting Rasa server on http://localhost:5005
2021-05-20 19:00:34 INFO     rasa.model - Loading model models/20210520-185338.tar.gz...
/home/anurag/.local/lib/python3.8/site-packages/rasa/utils/train_utils.py:533: UserWarning:
model_confidence is set to 'softmax'. It is recommended to try using 'model_confidence=lin
ear_norm' to make it easier to tune fallback thresholds.
  rasa.shared.utils.io.raise_warning(
2021-05-20 19:00:57 INFO     root - Rasa server is up and running.
Bot loaded. Type a message and press enter (use '/stop' to exit):
Your input -> Hi
Hey! How are you?
Your input -> fine
Taking the vaccine as early as possible is the better way protect ourselves.
Your input -> symptoms of covid
The common signs include respiratory symptoms, fever, cough, shortness of breath, and breat
hing difficulties. In more severe cases, infection can cause pneumonia, severe acute respir
atory syndrome, kidney failure and even death.
Your input -> affect of covid on children
You are less likely to be affected but once got affected it is dangerous
Your input -> affect of covid on adults
You are more likely to be affected, so following the gudielines issued by your government w
ould be a safest option
Your input -> affect of covid on animal
These are less likely to get affected by covid-19
Your input -> effect of covid on India
The virus has became more dangerous and fastly spreading with new starins and mutations.
Your input -> effect of covid on western countries
The virus has mostly been curbed now(April 2021 onwards) because of vaccination but in the
earlier stage around March 2020 - October 2020 was desastrous for yor country
Your input -> bye
Bye
Your input ->
```

- Here, by analysing the conversation we depict that chatbot is able to identify entities correctly.
- Here we ask the same question to chatbot by changing entity name.
- Chatbot will be able to identify that entity and be able to give sensible answers.

Question - 3

Bi-LSTM:

- The Bi-LSTM architecture used for this assignment is,

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 1641, 50)	2378050
bidirectional (Bidirectional)	(None, 1641, 40)	11360
dropout (Dropout)	(None, 1641, 40)	0
batch_normalization (Batch Normalization)	(None, 1641, 40)	160
bidirectional_1 (Bidirectional)	(None, 1641, 40)	9760
dropout_1 (Dropout)	(None, 1641, 40)	0
batch_normalization_1 (Batch Normalization)	(None, 1641, 40)	160
bidirectional_2 (Bidirectional)	(None, 40)	9760
dropout_2 (Dropout)	(None, 40)	0
batch_normalization_2 (Batch Normalization)	(None, 40)	160
dense (Dense)	(None, 64)	2624
dense_1 (Dense)	(None, 64)	4160
dense_2 (Dense)	(None, 1)	65
Total params: 2,416,259		
Trainable params: 2,416,019		
Non-trainable params: 240		

Word2vec embeddings:

- Word2vec represents each distinct word with a particular list of numbers called a vector.
- For training purposes we used a gensim module.
- Here we used CBOW model of word2vec to generate the embeddings

GloVe embeddings:

- GloVe is an unsupervised learning algorithm for obtaining vector representations for words.
- Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.
- Firstly, we loaded the pre-trained embeddings downloaded
- Secondly, we create the list of words which are not in the vocabulary

- Thirdly, we create the countvectorizer for the out of the word vocabulary
- Fourthly, co occurrence matrix for out of the vocabulary vector
- Fifthly, we used the Mitten library and passed out of the word vocabulary embeddings, pre-trained embeddings and co occurrence matrix as parameters to the mittens model to get new finetunes embeddings. Hyper parameter 500 iterations to get vector rep of 50
- Sixthly, we use this fine tuned embeddings accordingly

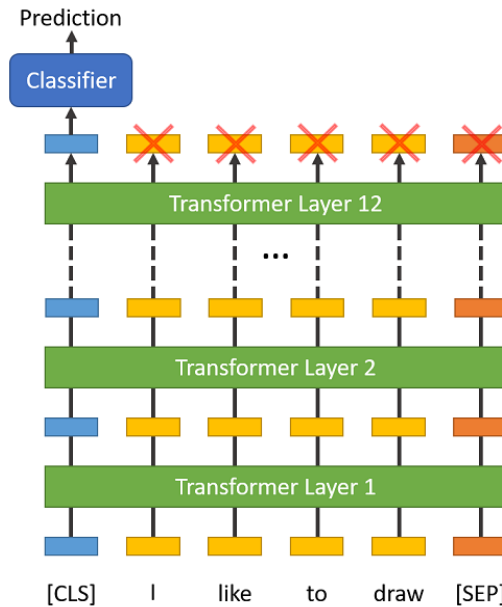
TF-IDF embeddings:

- TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.
- This is done by multiplying two metrics, how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.
- TF-IDF will create the size of the each vector as the size of the dictionary of unique words, because of this the matrix weight matrix generated will be a sparse matrix
- For the given corpus the dictionary of unique words is of roughly 90,000 size
- We limited the length of the each embedding vector to 500 and processed accordingly.

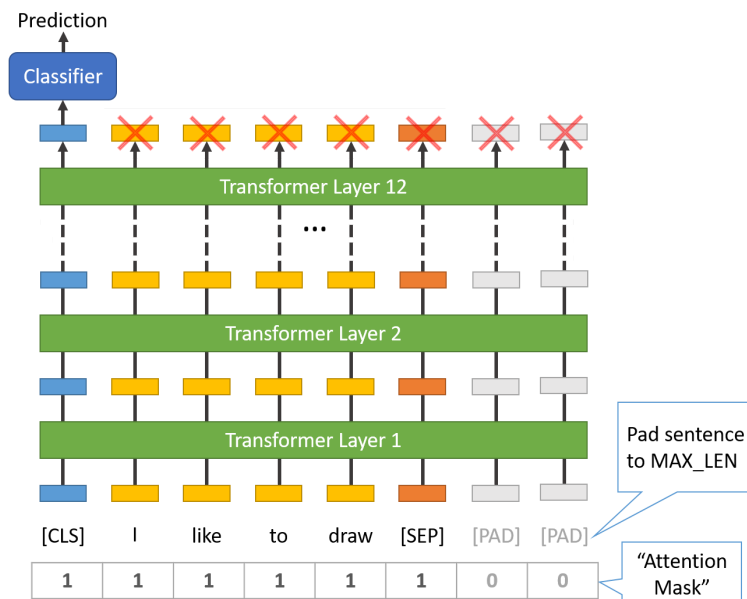
BERT:

- BERT (Bidirectional Encoder Representations from Transformers) is a method of pretraining language representations that was used to create models
- We have taken the pre-trained BERT model, added an untrained layer of neurons on the end, and trained the new model for our classification task
- The pre-trained BERT model weights already encode a lot of information about our language
- As a result, it takes much less time to train our fine-tuned model - it is as if we have already trained the bottom layers of our network extensively and only need to gently tune them while using their output as features for our classification task
- The pre-trained weights this method allows us to fine-tune our task on a much smaller dataset than would be required in a model that is built from scratch
- BERT fine tuning on our dataset:
 1. We used the transformers package from Hugging Face which gave us a pytorch interface for working with BERT
 2. To feed our text to BERT, it must be split into tokens, and then these tokens must be mapped to their index in the tokenizer vocabulary
- BERT tokenization:
 1. Add special tokens to the start and end of each sentence. At the end of every sentence, we need to append the special [SEP] token. This token is an artifact of two-sentence tasks, where BERT is given two separate sentences and asked to determine something

- For classification tasks, we must prepend the special [CLS] token to the beginning of every sentence. This token has special significance. BERT consists of 12 Transformer layers. Each transformer takes in a list of token embeddings, and produces the same number of embeddings on the output



- BERT has two constraints, all sentences must be padded or truncated to a single, fixed length and the maximum sentence length is 512 tokens.
 - Padding is done with a special [PAD] token, which is at index 0 in the BERT vocabulary
 - The below illustration demonstrates padding out to a "MAX_LEN" of 8 tokens



- The "Attention Mask" is simply an array of 1s and 0s indicating which tokens are padding and which aren't
 - This mask tells the "Self-Attention" mechanism in BERT not to incorporate these PAD tokens into its interpretation of the sentence
- Training on our BERT model:
 1. We first want to modify the pre-trained BERT model to give outputs for classification, and then we want to continue training the model on our dataset
 2. Huggingface pytorch implementation includes a set of interfaces designed for a variety of NLP tasks
 3. Though these interfaces are all built on top of a trained BERT model, each has different top layers and output types designed to accommodate their specific NLP task
 4. We'll be using BertForSequenceClassification which is the normal BERT model with an added single linear layer on top for classification that we will use as a sentence classifier
 5. As we feed input data, the entire pre-trained BERT model and the additional untrained classification layer is trained on our specific task

Implementation:

1. Tokenize data and create vocabulary.
2. Fine Tune Embedding mentioned in question(if required)
3. Generate Embedding Matrix of using vocabulary dictionary for each word using Embedding model. Now, we have embedding matrix which have dimension

$$\text{dim(Embedding Matrix)} = (\text{vocab size} \times \text{Embedding Vector size})$$

4. Now , prepare a LSTM model having the first layer as an Embedding layer with the same size as that of Embedding Matrix. Feed Embedding Matrix as weights of Embedding Layer. Here, we are also learning/changing Embedding Vectors for each word in vocab.
5. Final Classification layer of the model is the dense layer with softmax as activation function.
6. Train Model and Evaluate model for different metrics.

Hyperparameters:

1. Batch size is 32,
2. Learning rate is 0.00001
3. Number of epochs are 4

Optimizer and loss function:

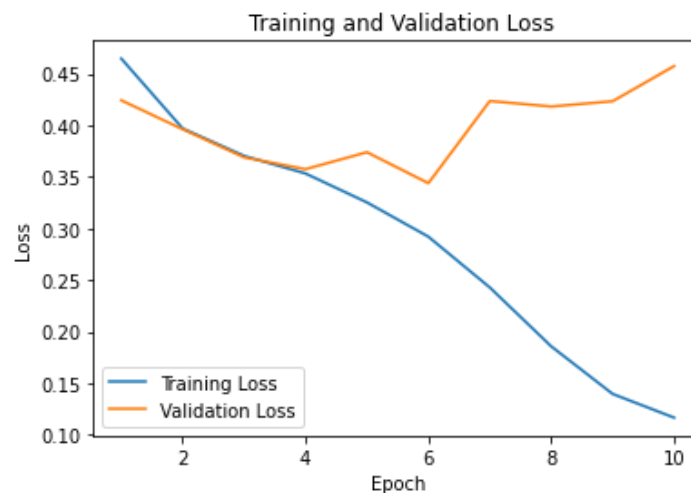
We used Adam optimiser and the loss is classification loss.

Metrics used to evaluate:

1. Precision:
 - It is the parameter that determines how close is the output from the model to that of the original output
 - In our case for the whole test dataset, how many labels are predicted true which are actually true and predicted false which are actually false
2. Recall:
 - This is metric which measures how many actual true labels are predicted as true labels by the model
3. Confusion matrix:
 - It is a metric used to measure the quality for binary classification and multi class classification as well
 - It is summary of the predictions made by the model on a classification problem
 - We can interpret the precision, recall, and accuracy from the confusion matrix
4. Matthews correlation coefficient (MCC):
 - MCC is used to measure the quality of binary or multi class classification
 - It can be used as a balanced measure when we have the classes which are of different sizes because it takes into account of true positives, true negatives, false positives, and false negatives
 - The MCC is in essence a correlation coefficient value between -1 and +1.
 - A coefficient of +1 represents a perfect prediction, 0 an average random prediction and -1 an inverse prediction.

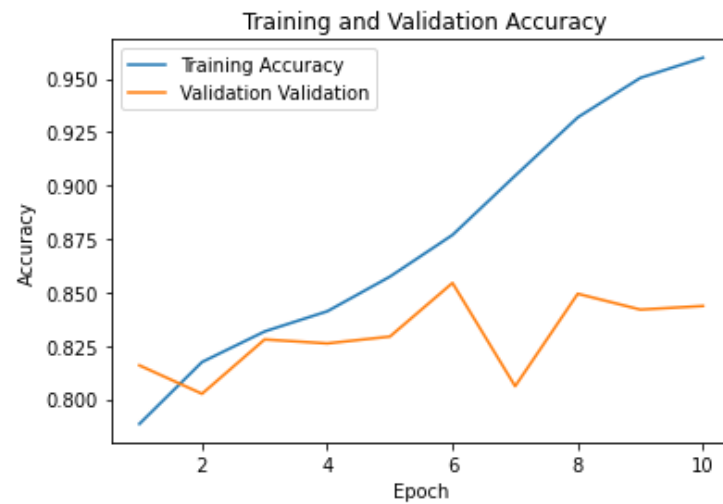
Results and Analysis:

1. Bi-LSTM with word2vec embedding:

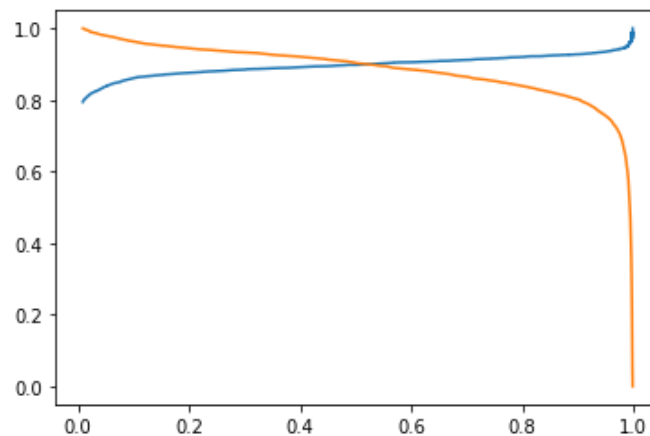


- The above graph shows the training and the validation loss curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding loss for both training and validation

- We trained the model for 10 epochs because we can clearly infer from the above figure that the model overfits after 6 epochs as validation loss starts increasing



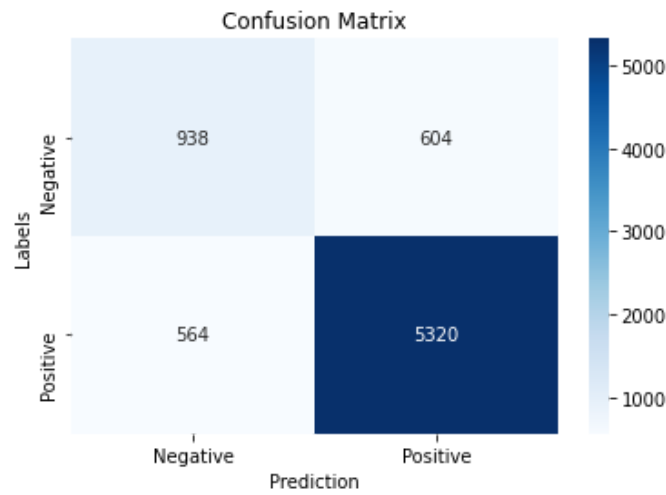
- The above graph shows the training and the validation accuracy curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding accuracy for both training and validation
- We can see that we reached 87.5% of training accuracy and 85.16% of validation accuracy after 6 epochs



- The above figure is the Recall-precision crossover
- The blue line represents precision
- The orange line represents recall
- At the crossover point is at 0.52 and recall is 0.90 at that point

	precision	recall	f1-score	support
0	0.62	0.62	0.62	1542
1	0.90	0.90	0.90	5884
accuracy			0.84	7426
macro avg	0.76	0.76	0.76	7426
weighted avg	0.84	0.84	0.84	7426

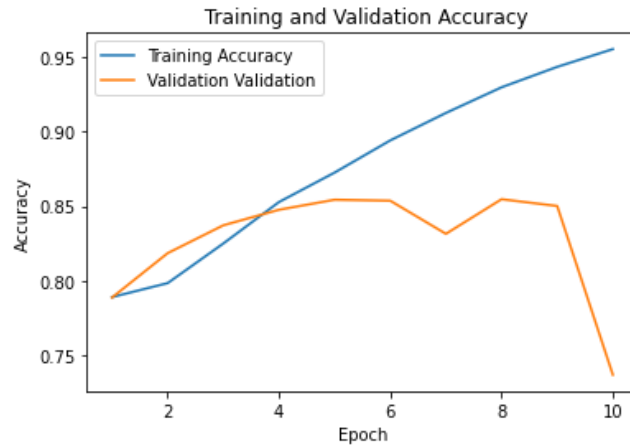
- The above figure shows the classification report for the model evaluated
- We can infer that the accuracy is 84%



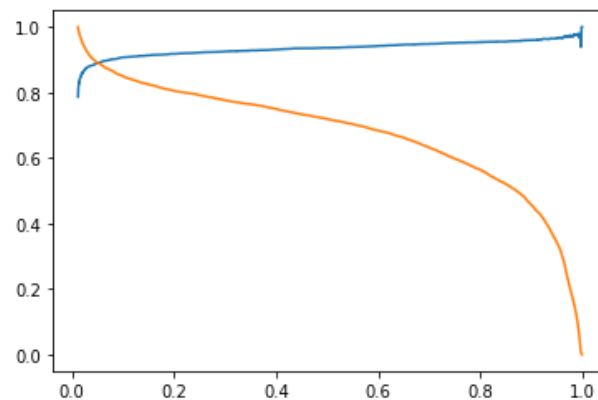
2. Bi-LSTM with glove embeddings:



- The above graph shows the training and the validation loss curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding loss for both training and validation
- We trained the model for 10 epochs because we can clearly infer from the above figure that the model overfits after 6 epochs as validation loss starts increasing



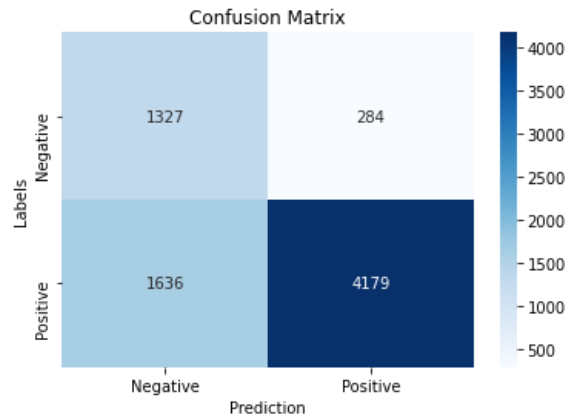
- The above graph shows the training and the validation accuracy curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding accuracy for both training and validation
- We can see that we reached 89.65% of training accuracy and 85.37% of validation accuracy after 6 epochs



- The above figure is the Recall-precision crossover
- The blue line represents precision
- The orange line represents recall
- At the crossover point is at 0.05 and recall is 0.84 at that point

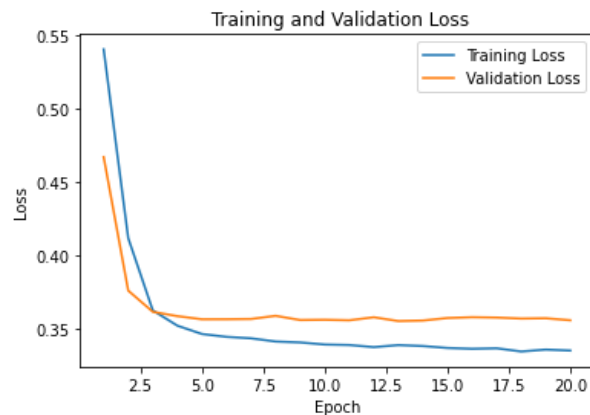
	precision	recall	f1-score	support
0	0.60	0.60	0.60	1611
1	0.89	0.89	0.89	5815
accuracy			0.83	7426
macro avg	0.75	0.75	0.75	7426
weighted avg	0.83	0.83	0.83	7426

- The above figure shows the classification report for the model evaluated
- We can infer that the accuracy is 83%

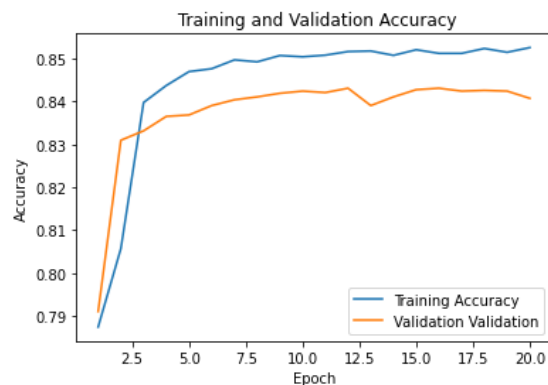


3. Bi-LSTM with TF-IDF embeddings:

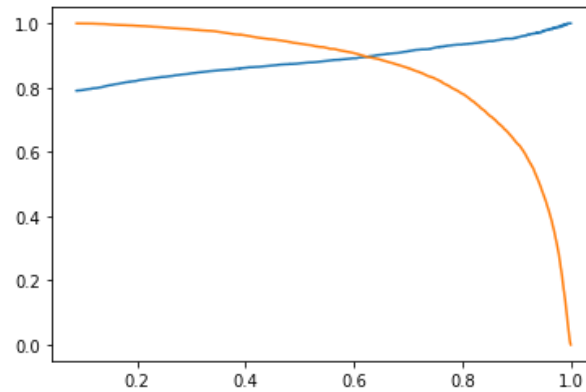
- We have completed TF-IDF before it is scraped
- In particular, instead of the word TF-IDF produced document embedding, we fed document embedding to the LSTM Model
- Since, TF-IDF follows BoW model it does not make any sense to feed it to LSTM since sequence information is lost. Here LSTM simply performs classification with no relevant sequence information.



- The above graph shows the training and the validation loss curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding loss for both training and validation



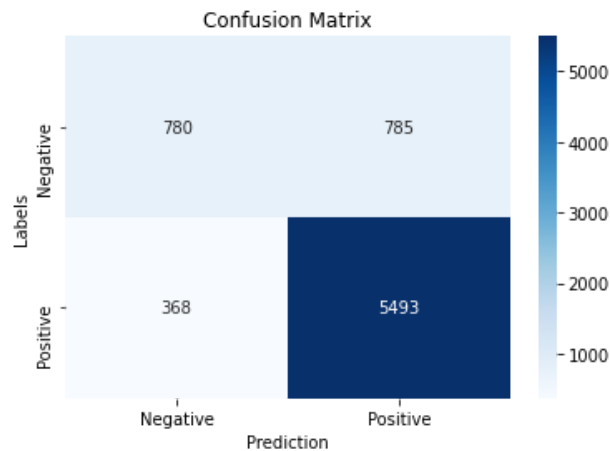
- The above graph shows the training and the validation accuracy curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding accuracy for both training and validation
- We can see that we attained 85.19% of training accuracy and 84.07% of validation accuracy



- The above figure is the Recall-precision crossover
- The blue line represents precision
- The orange line represents recall
- At the crossover point is at 0.62 and recall is 0.90 at that point

	precision	recall	f1-score	support
0	0.61	0.61	0.61	1565
1	0.90	0.90	0.90	5861
accuracy			0.84	7426
macro avg	0.75	0.75	0.75	7426
weighted avg	0.84	0.84	0.84	7426

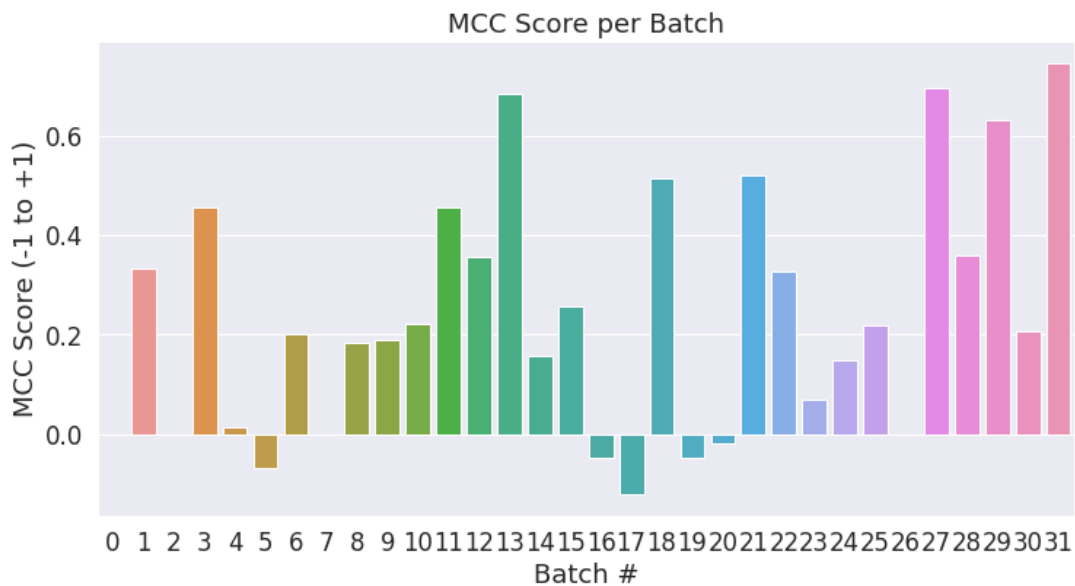
- The above figure shows the classification report for the model evaluated
- We can infer that the accuracy is 84%



4. BERT based Sentiment Analysis



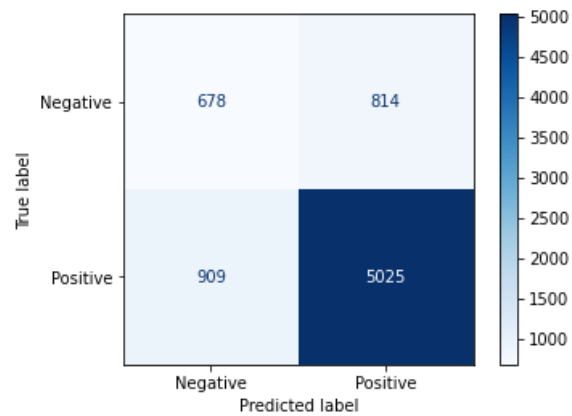
- The above graph shows the training and the validation loss curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding loss for both training and validation
- We trained the model for 4 epochs because we can clearly infer from the above figure that the model overfits after running for just 1 epoch as validation loss increased



- The above figure shows the Matthews correlation coefficients score per batch
- X-axis represents the number of batches
- Y-axis represents the corresponding MCC score for each batch

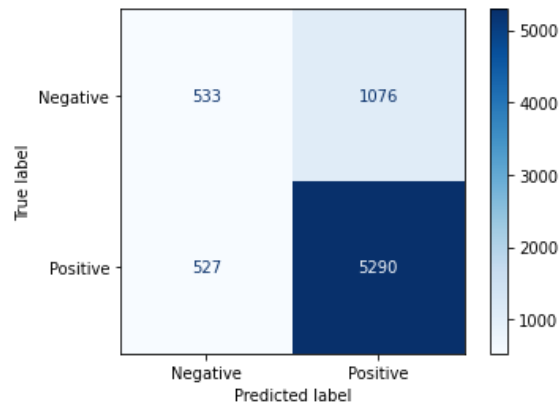
5. Naive bayes with Word2vec:

- F1 score attained is 0.85
- Accuracy attained is 76.8%



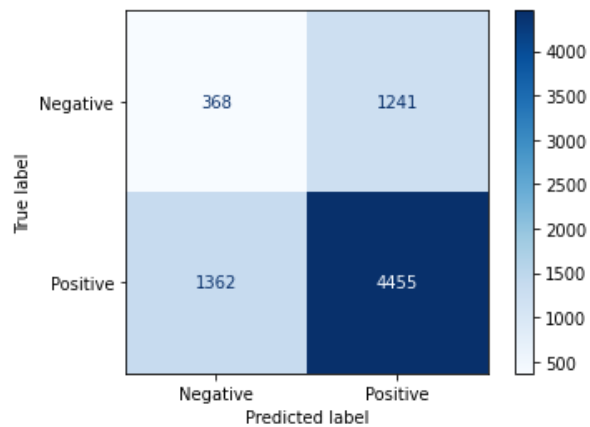
6. Naive bayes with GloVe:

- F1 score attained is 0.87
- Accuracy attained is 78.41%



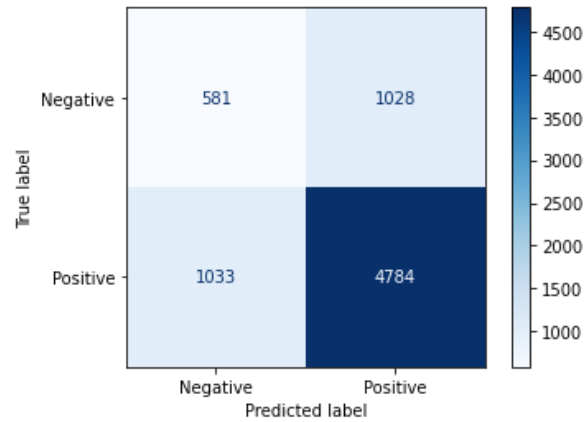
7. Decision Tree with Word2vec:

- F1 score attained is 0.77
- Accuracy attained is 64.95%



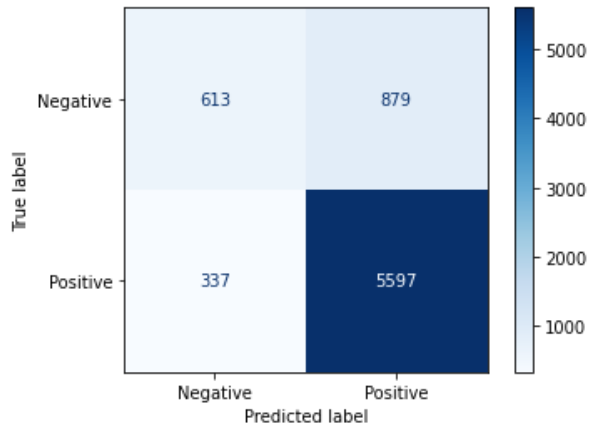
8. Decision Tree with GloVe:

- F1 score attained is 0.82
- Accuracy attained is 72.25%



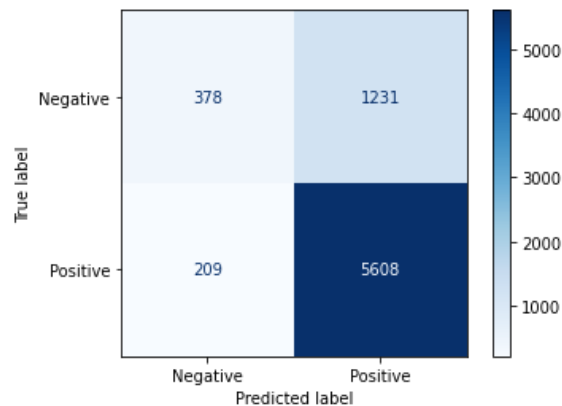
9. Logistic regression with Word2vec:

- F1 score attained is 0.9
- Accuracy attained is 83.63%



10. Logistic regression with GloVe:

- F1 score attained is 0.89
- Accuracy attained is 80.61%



Comparison:

- The below table shows the comparison of

S.NO	DL Architecture	Embeddings	Accuracy
1	Bi-LSTM	Word2vec	84%
2	Bi-LSTM	GloVe	83%
3	Bi-LSTM	TF-IDF	84%
4	BERT	BERT embeddings	85%

- The below table is comparison of the Bi-LSTM models with that of the assignment-1 models,

S.NO	ML Techniques	Vector Space Models	Accuracy
1	Naive Bayes Model	Word2vec	76.80%
2	Naive Bayes Model	GloVe	78.41%
3	Naive Bayes Model	TF-IDF	84.23 %
4	Decision Tree	Word2vec	64.95%
5	Decision Tree	GloVe	72.25%
6	Decision Tree	TF-IDF	76.88 %
7	Logistic Regression	Word2vec	83.63%
8	Logistic Regression	GloVe	80.61%
9	Logistic Regression	TF-IDF	87.06 %
10	Bi-LSTM	Word2vec	84%
11	Bi-LSTM	GloVe	83%
12	Bi-LSTM	TF-IDF	84%

- We can clearly see that Bi-LSTM with Word2vec performs better than all the models with Word2vec and GloVe embeddings as input.
- We took the accuracy values of TF-IDF embeddings in ML techniques like Naive bayes, decision tree, and logistic regression from assignment 1

References

1. Word2Vec Model, https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html
2. GloVe: Global Vectors for Word Representation, <https://nlp.stanford.edu/projects/glove/>
3. BERT, https://huggingface.co/transformers/model_doc/bert.html
4. Named Entity Recognition with NLTK and SpaCy, <https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da>
5. NLP: Contextualized word embeddings from BERT, <https://towardsdatascience.com/nlp-extract-contextualized-word-embeddings-from-bert-keras-tf-67ef29f60a7b>
6. Rasa Playground, <https://rasa.com/docs/rasa/playground>
7. Create Chatbot using Rasa, <https://towardsdatascience.com/create-chatbot-using-rasa-part-1-67f68e89ddad>
8. Named Entity Recognition in Python with Stanford-NER and Spacy, <https://lvngd.com/blog/named-entity-recognition-in-python-with-stanford-ner-and-spacy/>
9. BERT for Named entity recognition, <https://www.kaggle.com/pendu777/bert-for-named-entity-recognition>
10. GloVe Embeddings + BiLSTM Sentiment Analysis, <https://www.kaggle.com/abhijeetstaulikar/glove-embeddings-bilstm-sentiment-analysis>
11. https://github.com/susanli2016/NLP-with-Python/blob/master/Sklearn_20newsgroups.ipynb
12. How to use TF IDF vectorizer with LSTM in Keras Python, <https://stackoverflow.com/questions/52182185/how-to-use-tf-idf-vectorizer-with-lstm-in-keras-python>
13. <https://www.youtube.com/watch?v=MqQ7rqRllIc>
14. NER Using LSTM in Keras, <https://www.kaggle.com/aiswaryaramachandran/ner-using-lstm-s-and-keras>
15. https://github.com/susanli2016/NLP-with-Python/blob/master/NER_NLTK_Spacy.ipynb