Assignment-2 Report(NLP)
Gaurav Kumar - u2151556

Data Cleaning

a. Removed url links
b. Removed marked user, or words which has “@” character, or starts with it.
c. Removed “#” out of all the hashtags as some of the hashtags are simple words and may capture important information. And some of the common hashtags are surprisingly captured in the Glove as well.
d. Removed single charcater words and the words with digits in it, or which are just digits.
e. Removed stop words as they just create noise in the data and don’t add much information.
f. Finally Tokenization with Lemmatization.

Classifiers: -

Explored Traditional approach, in which machine learning classifiers are trained with features such as Countvectorizer, TFIDF and Glove. And in Deep Learning approach, explored sequential neural networks based on LSTM. Results and findings are provided below with the corresponding approaches.

1. Machine learning approach

1. Naive Bayes(NB) - Chosen Naive Bayes, as it is a probabilistic model, which makes it efficient for high dimensional data.
    a. Multinomial NB - It is based on multimodal distribution, so typically used for multilabel classification problem.
    b. Complement NB - Complemen NB works well in case of Imbalanced datasets, where one class is in dominance that can lead to overfitting of the model for the dominant class. It works on the principle of complement probabilities to handle imbalanced data. Because of this, it outperformed other NB models.
    Here,our training data is imbalanced, with more records of *Neutral* labels and less records for *Negative* labels. This model has shown better results than Gaussian and Multinomial NB
    c. Gaussian NB - Above mentioned NB models are limited to continuous data, so to train the model with glove embeddings gaussian NB is required.

All above NB models are trained with each of the feature vectors namely CountVectorizer and TFIDF. And also tried different combinations of smoothing parameters , and n-grams ranges. All models are performing best with *Alpha = 1* (smoothing parameter) and features based on unigram & bigram combination of words in the tweet. Refer to the table below.

2. SVM
    a. SGDclassifier : Trained this model with all types of feature vectors, taking default parameters in the model.

Comparison of macroaveraged f1-score of trained classifiers-feature combinations on all test datasets and development dataset. Complement NB with TFIDF is outperforming all other ML models-feature combinations.

			Macroaverage f1 score(pos+neg)			
Classifier		Features	Dev	Test1	Test2	Test3
Naive Bayes	Multinomial* NB	CountVectorizer	0.542	0.500	0.531	0.492
		TFIDF	0.454	0.360	0.405	0.358
	Complement * NB	CountVectorizer	0.565	0.542	0.547	0.532
		TFIDF	0.572	0.545	0.560	0.536
	Gaussian NB	CountVectorizer	0.520	0.483	0.506	0.464
		TFIDF	0.536	0.494	0.515	0.483
		Glove	0.512	0.514	0.516	0.506
SVM	SGD Classifier	CountVectorizer	0.436	0.438	0.441	0.421
		TFIDF	0.298	0.316	0.330	0.290
		Glove	0.476	0.438	0.466	0.476

* Note : Multinomial NB and Complement NB did not accept glove embeddings as input because of negative values in embedding vector.

2. Deep learning approach:

Initial Setup & Issues:

1. Started with a sparse *Index Matrix*(**IM**) of training data as the input, which has dimensions, ***“Length of Training datasets(post preprocessing)”* X *“Maximum length of Tweet’s tokens in the training dataset”.*** So, the row is the tweet index and the column is an array matrix of word indexes in the created embedding matrix. Dimension “~45000 x 197”. Smaller tweets (with lengths less than 197) were padded with zero.
2. **Embedding matrix** is created considering the top frequent 5000 words along with two other words “<pad>” and “<unknown>” indexed as **0** and **1** in the matrix. So the dimension is “5002 x 100”. Index 0(<pad>) has been assigned a 100 sized array of zeroes, whereas Index 1(<unknown>) is a array of random numbers(0 to 1) of size 100.
3. Created batches of the **IM** from previous step, which led to a sparse input matrix and resulted in low F-score after training the model with a single embedding layer, an LSTM layer, and a linear classification layer on top of them.

How it is resolved:

4. Implemented three approaches to make it efficient in order to reduce data sparsity.
    a. Sorted entire data by the length of tweet’s tokens.

b. *Pad sequenced* a batch created from the sorted data that is going for training,rather than creating a batch after padding the entire training data matrix(**IM**). This step is performed before forwarding the batch input to embedding layer.
c. To make the learning of lstm layer more efficient, output of embedding layer is pack padded before forwarding to lstm.

Further explorations:

After performing above steps, the macroavraged F-score has improved significantly and became close to the conventional(ML) approaches(macroF1**~0.56**). That's a relief !!

Selection of Vocab

Then I focused on the selection of 5000 words for vocab.

1. Instead of selecting top 5000 frequent words, I thought of getting the top frequent words within each sentiment category and merging them to get 5000 words.
2. Realized that top **20** most frequent words are common across all the sentiments. So, I excluded them first as they won't contribute towards classification.
3. Then, I selected the most frequent words in each sentiment category based on the inverse proportion of records in sentiment classes. Wrote an algorithm (refer to the lstm_preprocessor function in the code) to do this.
   Example: In the training data if the sentiments are in proportion, say **0.40**(*Neutral*), **0.35**(*Positive*), and **0.25**(*Negative*). Then it would choose vocabs in a way that the total number of most frequent words chosen in *Negative* will be **0.4** times the total number of frequent words in *Neutral*, and so on.
4. Implementing this resulted in a significant jump in the evaluation parameter, from *~0.56* to *~0.6* approximately. Refer to the comparison table at end of the report for details.

Error Analysis:

1. The confusion matrix on the development data set showed that the false negatives **FN** of the *Negative* class were more than its true positives **TP**, unlike other labels. Refer to the confusion matrix below.

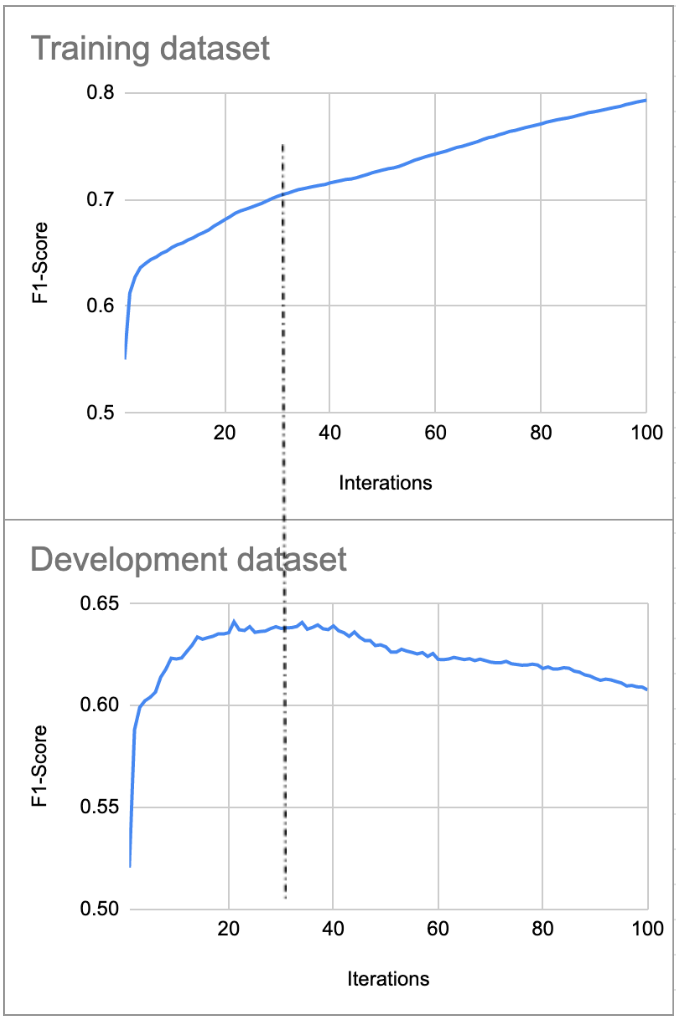
Vocab selection by labels		Predicted		
		negative	neutral	positive
Actual	negative	158	194	26
	neutral	72	679	167
	positive	21	226	456

2. This shows that majority of *Negative* labels are predicted as *Neutral*. So, I came up with two ideas after analysing this, **(i)** Check list of stop words, if negative words are getting excluded with other stop words. **(ii)** The training data seems to be quite biased for *Neutral* label(46%) as it is dominant label in training data, so under-sampling might improve the classification power of network.
3. **Stop Words analysis** - Added back the negative words which were being excluded due to stop words removal earlier.
4. **Under-sampling *Neutral* label** - Under-sampled Neutral label data on the basis of randomly generated indexes.
5. And it resulted in better performance. Below is the confusion matrix post implementing above ideas.

Vocab selection with other feature engineering		Predicted		
		negative	neutral	positive
Actual	negative	218	127	33
	neutral	114	565	239
	positive	38	157	508

Experimentation with the Achitecture:

1. Trained with different combinations of hidden size of the *LSTM layer* and *Batch size*, **[64,128,512] X [64,128]** respectively. And the best combination is **64x64,** lstm hidden layer size and batch size respectively.
2. Trained with two *LSTM layers* but the performance was poor, it was not comparable with the single layer, so I haven't shown that in the report.
3. Kept the *learning rate* constant and tracked the performance of both train and development dataset to get the best *epoch* for above combinations. Refer to the chart below which shows the trend of macroaveraged f1-scores for train and development datasets with number of *epochs*.



4. Trained network for a range of *epochs* to check if model shows any further improvement, but found that **100** *epochs* are conclusive enough. It is visible that after 30-40 epochs, the model starts overfitting on training data.
5. Refer to the below comparison table of macoraveraged f1-scores for all 5 data sets. It contains results from all the experiments mentioned in previous steps.

						Macroaverage F1 Score(pos+neg)				
Exp	Feature Engineering	Batch Size	Learning Rate	LSTM Hidden Size	Best Epoch	Train	Dev	Test1	Test2	Test3
1	Vocab selection by label	64	0.0001	64	35	0.636	0.593	0.521	0.557	0.523
2		128	0.0001	64	37	0.647	0.625	0.554	0.586	0.550
3		64	0.0001	128	33	0.675	0.609	0.543	0.588	0.540
4		128	0.0001	128	29	0.633	0.606	0.536	0.561	0.520
5		64	0.0001	512	22	0.698	0.623	0.589	0.618	0.569
6		128	0.0001	512	28	0.694	0.615	0.597	0.614	0.565

7	downsample neutral , remove negative words from stop words along with vocab selection by label	64	0.0001	64	30	0.701	0.639	0.616	0.652	0.605
8		128	0.0001	64	34	0.704	0.633	0.603	0.613	0.578
9		64	0.0001	128	20	0.686	0.636	0.611	0.657	0.592
10		128	0.0001	128	35	0.699	0.629	0.597	0.628	0.579
11		64	0.0001	512	18	0.692	0.621	0.599	0.637	0.561
12		128	0.0001	512	20	0.691	0.629	0.612	0.642	0.592

Conclusion:

1. Deep learning has shown significant improvement in macroaveraged f1-score over machine learning approaches
2. Things that can be explored further -
    a. considering features based on emoticons, which I believe could bring better results.
    b. Ensembling of the different models etc.…