Report

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

Colab Links:

https://colab.research.google.com/drive/1HhBjx-vL9tdhyLi-YtdXzMFO_iwVkjmn?usp=sharing

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 (BatchNo (None, 1641, 40)          160
_________________________________________________________________
bidirectional_1 (Bidirection (None, 1641, 40)          9760
_________________________________________________________________
dropout_1 (Dropout)          (None, 1641, 40)          0
_________________________________________________________________
batch_normalization_1 (Batch (None, 1641, 40)          160
_________________________________________________________________
bidirectional_2 (Bidirection (None, 40)                9760
_________________________________________________________________
dropout_2 (Dropout)          (None, 40)                0
_________________________________________________________________
batch_normalization_2 (Batch (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 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

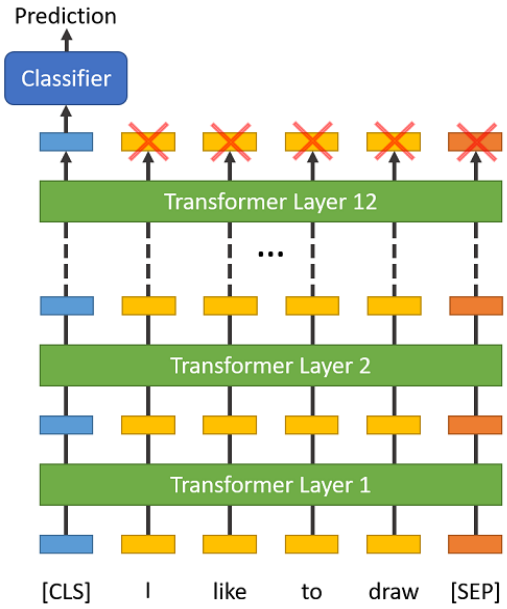
TF-IDF embeddings:

- 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.

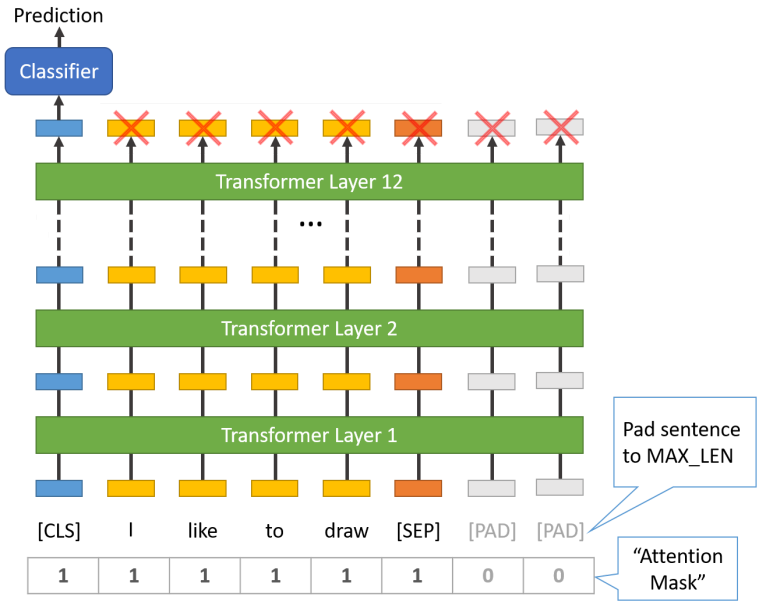
BERT:

- 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

2. 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



3. 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

    dim(Embedding Matrix) = (vocab size X 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:

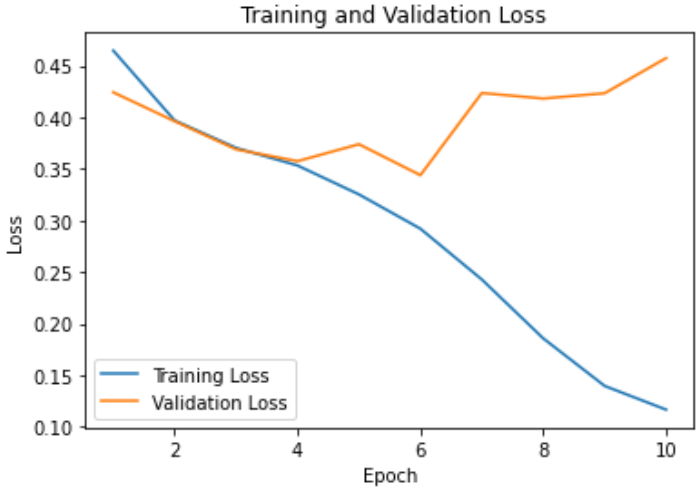
We used Adam optimiser and the loss is classification loss.

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.

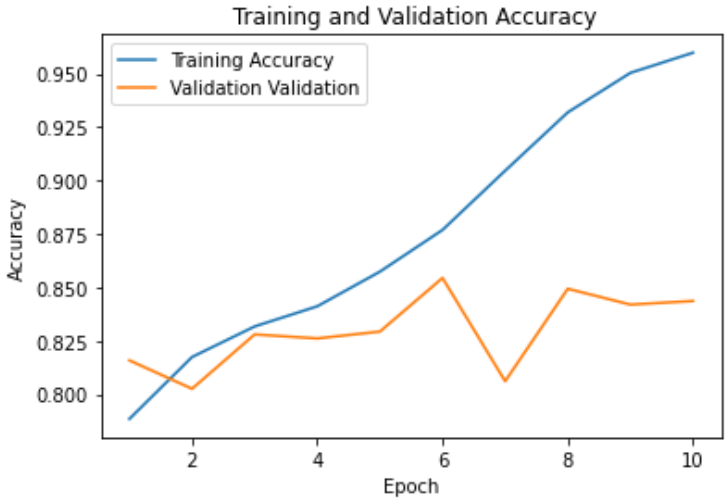
Results and Analysis:

1. Bi-LSTM with word2vec embedding:

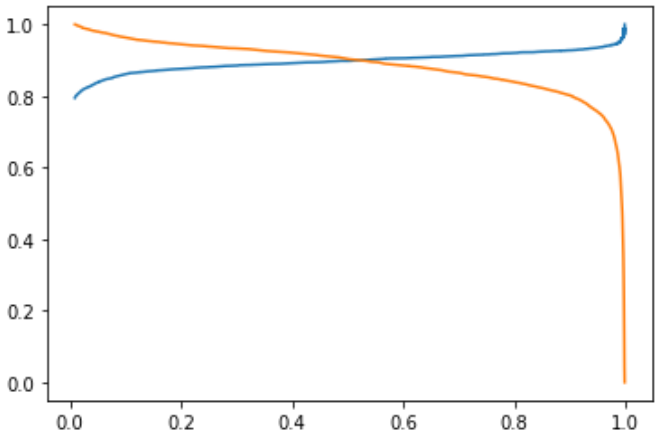


- The above graph shows the training and the validation loss curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding loss for both training and validation

- 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



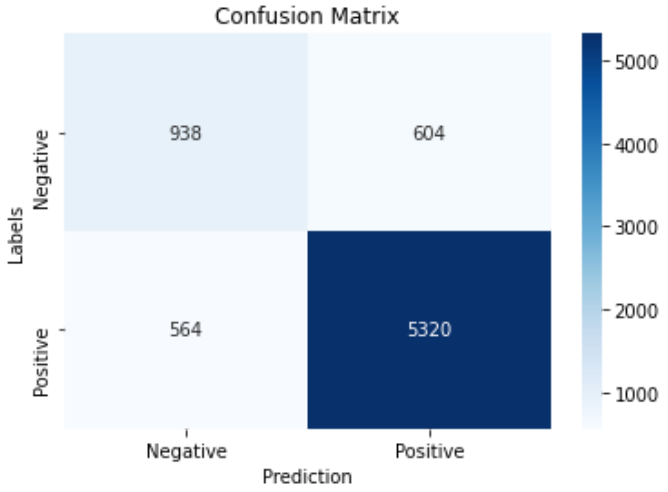
- The above graph shows the training and the validation accuracy curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding accuracy for both training and validation
- We can see that we reached 87.5% of training accuracy and 85.16% of validation accuracy after 6 epochs



- 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

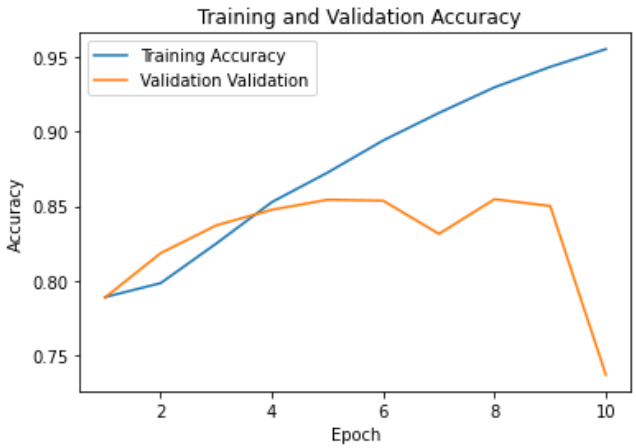
	precision	recall	f1-score	support
0	0.62	0.62	0.62	1542
1	0.90	0.90	0.90	5884
accuracy			0.84	7426
macro avg	0.76	0.76	0.76	7426
weighted avg	0.84	0.84	0.84	7426

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

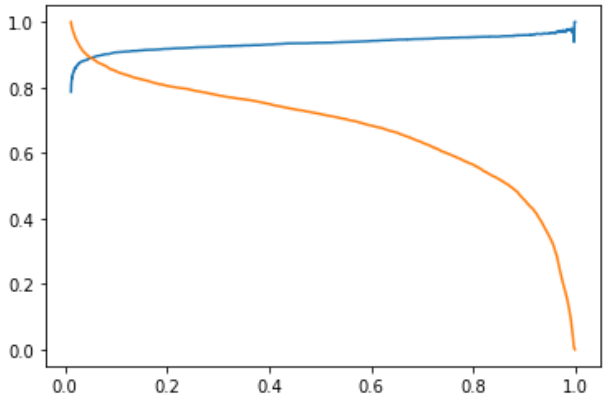


2. Bi-LSTM with glove embeddings:

- 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



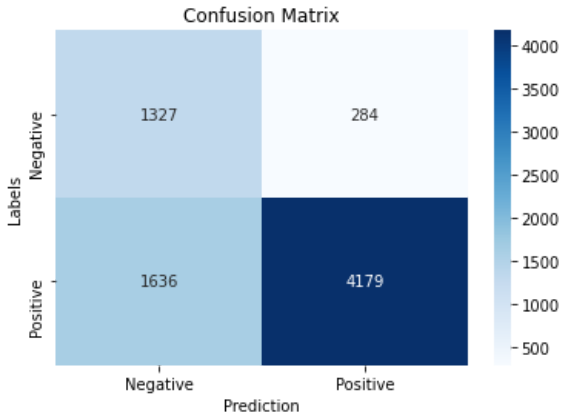
- The above graph shows the training and the validation accuracy curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding accuracy for both training and validation
- We can see that we reached 89.65% of training accuracy and 85.37% of validation accuracy after 6 epochs



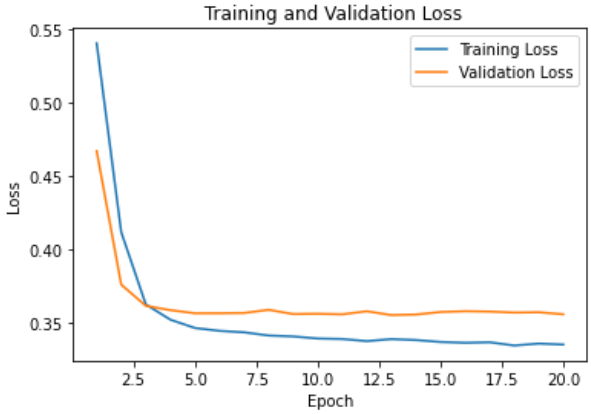
- The above figure is the Recall-precision crossover
- The blue line represents precision
- The orange line represents recall
- At the crossover point is at 0.05 and recall is 0.84 at that point

	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

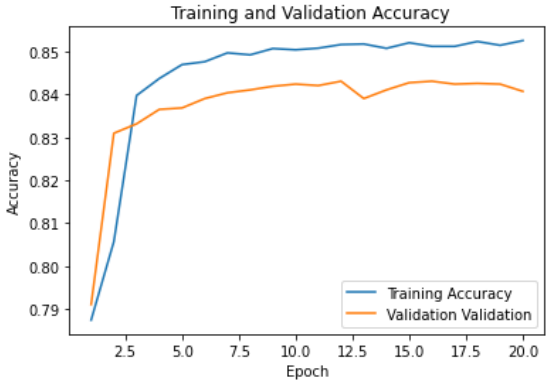
- The above figure shows the classification report for the model evaluated
- We can infer that the accuracy is 83%



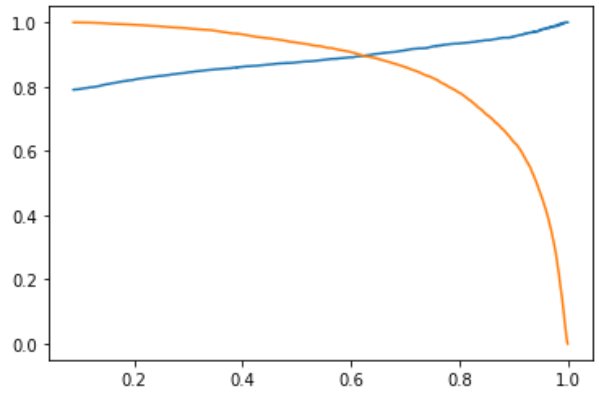
3. Bi-LSTM with TF-IDF embeddings:
    - We have completed TF-IDF before it is scraped
    - In particular, instead of the word TF-IDF produced document embedding, we fed document embedding to the LSTM Model
    - Since, TF-IDF follows BoW model it does not make any sense to feed it to LSTM since sequence 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



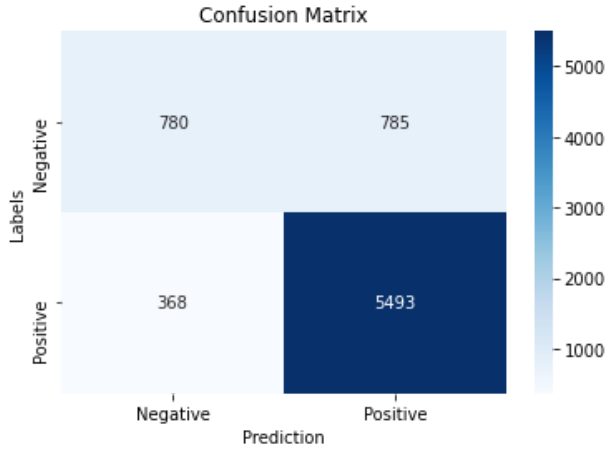
- The above graph shows the training and the validation accuracy curves
- X-axis represents the number of epochs
- Y-axis represents the corresponding accuracy for both training and validation
- We can see that we attained 85.19% of training accuracy and 84.07% of validation accuracy



- 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

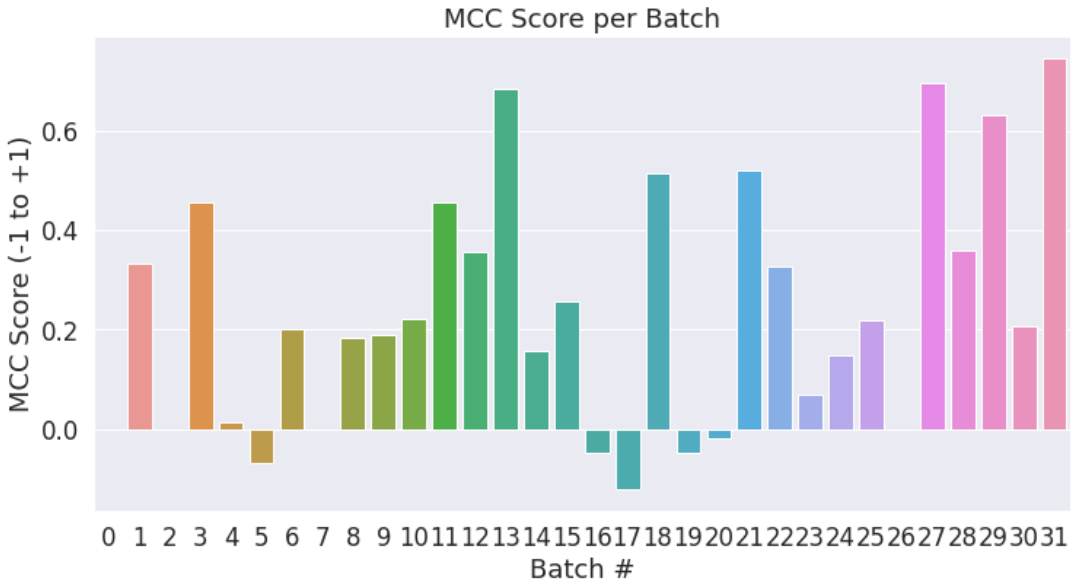
	precision	recall	f1-score	support
0	0.61	0.61	0.61	1565
1	0.90	0.90	0.90	5861
accuracy			0.84	7426
macro avg	0.75	0.75	0.75	7426
weighted avg	0.84	0.84	0.84	7426

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



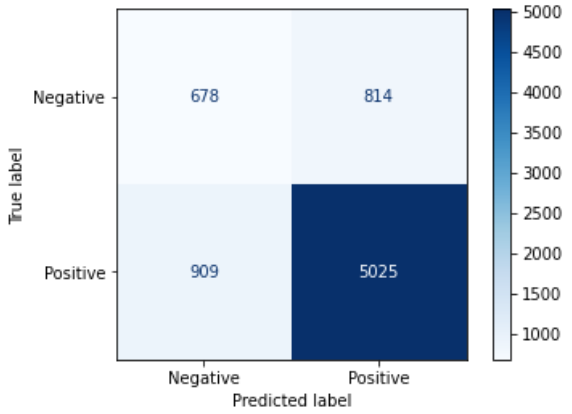
4. BERT based Sentiment Analysis

- 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

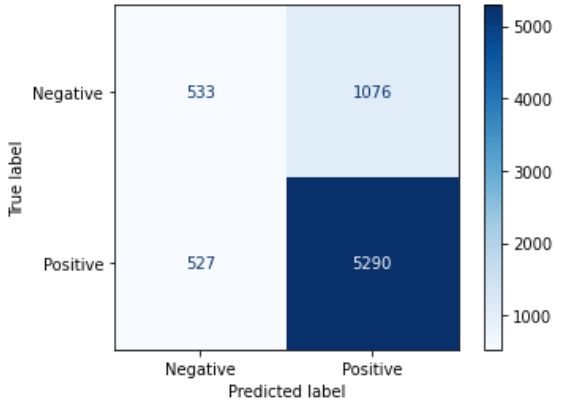


- The above figure shows the Matthews correlation coefficients score per batch
- X-axis represents the number of batches
- Y-axis represents the corresponding MCC score for each batch

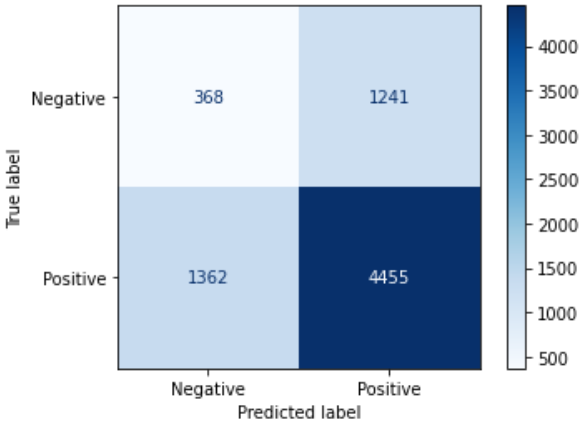
5. Naive bayes with Word2vec:
    - F1 score attained is 0.85
    - Accuracy attained is 76.8%



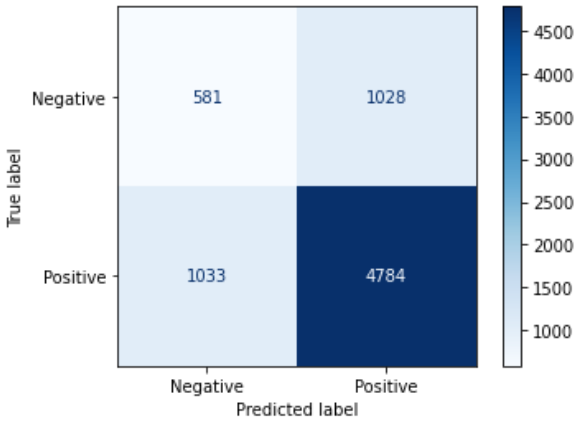
6. Naive bayes with GloVe:
    - F1 score attained is 0.87
    - Accuracy attained is 78.41%



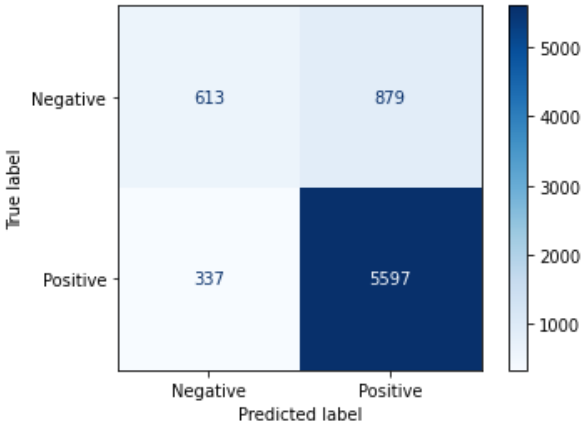
7. Decision Tree with Word2vec:
    - F1 score attained is 0.77
    - Accuracy attained is 64.95%



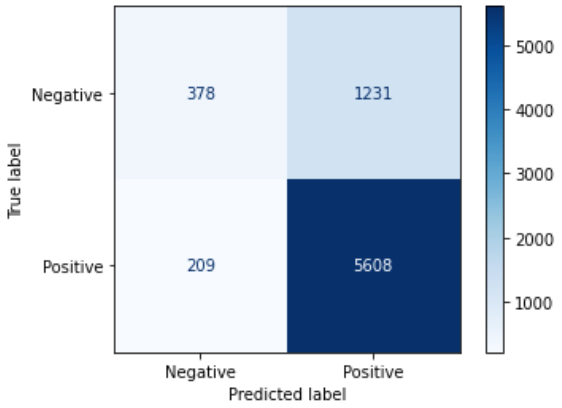
8. Decision Tree with GloVe:
    - F1 score attained is 0.82
    - Accuracy attained is 72.25%



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. NLP: Contextualized word embeddings from BERT, https://towardsdatascience.com/nlp-extract-contextualized-word-embeddings-from-bert-keras-tf-67ef29f60a7b
5. Named Entity Recognition in Python with Stanford-NER and Spacy, https://lvngd.com/blog/named-entity-recognition-in-python-with-stanford-ner-and-spacy/
6. BERT for Named entity recognition, https://www.kaggle.com/pendu777/bert-for-named-entity-recognition
7. GloVe Embeddings + BiLSTM Sentiment Analysis, https://www.kaggle.com/abhijeetstalaulikar/glove-embeddings-bilstm-sentiment-analysis
8. https://github.com/susanli2016/NLP-with-Python/blob/master/Sklearn_20newsgroups.ipynb
9. 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
10. https://www.youtube.com/watch?v=MqQ7rqRllIc
11. NER Using LSTM in Keras, https://www.kaggle.com/aiswaryaramachandran/ner-using-lstm-s-and-keras
12. https://github.com/susanli2016/NLP-with-Python/blob/master/NER_NLTK_Spacy.ipynb