CZ4032 Data Analytics and Mining Assignment

Team 28

Marcia Ong Wen Jie	U1622511F
Jonathan Wong	U1622446E
Marathe Ajinkya Avinash	U1522716K
Wong Zhen Wei	U1621109F
Vu Duc Long	U1520526J
Nguyen Dang Duy Nghia (Kyle)	U1520536G

Abstract

Sentiment Analysis is the process of computationally identifying and categorizing opinions expressed in a piece of text in order to determine the writer’s attitude towards a particular topic. This process of extracting insights from social data has been widely adopted by all over the world for tasks like opinion mining about products and stock market prediction systems.

The aim of this report is to summarise the different approaches we used for sentiment analysis on movie review data and to evaluate their efficacy. Kaggle’s Rotten Tomatoes Movie Review dataset has been used for the experimentation. The data preprocessing, data cleaning and data mining steps for each of these approaches have been explained in detail along with the rationale for choosing the approach and the accuracy from the Kaggle submissions. The report ends with a conclusion derived from our approaches and suggesting some improvements.

Problem Description

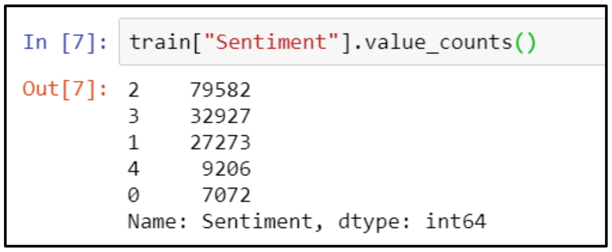
The Rotten Tomatoes movie review dataset is a corpus of movie reviews used for sentiment analysis, originally collected by Pang and Lee [1]. In their work on sentiment treebanks, Socher et al. [2] used Amazon's Mechanical Turk to create fine-grained labels for all parsed phrases in the corpus.

Kaggle is hosting this sentiment analysis competition for the machine learning community. Participants are asked to label phrases on a scale of five values: negative, somewhat negative, neutral, somewhat positive, positive. Obstacles like sarcasm, sentence negation, terseness, language ambiguity, and others makes this competition very challenging.

This competition presents a chance for our team to use the concepts of data mining learnt through the course and benchmark our sentiment-analysis model on the Rotten Tomatoes dataset.

Exploratory Data Analysis

Before we begin building model, we start off by doing Exploratory Data Analysis (EDA).



```
In [7]: train["Sentiment"].value_counts()

Out[7]: 2    79582
        3    32927
        1    27273
        4     9206
        0     7072
        Name: Sentiment, dtype: int64
```

Figure 1

Referring to Figure 1 the train file contains 156060 samples of 5 classes, with 0 being the most negative and 4 being the most positive; value 2 is considered to be neutral. Counting the number of each class, we saw that neutral comments take almost half of the dataset with 79582 samples leaving the rest summing up to 76478 samples altogether.

Attached below is the distribution chart of the classes:

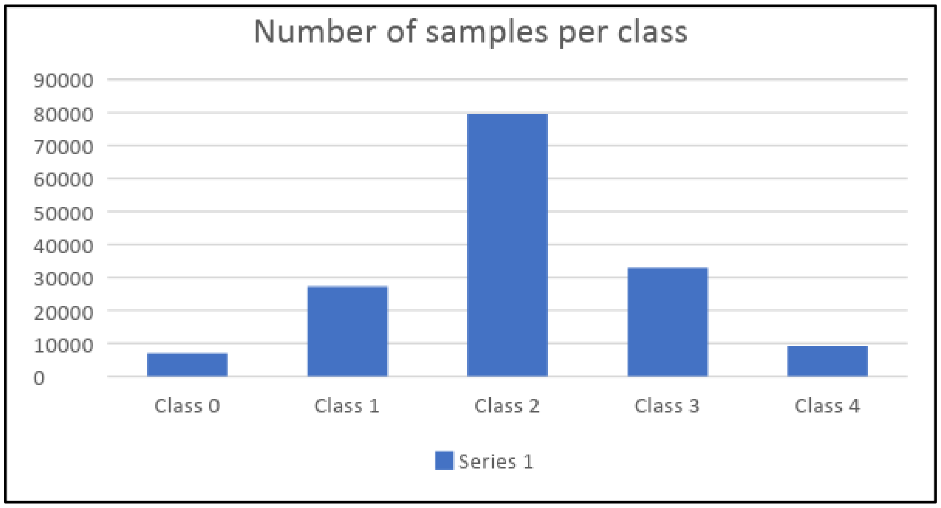


Figure 2: Distribution Chart

Data Preprocessing

The preprocess on the text includes: transform all text to lowercase, tokenizing the words and transform a phrase into a vector of the corresponding tokens of each word. In the Tokenizing phase we needed to determine a common length for the vector representing a phrase. To determine the cut, we plotted out the distribution of phrase length as follows:

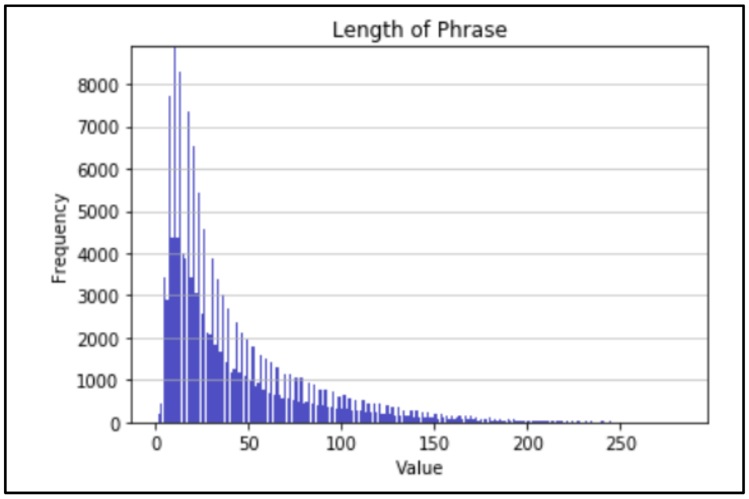


Figure 3: Distribution of Phrase length chart

As such, the maximum number of words kept is 100, since most of the phrases' length are within that
range. For phrases that are longer than that, we cut the excessive words at the end.

Data Mining Models

Our team have developed 6 models using various toolkits and algorithms. The models are:

1. TextBlob
2. LSTM
3. GloVe + LSTM
4. TF-IDF + Linear SVM
5. Word2Vec + CNN
6. Word2Vec + LSTM

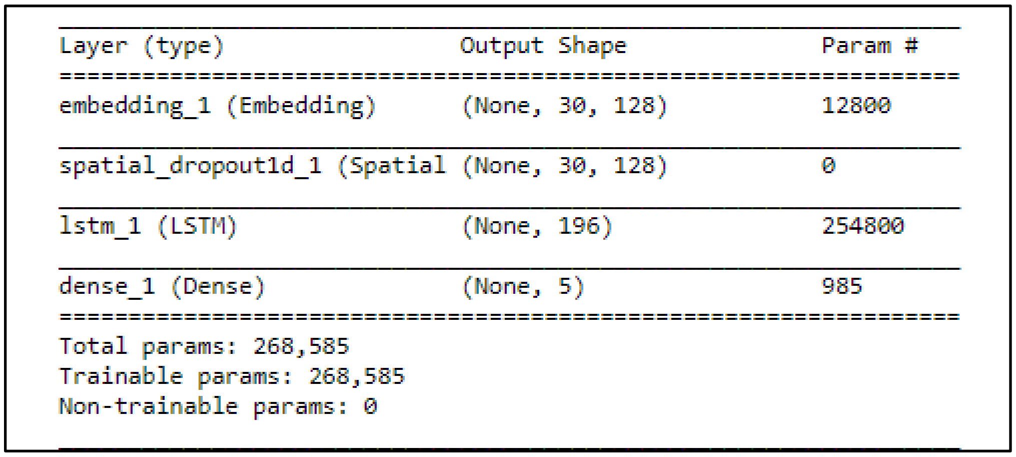
TextBlob

The first experiment was done with an out-of-the-box library for sentiment analysis, Textblob. There is no
training phase involved here and we used Textblob merely for the inference process. The Phrase was
directly fit into the Textblob object without any preprocessing of the text. The output is a metric called
popularity and is a real number of -1.0 to 1.0. We divided the range to 5 equal range and assigned each
range to a corresponding class of the classification task. The Kaggle score for the submission was 0.53232.

movies_reviews (version 2/7) 0.53232
2 months ago by helloWorld
From "movies_reviews" Script

LSTM

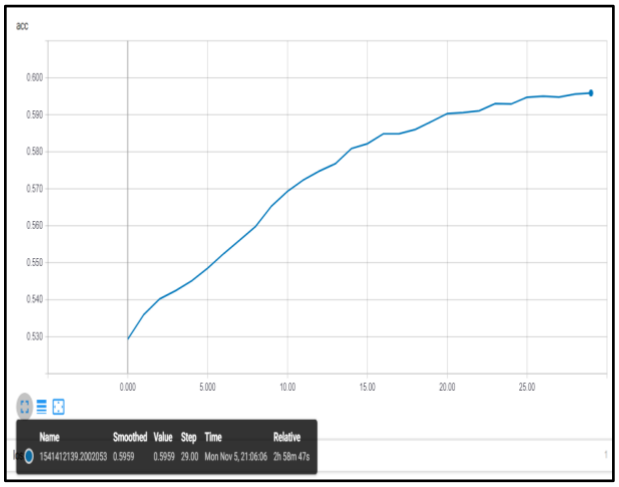
Recurrent model is known to be suitable for sequence data and text can be considered as a sequence of word. We trained an LSTM model to classify the sentiment class. We had an embedding layer before the LSTM layer with output dimension of 128. The 128-dimension encoding is fed into the 196-cell LSTM with drop out rate 0.2. The summary of the model is as follows:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 30, 128)           12800     
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 30, 128)           0         
_________________________________________________________________
lstm_1 (LSTM)                (None, 196)               254800    
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 985       
=================================================================
Total params: 268,585
Trainable params: 268,585
Non-trainable params: 0
_________________________________________________________________
```

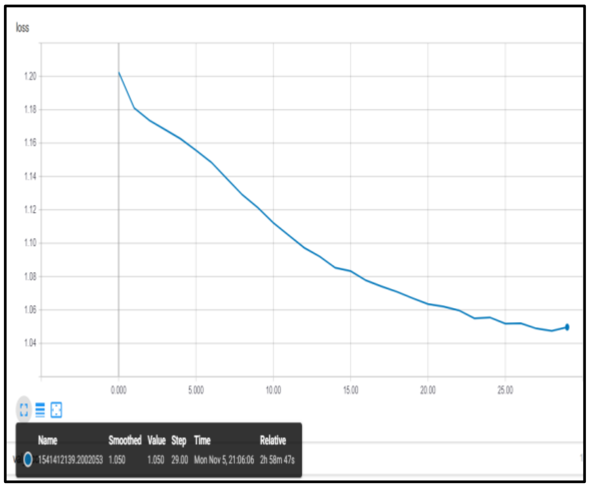
Figure 4: Summary of LSTM Model

We split the whole train dataset into train and validation set with 33% test ratio. The model was trained with Adam optimizer in an mini-batch manner with batch size of 32. After training for 30 epochs in 2.375 hours with Tensorflow-gpu with a device of gpu compute capability 5.0 (calculated by Tensorflow), the model plateau:



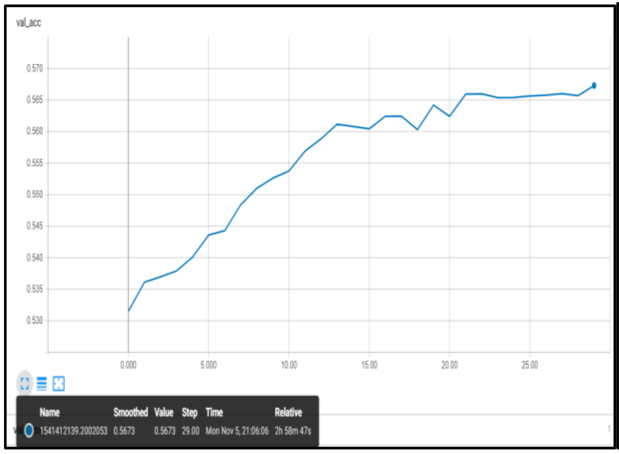
	Name	Smoothed	Value	Step	Time	Relative
●	1541412139.2002053	0.5959	0.5959	29.00	Mon Nov 5, 21:06:06	2h 58m 47s

Figure 5: Training Accuracy



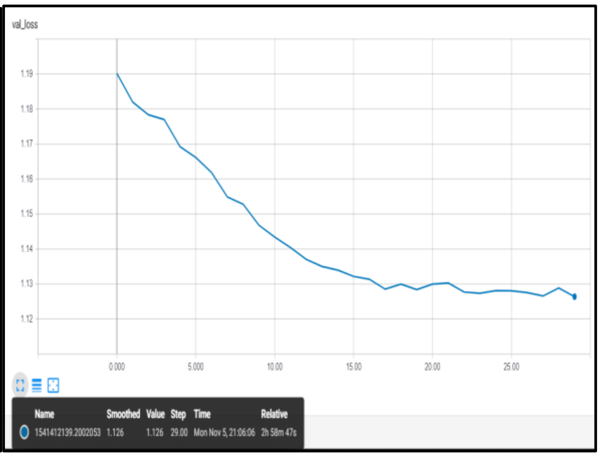
	Name	Smoothed	Value	Step	Time	Relative
●	1541412139.2002053	1.050	1.050	29.00	Mon Nov 5, 21:06:06	2h 58m 47s

Figure 6: Training Loss



	Name	Smoothed	Value	Step	Time	Relative
●	1541412139.2002053	0.5673	0.5673	29.00	Mon Nov 5, 21:06:06	2h 58m 47s

Figure 7: Validation Accuracy



	Name	Smoothed	Value	Step	Time	Relative
●	1541412139.2002053	1.126	1.126	29.00	Mon Nov 5, 21:06:06	2h 58m 47s

Figure 8: Validation Loss

We picked the model at epoch 7 and epoch 30.

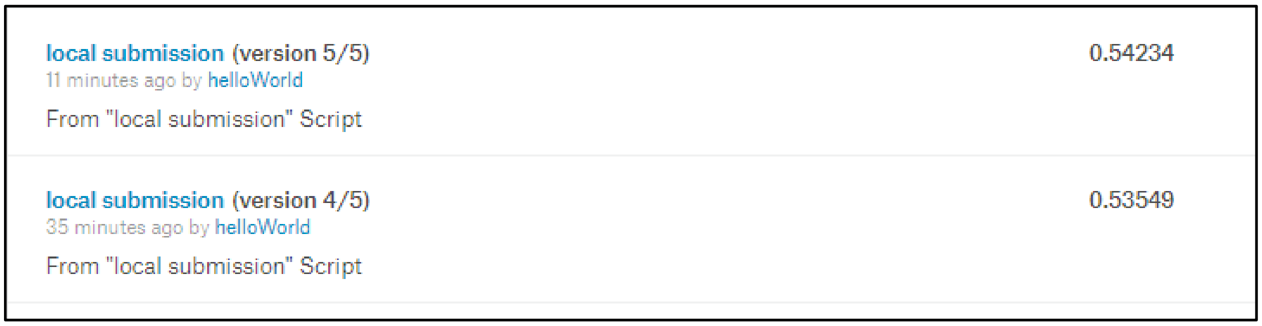
205	511	1352	258	15
206	1709	650	622	21
78	1035	27732	1112	41
32	499	8133	2463	163
12	111	1800	942	245

Table 1: Confusion Matrix on Validation Set (epoch 7)

404	625	1100	186	26
351	2166	5902	592	50
90	1132	23046	1616	114
36	294	7137	2977	546
34	60	1401	992	623

Table 2: Confusion Matrix on Validation Set (epoch 30)

Use both models to do prediction on the test set and submit to Kaggle we got 0.54234 for epoch 7 model and 0.53549 for epoch 30 model.

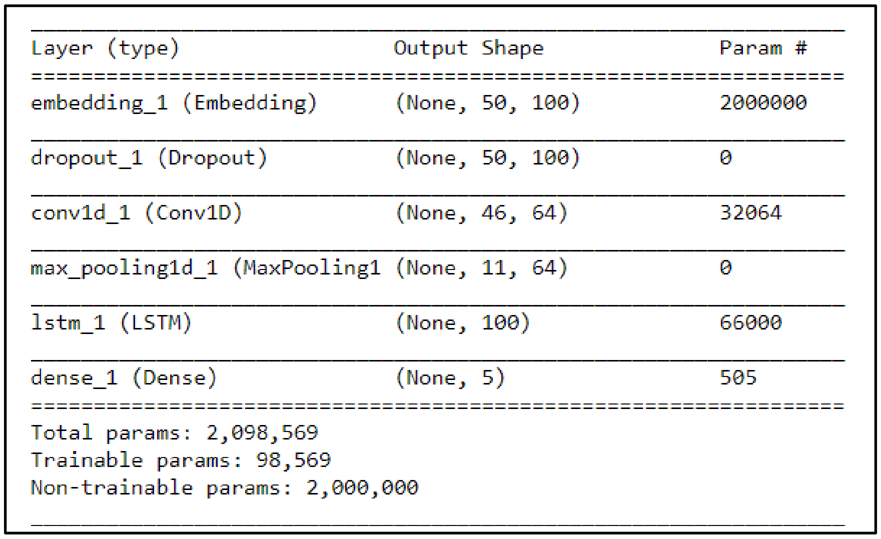


local submission (version 5/5) 0.54234
11 minutes ago by helloWorld
From "local submission" Script

local submission (version 4/5) 0.53549
35 minutes ago by helloWorld
From "local submission" Script

GloVe + LSTM

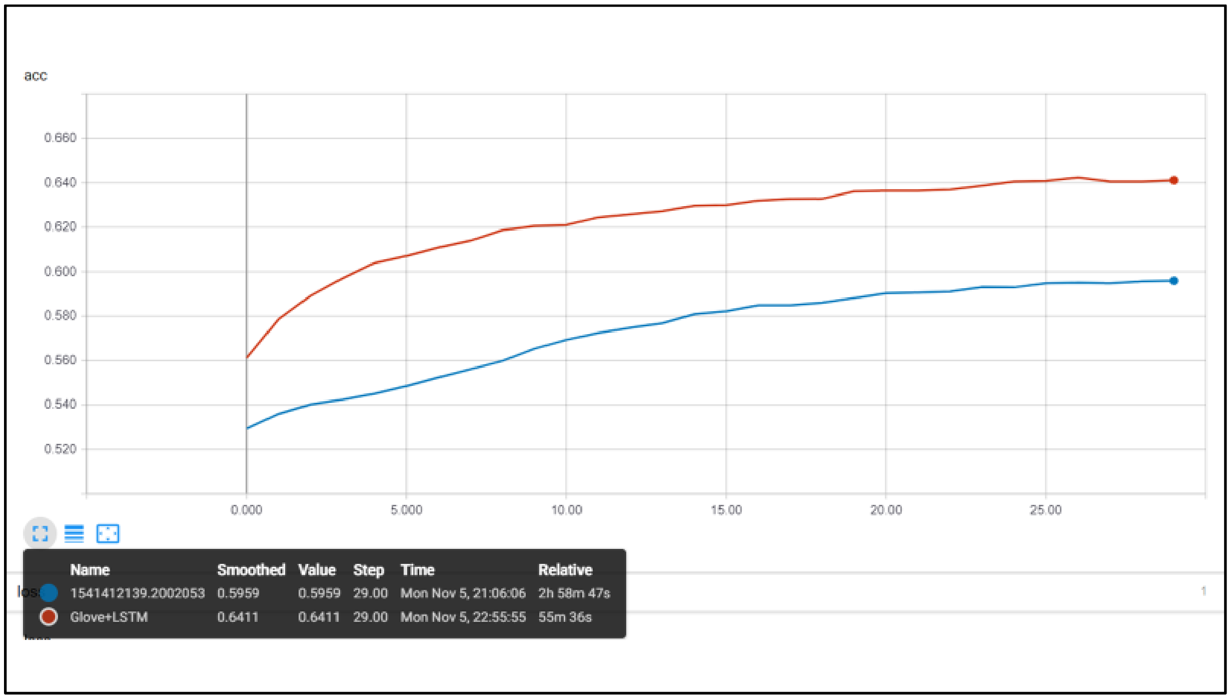
Instead of training our own embedding layer, in this model we made use of a pretrained embedding layer that is pretrained on 6 billion words on Wikipedia, called GloVe. It maps a certain word in the corpus to a vector of length 100. The network is changed to match the dimension of the new feature vector. Furthermore, we added an extra 1D convolutional layer on top of the LSTM layer to reduce the training time. The summary of the model is as follows:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 50, 100)           2000000
_________________________________________________________________
dropout_1 (Dropout)          (None, 50, 100)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 46, 64)            32064
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 11, 64)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               66000
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 505
=================================================================
Total params: 2,098,569
Trainable params: 98,569
Non-trainable params: 2,000,000
_________________________________________________________________
```

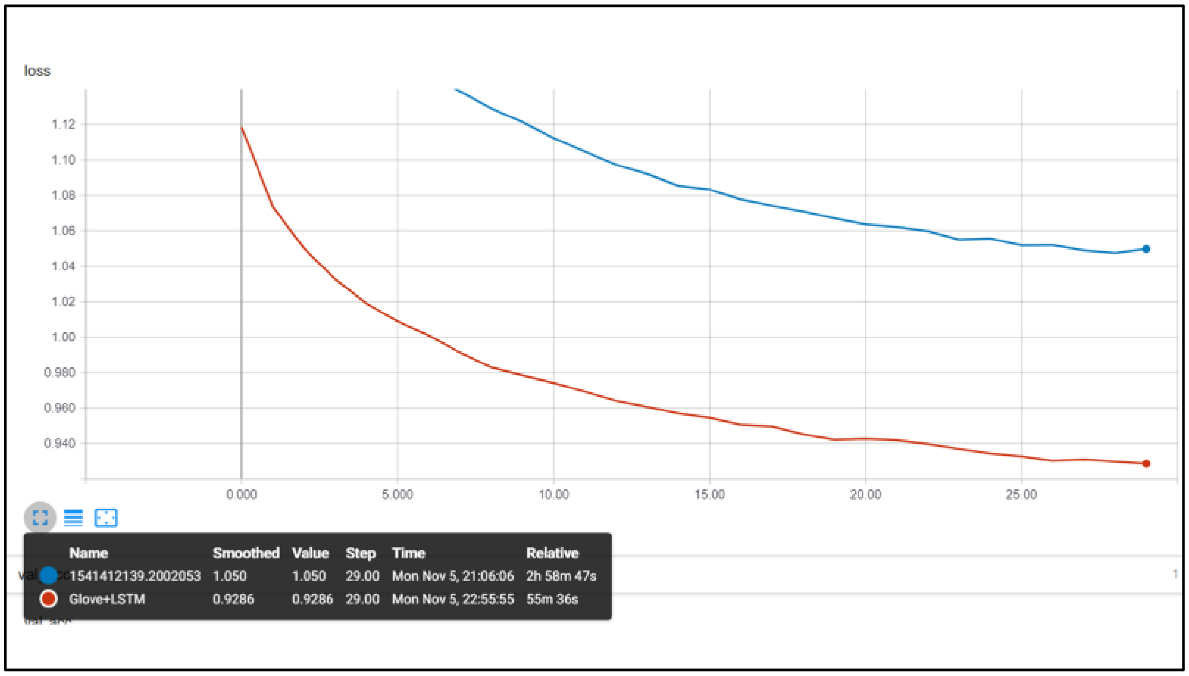
Figure 9: Summary of GloVe + LSTM model

This model showed more generalisation capability than LSTM model. We compared the LSTM models in training test validation phase, GloVe + LSTM model 3 shown in red, LSTM model in blue:



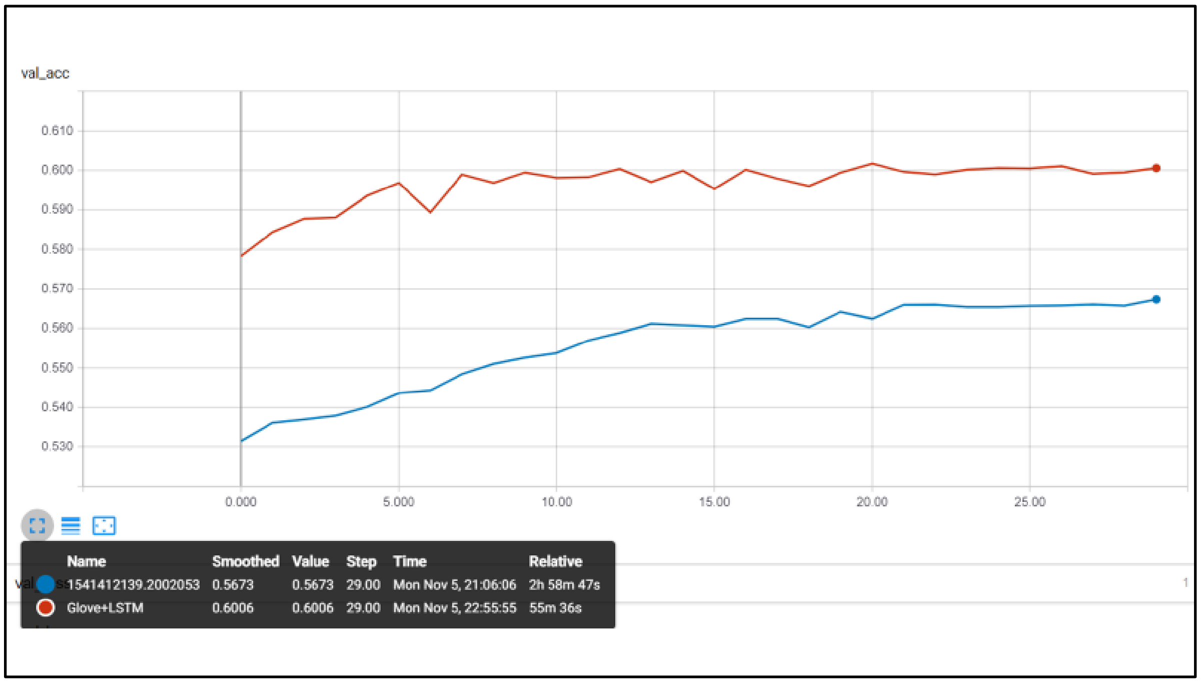
	Name	Smoothed	Value	Step	Time	Relative
🔵	1541412139.2002053	0.5959	0.5959	29.00	Mon Nov 5, 21:06:06	2h 58m 47s
🔴	Glove+LSTM	0.6411	0.6411	29.00	Mon Nov 5, 22:55:55	55m 36s

Figure 10: Training Accuracy model of LSTM and GloVe + LSTM



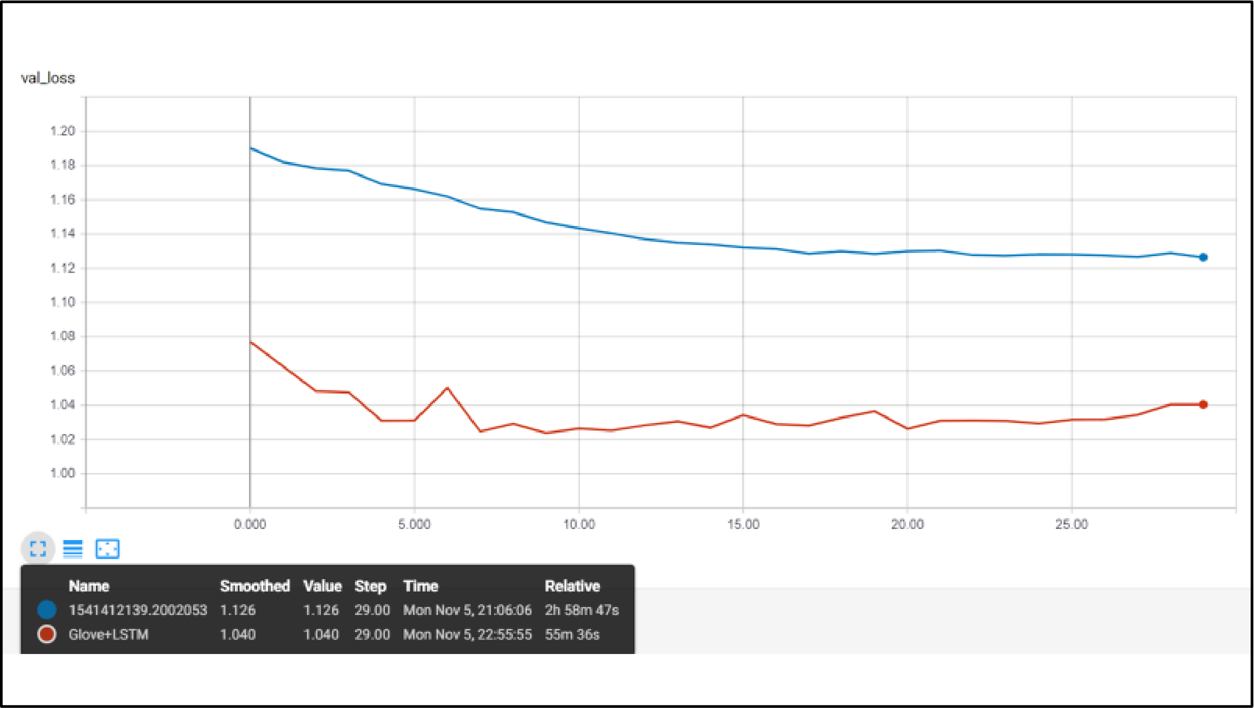
Name	Smoothed	Value	Step	Time	Relative
● 1541412139.2002053	1.050	1.050	29.00	Mon Nov 5, 21:06:06	2h 58m 47s
● Glove+LSTM	0.9286	0.9286	29.00	Mon Nov 5, 22:55:55	55m 36s

Figure 11: Training Loss model of LSTM and GloVe + LSTM



Name	Smoothed	Value	Step	Time	Relative
● 1541412139.2002053	0.5673	0.5673	29.00	Mon Nov 5, 21:06:06	2h 58m 47s
● Glove+LSTM	0.6006	0.6006	29.00	Mon Nov 5, 22:55:55	55m 36s

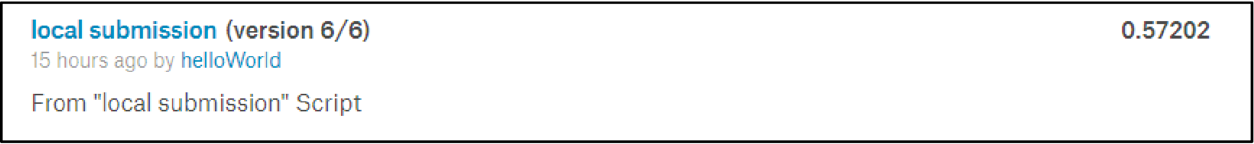
Figure 11: Validation Accuracy model of LSTM and GloVe + LSTM



	Name	Smoothed	Value	Step	Time	Relative
●	1541412139.2002053	1.126	1.126	29.00	Mon Nov 5, 21:06:06	2h 58m 47s
●	Glove+LSTM	1.040	1.040	29.00	Mon Nov 5, 22:55:55	55m 36s

Figure 12: Validation loss model of LSTM and GloVe + LSTM

Took the model at epoch 30 to do prediction on test dataset and submit to Kaggle, we got a better score than the old model, achieving 0.57202.



local submission (version 6/6) 0.57202

15 hours ago by helloWorld

From "local submission" Script

TF-IDF + Linear SVM

Feature extraction

Each token is assigned with a unique integer identifier and the occurrence of each token in each sentence is recorded. The token frequency in each sentence will be represented as a document-term matrix. Term frequency and inverse document frequency(tf-idf) term weighting scheme will be applied to the document-term matrix. TF-IDF is a numerical statistic intended to reflect the importance of a word to a document in a collection. It diminished weight of terms that occurs very frequently such as ‘the’, ‘a’ and increase weight of term that occurs rarely. The inverse document frequency is computed by dividing the total number of documents by number of documents that has a given term.

Text Classification

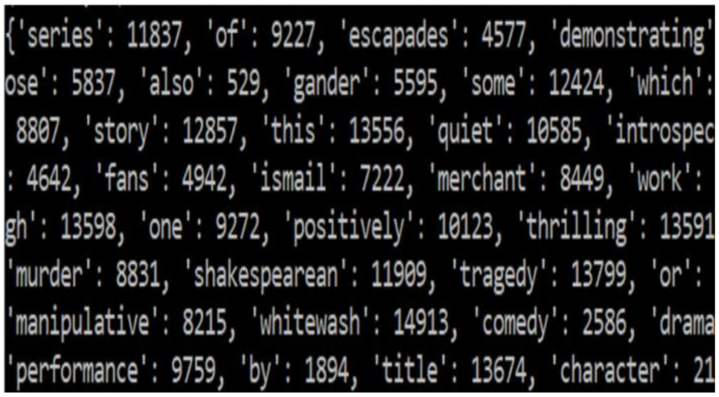
Linear Support Vector Classifier(SVC) is used for the classification experiment. It is a binary linear classifier that attempt to find the hyperplane that separates two different classes of points. Hyperplane is a line that maximises the margin between the two classes. Data are represented as points in space and the data of the different categories are divided by a clear gap that is as wide as possible. The points that lie on this margin are the support vectors and are the nearest points to the hyperplane. New instances of data are then mapped onto the same space and predicted category is based on which side of the gap the data point belongs.

In our case where there are more than 2 classes, SVC uses the one-vs-rest scheme(OVR) strategy where each class creates a binary classifier that predicts that class against all the other classes. To predict a new instance of data, all the binary classifiers will be applied onto the data and the classifier with the highest output function will assign its class to the new data.

Implementation

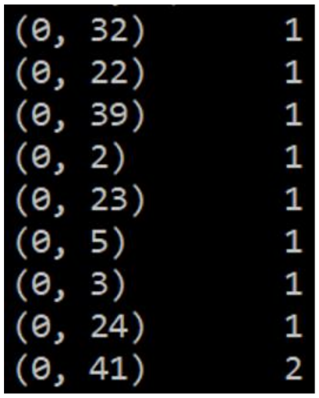
The implementation of pre-processing, feature extraction and text classification are done by using the modules found in **sklearn** libraries namely ***CountVectorizer***, ***TfidTransformer***, ***LinearSVC***, ***Pipeline*** and ***GridSearchCV***.

The ***CountVectorizer*** module is responsible for the pre-processing of data and representing the term frequency as a compressed sparse row matrix. Sparse matrix is used because less memory is required as only non-zero elements are stored and speed up computing as it only traverse through the nonzero elements. Figure 13 shows the list of vocabulary and its integer identifier created from tokenisation based on the training dataset. Figure 14 shows the term frequency in each sentence represented by compressed sparse row matrix. As shown in figure 14, the first value in the parentheses is the index of the sentence and the second value is the token identifier. The last value is the frequency of the token in the sentence.



```
{'series': 11837, 'of': 9227, 'escapades': 4577, 'demonstrating'
ose': 5837, 'also': 529, 'gander': 5595, 'some': 12424, 'which':
 8807, 'story': 12857, 'this': 13556, 'quiet': 10585, 'introspec
: 4642, 'fans': 4942, 'ismail': 7222, 'merchant': 8449, 'work':
gh': 13598, 'one': 9272, 'positively': 10123, 'thrilling': 13591
'murder': 8831, 'shakespearean': 11909, 'tragedy': 13799, 'or':
'manipulative': 8215, 'whitewash': 14913, 'comedy': 2586, 'drama
'performance': 9759, 'by': 1894, 'title': 13674, 'character': 21
```

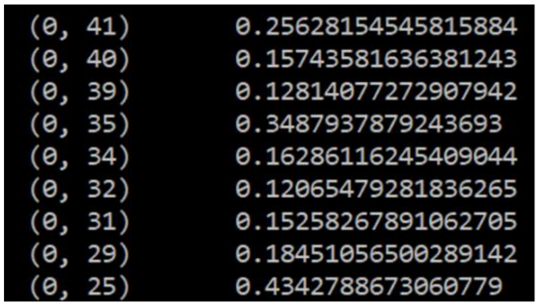
Figure 13: Tokenisation with an integer identifier



```
(0, 32)       1
(0, 22)       1
(0, 39)       1
(0, 2)        1
(0, 23)       1
(0, 5)        1
(0, 3)        1
(0, 24)       1
(0, 41)       2
```

Figure 14: TF in Doc-term matrix

The ***TfidTransformer*** module implement the tf-idf term weighting scheme to calculate the tf-idf weights for the document-term matrix. Figure 15 shows the tf-idf weights for each token in each sentence.



```
(0, 41)       0.25628154545815884
(0, 40)       0.15743581636381243
(0, 39)       0.12814077272907942
(0, 35)       0.3487937879243693
(0, 34)       0.16286116245409044
(0, 32)       0.12065479281836265
(0, 31)       0.15258267891062705
(0, 29)       0.18451056500289142
(0, 25)       0.4342788673060779
```

Figure 15: Tf-idf weights for each token in each sentence

To train a classifier to predict the sentiment value of a sentence which ranges from 0 to 4, the tf-idf weights for the document-term matrix is fitted into the ***LinearSVC*** along with the sentiment value of each sentence.

For simplicity, a pipeline is created which assemble the steps for feature extraction, tf-idf weighting and linear SVC by using the Pipeline module. It allows all the steps to be cross-validated together while setting different parameters for hyperparameter tuning. To find the optimum hyperparameter, ***GridSearchCV*** module is used. The module performs exhaustive search over specified parameter values and find the estimator that gives the best performance. 3-fold cross validation is performed to reduce the occurrence of overfitting. In our implementation, the parameters to be optimised are the maximum term frequency for ***CountVectorizer*** and the regularization parameter C for ***LinearSVC***. Maximum term frequency(max_df) is used for removing terms that appears too frequently, for example max_df = 0.50 means remove term that appear in more than 50% of the sentences in the training dataset. Maximum term frequency of 0.8, 0.9 and 1.0 is specified to be tuned. Regularization parameter C is used to determine the influence of misclassification for each training data. A large value of C will result in a smaller-margin hyperplane to classify all training data correctly which will lead to overfitting. Conversely, a very small value of C will result in a larger-margin separating hyperplane, even if the hyperplane misclassifies more points. C with a value of 0.01, 0.1 and 1.0 is specified to be tuned.

Result Analysis

The average accuracy obtained through cross validation using linear SVC is 0.58 with optimized parameter max_df = 0.80 and C=0.1.

Figure 4 shows the weightage of different sentiment movie review found in the training dataset. As shown in figure 4, 50% of the training dataset contains movie review with sentiment value 2. Movie review with sentiment value 1 and 3 is the second most followed by movie review with sentiment value 0 and 4. Figure 5 shows the confusion matrix using linear SVC with optimized parameter. As shown in figure 5, movie review with a sentiment value of 2 has an 88% accuracy and movie review with sentiment value of 0,1,3 and 4 has an accuracy of 15%, 23%, 33% and 24% respectively. It can be noticed that the accuracy is proportional to the number of samples in the training data having the specific sentiment value. Hence, we can conclude that more sample for a specific sentiment value can improve the accuracy of the class.

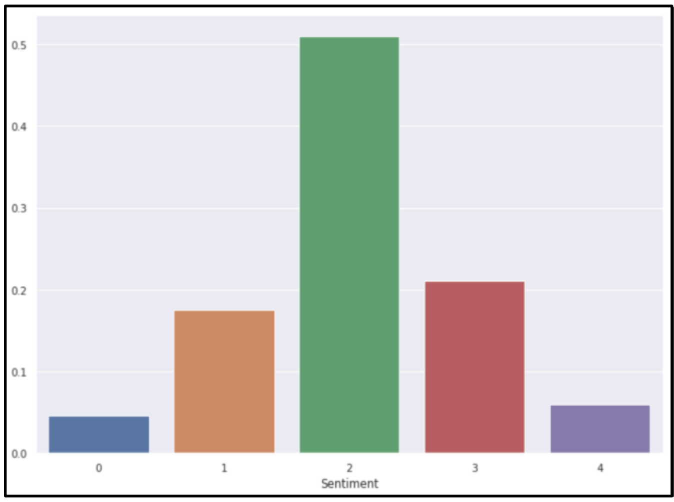


Figure 16: Distribution of Movie Review

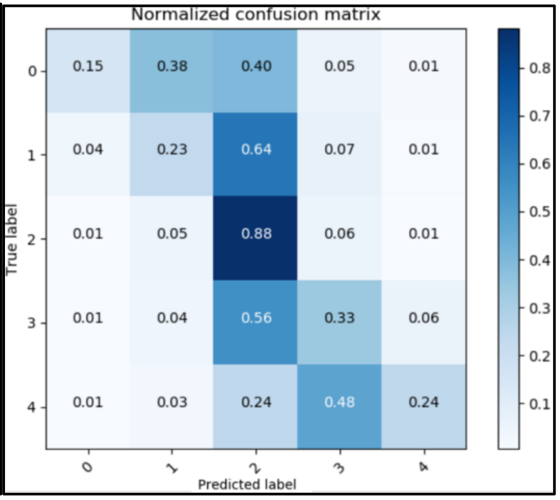
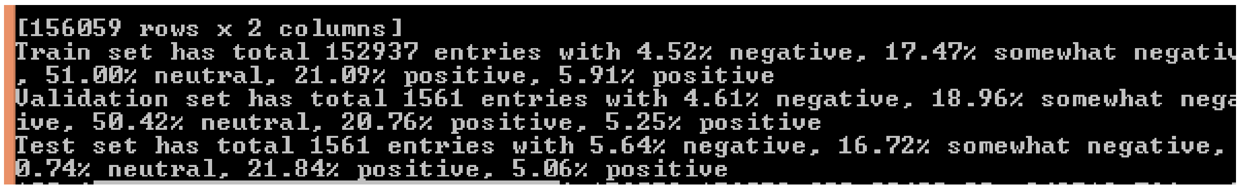


Figure 17: Confusion Matrix

Word2Vec + CNN

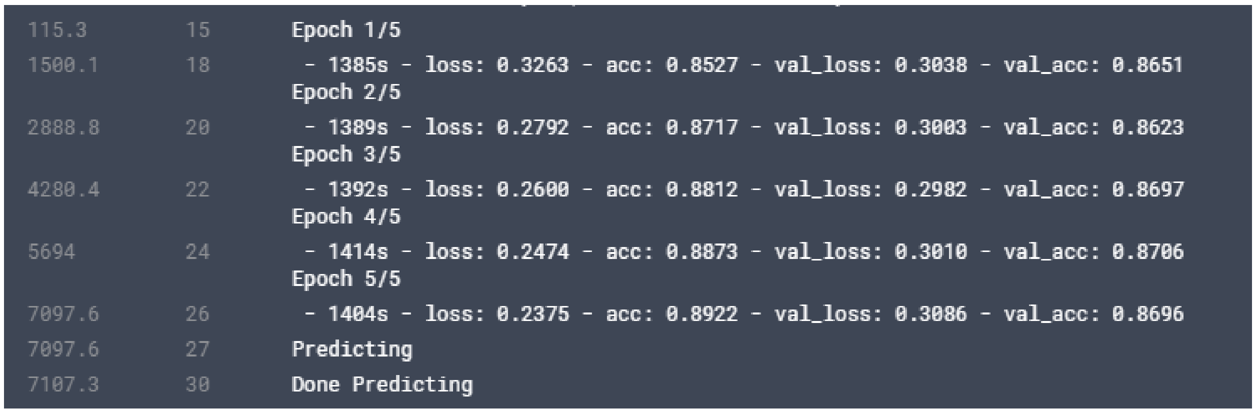
Word2Vec is a group related model that is used to produce word embeddings. This model is used to train and reconstruct linguistic contexts of words. Word2vec consists of two algorithms, Continuous bag of words and Skip-gram model. Both algorithms will learn weights which act as word vector representations. Continuous bag of words predicts a current word from a window of surrounding context words. Skip-gram models predicts surrounding context words given the current word. In other words, Word2Vec was used to calculate the summation and mean to get a vector representation of each review.

To implement the Word2Vec into the CNN model, we had to do a padding of the words based on the longest review. Based on the analysis we extracted the longest word which is 42 words and to add buffer, we padded all reviews to 47 words. Referring to figure 18 ,we split the whole train dataset into train, validation and test set. 98% of the data was kept for training while validation and test set was split into the remaining 2%.



```
[156059 rows x 2 columns]
Train set has total 152937 entries with 4.52% negative, 17.47% somewhat negati
, 51.00% neutral, 21.09% positive, 5.91% positive
Validation set has total 1561 entries with 4.61% negative, 18.96% somewhat nega
ive, 50.42% neutral, 20.76% positive, 5.25% positive
Test set has total 1561 entries with 5.64% negative, 16.72% somewhat negative,
0.74% neutral, 21.84% positive, 5.06% positive
```

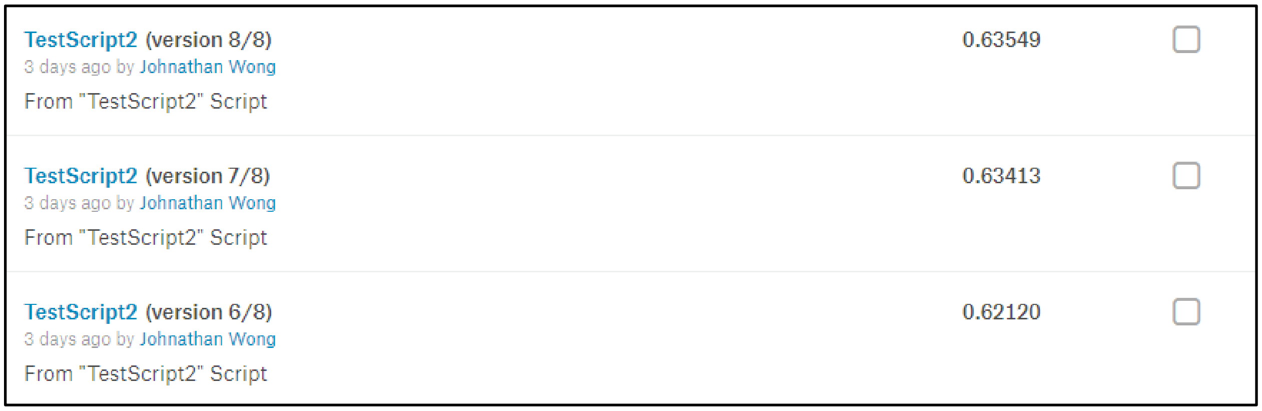
Figure 18: Dataset Split



```
115.3     15      Epoch 1/5
1500.1    18       - 1385s - loss: 0.3263 - acc: 0.8527 - val_loss: 0.3038 - val_acc: 0.8651
                  Epoch 2/5
2888.8    20       - 1389s - loss: 0.2792 - acc: 0.8717 - val_loss: 0.3003 - val_acc: 0.8623
                  Epoch 3/5
4280.4    22       - 1392s - loss: 0.2600 - acc: 0.8812 - val_loss: 0.2982 - val_acc: 0.8697
                  Epoch 4/5
5694      24       - 1414s - loss: 0.2474 - acc: 0.8873 - val_loss: 0.3010 - val_acc: 0.8706
                  Epoch 5/5
7097.6    26       - 1404s - loss: 0.2375 - acc: 0.8922 - val_loss: 0.3086 - val_acc: 0.8696
7097.6    27      Predicting
7107.3    30      Done Predicting
```

Figure 19: Epoch Training Model

Using the model to do prediction on the test set and submit to Kaggle we got 0.62120 for epoch 1 model 0.63413 for epoch 3 model and 0.63549 for epoch 5 model.



TestScript2 (version 8/8) 0.63549
3 days ago by Johnathan Wong
From "TestScript2" Script

TestScript2 (version 7/8) 0.63413
3 days ago by Johnathan Wong
From "TestScript2" Script

TestScript2 (version 6/8) 0.62120
3 days ago by Johnathan Wong
From "TestScript2" Script

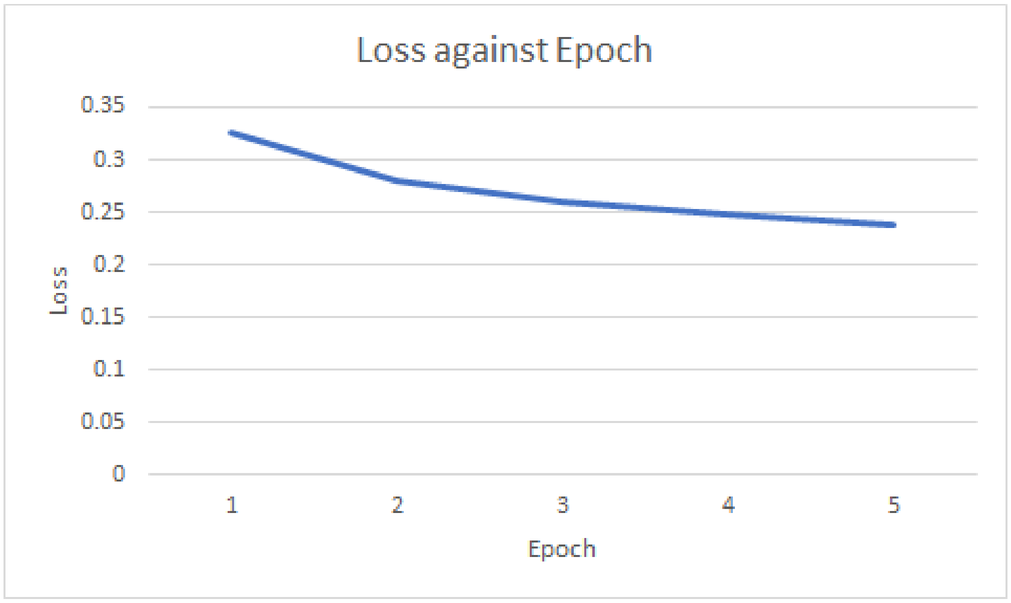


Figure 20: Loss Epoch Chart

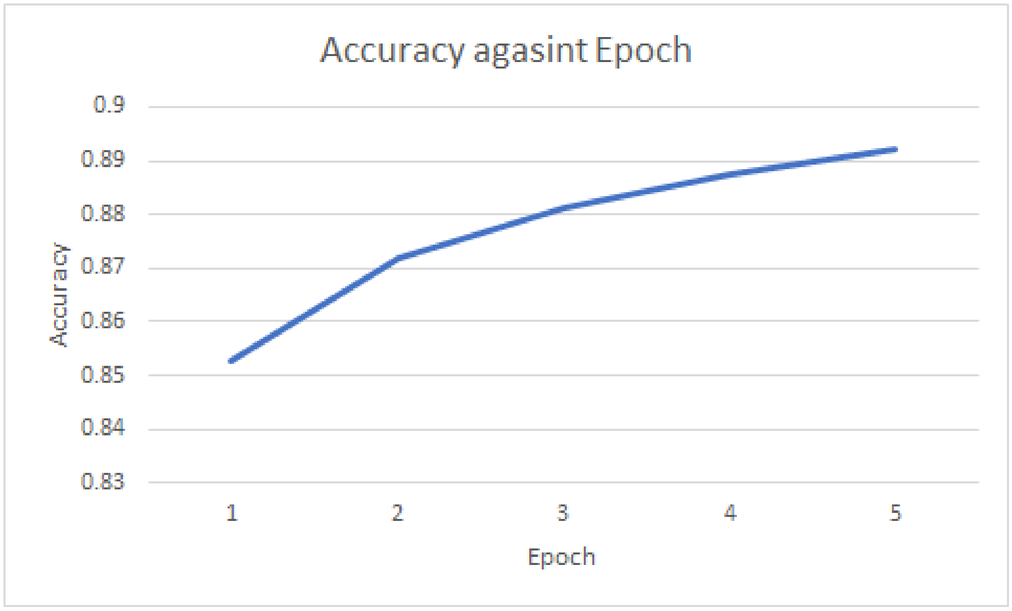


Figure 21: Accuracy Epoch Chart

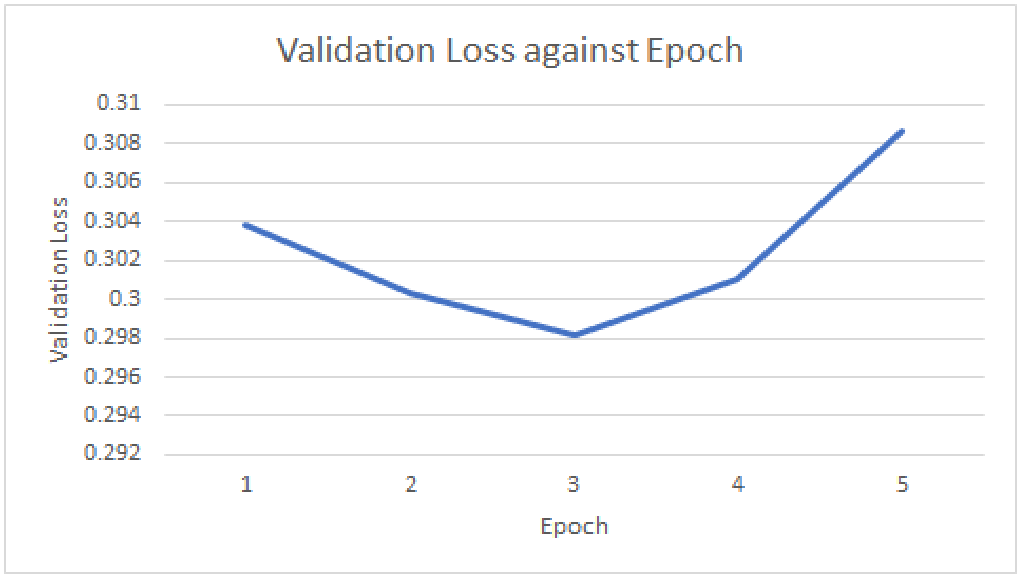


Figure 22: Validation Loss Epoch Chart

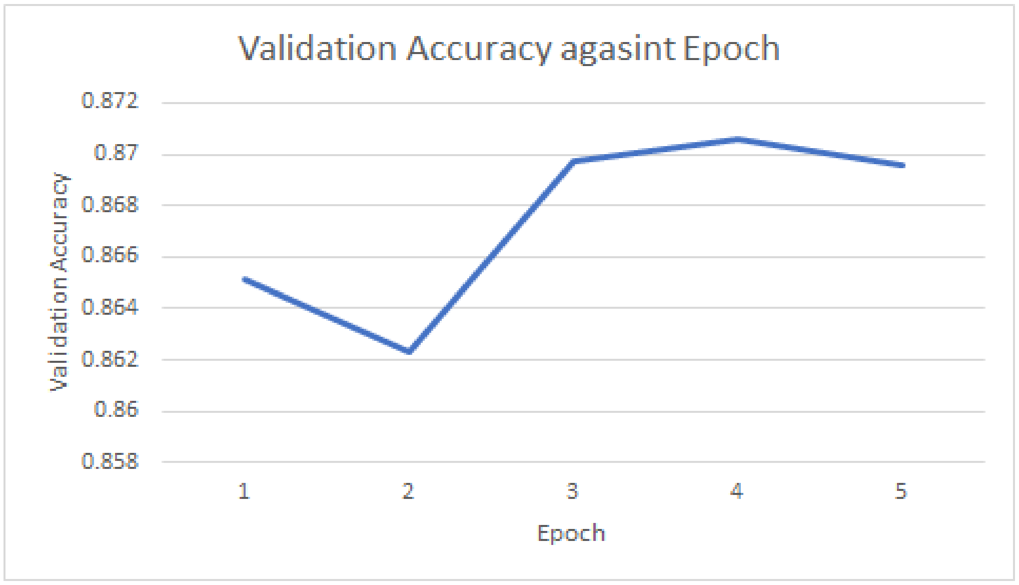


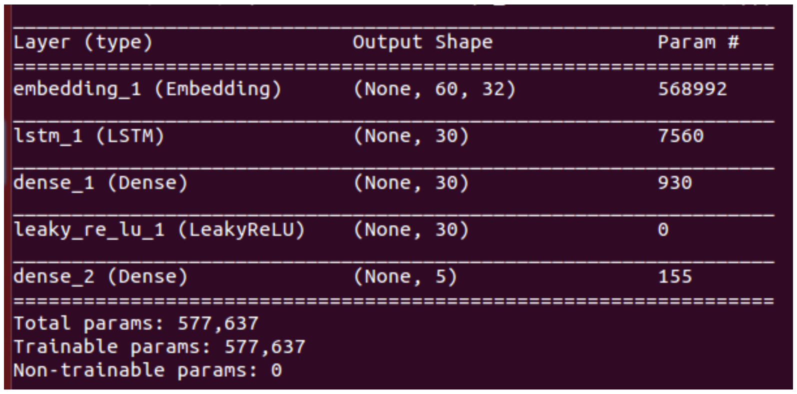
Figure 23: Validation Accuracy Epoch Chart

Word2Vec + LSTM

To implement the Word2Vec into the LSTM model, we are required to do padding base on the words of the longest review. Maximum of 60 words are used for padding. Since it is a neural network model, additions of new method will increase the neural network layer by 1.

At first we try all the possible layer we can find on Keras and discover that we are overfitting, thus we reduce the layer and feature. The dataset is a series of words connected with each other. Therefore we explore various recurrent neural network algorithm as the first layer. We tried RNN, GRN and LSTM and found that LSTM is the best fit. We use only one layer due to the complexed computing. We also disabled convolutional layer due to reduction of accuracy. Every word in sentence seems to be as vital as each other, thus and most vital info was processed in RNN layer.

After implementing embedding layer, LSTM, dense, leaky and another dense layer were add.



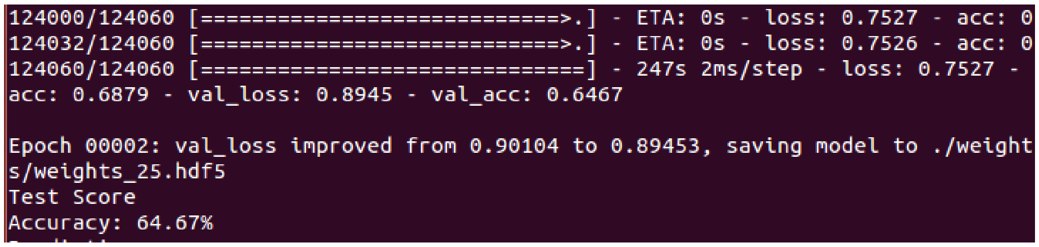
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 60, 32)            568992
_________________________________________________________________
lstm_1 (LSTM)                (None, 30)                7560
_________________________________________________________________
dense_1 (Dense)              (None, 30)                930
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 30)                0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 155
=================================================================
Total params: 577,637
Trainable params: 577,637
Non-trainable params: 0
```

Figure 24: NN Layers

Choosing Learning rate

We use stochastic gradient descent thus we have to chose learning rate for each jump. We divided data into small batches then train data for each batch, and adjust rate for each subsequence batch using Nesterov Adam algorithm. We tried 0.00001, 0.0001, 0.001, 0.01 and the second choice appear to be best for fast convergence and no overshooting for lost function value.

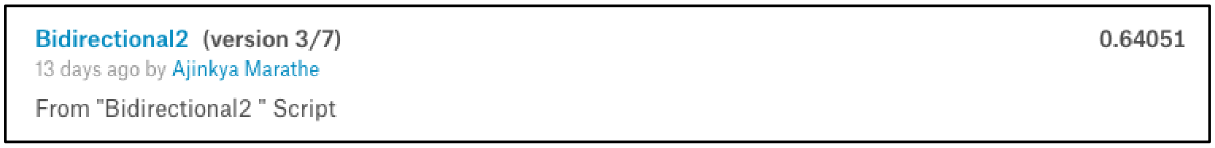
Epoch is the number of times that we train the data. we tried 1, 2, 5, 10, 20, 50. Generally the more epoch training we run the better the accuracy performance will get. But after 2 epoch we achieve 90% maximum efficiency, additional 10 epoch yields only 1% improvement.



```
124000/124060 [============================>.] - ETA: 0s - loss: 0.7527 - acc: 0
124032/124060 [============================>.] - ETA: 0s - loss: 0.7526 - acc: 0
124060/124060 [==============================] - 247s 2ms/step - loss: 0.7527 -
acc: 0.6879 - val_loss: 0.8945 - val_acc: 0.6467

Epoch 00002: val_loss improved from 0.90104 to 0.89453, saving model to ./weight
s/weights_25.hdf5
Test Score
Accuracy: 64.67%
```

Figure 25: Epoch Accuracy Test


Bidirectional2 (version 3/7)
13 days ago by Ajinkya Marathe
From "Bidirectional2 " Script

0.64051

Conclusions

After experimenting with different types of machine learning techniques, we observed that artificial neural networks perform better than almost all the other algorithms like Support Vector Machines, Naive Bayes Classifier and blackbox solutions like TextBlob Sentiment Analysis. We realised that using networks deeper than the ones we used does not increase the accuracy substantially but increases the computational cost. We also realised getting more data from other datasets having similar distributions, like the Imdb movie review dataset might help the overall accuracy.

As far as the Kaggle dataset was concerned, along with the language ambiguity generally faced in any natural language processing task, the obfuscation due to sarcasm and sentence negation was significant in sentiment analysis.

References

[1] Pang and L. Lee. 2005. *Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales*. In ACL, pages 115–124.

[2] *Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank*, Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Chris Manning, Andrew Ng and Chris Potts. Conference on Empirical Methods in Natural Language Processing (EMNLP 2013).

Appendix

https://www.kaggle.com/c/movie-review-sentiment-analysis-kernels-only