ST449 Project - Sentiment Analysis on Amazon Fine Food Reviews Dataset

Name: Shushu Lin

14 May, 2020

1 Introduction

Amazon is a well-known website for online shopping, products ranging from daily essentials to e-books and music. Hundreds and thousands of people make comments and rate the products they bought on the website. This information is highly valuable since customers could make use of such information to gain knowledge about how good the product really is from other users and avoid mismatching between the real merchandise and its online description. On the other hand, marketers could conduct analysis based on the reviews and ratings to understand the polarity of products and decide whether to order more stocks aiming for maximising profits. In recent years, various machine learning and deep learning techniques yield state-of-art results in sentiment analysis under the natural language processing (NLP) domain due to increase computer capacity [1, 2, 3]. Hence, being able to analyse text and detect the sentiment involved is demanding and plausible.

In this report, we work on Amazon fine food reviews dataset [4] and attempt to classify one's attitude towards high-quality food sold on Amazon based on food reviews. Machine learning method - navie Bayes classifier, and deep learning neural networks - recurrent neural network (RNN) with long short-term memory (LSTM), RNN with bidirectional LSTM and convolutional neural network (CNN) are considered. We benchmark these against multi-layer perceptron (MLP), a type of feed forward neural network that contains only an embedding layer and dense layers. We train the models to classify polarity of approximately 74000 food reviews and test on an additional 24000 reviews. We employ metrics such as confusion matrix and ROC curve to compare and assess model performance.

Our main goal is to find out which method is better for sentiment classification on Amazon fine food dataset. Similar techniques could be employed to various other areas, such as classifying one's attitude towards a particular political Party based on tweets posted in Twitter or analysing sentiment expressed by viewers on movie reviews.

We find that highest accuracy of classifying testing data into correct classes is achieved by RNN with bidirectional LSTM cells. Also, RNN and CNN outperform MLP by a clear margin.

The rest of the report is organised as follows. Section 2 introduces methodologies and neural networks architectures considered in this study. Section 3 offers a view on the Amazon fine food reviews dataset and pre-processing of text data. Section 4 presents results of model prediction and model performance. Section 5 concludes.

2 Methodology

2.1 Term frequency - Inverse document frequency

In order to proceed with machine learning methods such as navie Bayes classifier, we need to express text data in a sparse vector form. Here, we use term-frequency inverse document frequency (tf-idf) vector representation. This is a bag of words [5] model that encodes each word token a unique integer number and gives each word a weight. It counts the number of times each unique word appears in the document and divides it by total number of unique words in the whole document, and hence the name term frequency (tf) [6]. To avoid putting too much weights on common words and weakening the more important tokens, the weights are scaled down by a logarithm of total number of documents divides the number of documents where that unique word appears. We call this term inverse document frequency (idf) [7]. Hence, tf-idf value reflects how important a word is in the whole corpus. Each review is represented as a vector of length equals to the number of features/words used in the whole corpus, and tf-idf

values are given for words presented in that specific review and zeros for non-occurrence words. One disadvantage of tf-idf is that it does not take into account the order of the words.

2.2 Navie Bayes

Navie Bayes classifier is a simple and fast machine learning method to use. It makes a strong assumption that each word token in the review is independent of each other [8]. It makes use of conditional probability and comes under the Bayesian framework. Each review is assigned to a class $\hat{c}$ where $\hat{c}$ is either 0 (negative) or 1 (positive) in our study. Then,

$$\hat{c} = \arg\max_{c} P(c|review) \tag{1}$$

$$P(c|review) = P(c)\prod_{i=1}^{n} P(f_i|c) \tag{2}$$

where f_i represents the i^{th} word of the total n words. $P(c)$ is the prior probability for class c and $P(f_i|c)$ is the likelihood of each word given class c. In our study, $c = 2$ for binary classification.

2.3 Word Embedding

Word embedding is a technique for transforming text or documents into dense vector representation. Since a corpus contains large numbers of features, each review sequence is padded with lots of zeros. This leads to high-dimensional sparse vector representation. Word embedding is to resolve this problem, where it projects each word into continuous vector space and results in lower dimensional dense vector representation. The position, also called embedding, of the word is learned from text and relates to words surrounding it [12]. The learning process is tuned via neural network [9] and it is usually putted as the first layer of the network on text processing related tasks. Popular word embedding learning systems include Word2Vec [13] and GloVe [14], which are pre-trained embedding models available to use.

2.4 Feedforward Neural Network

Feedforward neural networks are the simplest neural network in deep learning. It consists of only input layer, hidden layers and output layers. Fig.1 shows a multi-layer perceptron network (MLP) [9], a type of feedforward neural network, where layer L1 is the input layer, L2 is the hidden layer and L3 is output layer. The lines connect neurons (represented by the circles) carry information, known as weights (w) and bias (b), from one layer to next layer, but only on forward direction. That is, there is no loop in such network. The hidden layer takes input neurons and bias (shown by +1) from the input layer, adjusts the weights and bias, and then passes them onto the fully connected output layer. The output values are normally non-linear and are transformed by an activation function to a desirable range. In our study, since we only have binary classes, we use sigmoid activation function [10] to transform inputs from hidden layer to a range between 0 and 1. Popular activation functions include hyperbolic tangent function (tanh) and rectified linear function (ReLU) [9, 11]. We will use MLP as our baseline model.

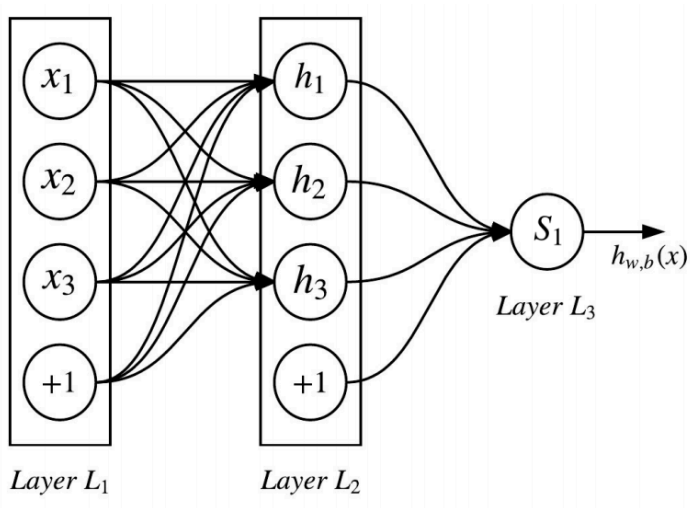

Figure 1: Feedforward neural network structure [9].

2.5 Convolutional Neural Network

Convolutional neural network (CNN) is a special type of feedforward neural network that it contains convolutional layers and mostly used in computer vision areas [9]. Convolutional layers contain filters that can extract features from data and hence it is good at dealing with spacial data. Kim [15] proposed a simple CNN that contains only one convolutional layer with pre-trained Word2Vec embedding [13]. Fig.2 shows his CNN architecture design.

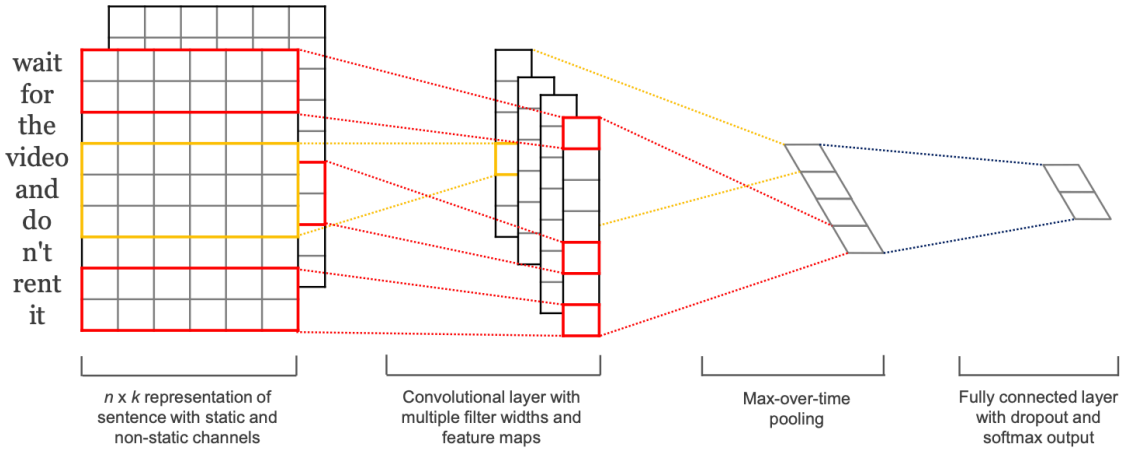

Figure 2: Convolutional neural network structure [15].

The padded sequence of words is passed into an embedding layer with pre-training Word2Vec model. Then a convolutional layer with multiple filters is applied to produce feature maps. Next, max-over-time pooling [16] is applied to the feature maps to extract the most important feature from each feature map. The outputs from the max pooling layer are then concatenated and pass onto a fully connected layer with softmax as activation function to produce a probability distribution over class labels. Dropout and L2-regularizer [17] are used to reduce over-fitting.

This simple CNN is shown to be competitively better than many more sophisticated models and achieves state-of-the-art results in various dataset.

In this project, I make use of the CNN architecture (Fig.3) that is proposed by Zhang [18], which specifies the hyperparameters to use for training that do not set up clearly in Kim's paper. This design makes use of 6 filters of 3 different sizes (2, 3, 4) and dropout rate 0.5 with l2-norm constraint 3.

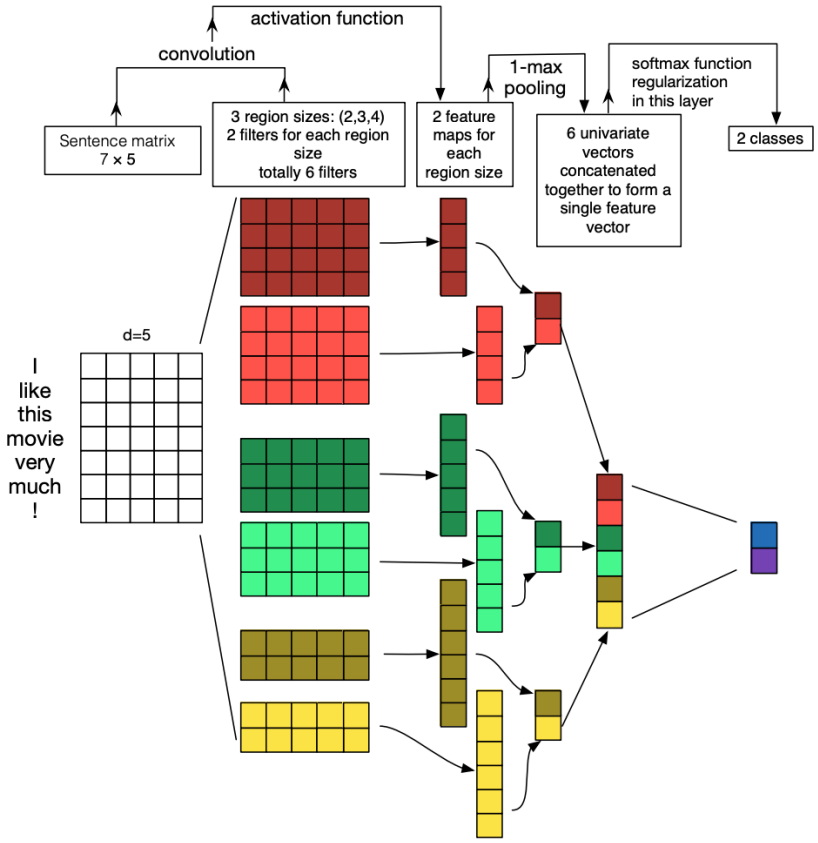

Figure 3: Convolutional neural network structure with hyperparameters specified [18].

2.6 Recurrent Neural Network

Recurrent neural network (RNN) [20] is a network that connections between neurons are directed. That is, unlike feedforward neural network, RNN can form a loop and so it can make use of information from previous nodes and combine with information from current nodes. Fig.4 [19] shows an unfolded RNN structure on the right. The network has total layers (time steps) that are the same as input length [9]. In our study, it will be equal to number of words in the reviews. x_i is the input at time step t and h_i is the hidden state (memory), which it is calculated based on information from all previous hidden states and also the current input. However, RNN suffers from vanishing gradient [21] problem due to long sequences. That is, it can only combine information from few steps before.

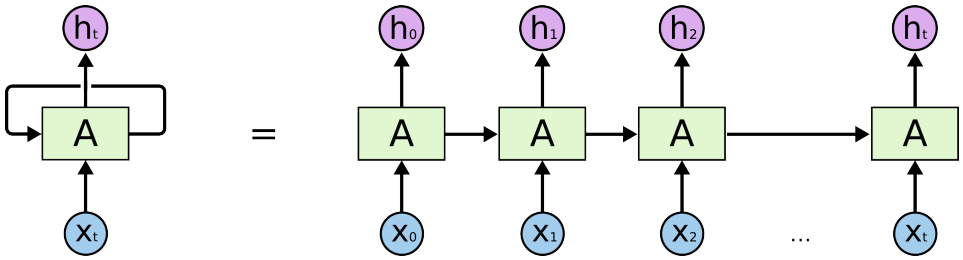

Figure 4: Recurrent neural network structure [19].

2.7 Bidirectional RNN

A bidirectional RNN [22] sticks two RNNs on top of each other, where one is used to take information from previous hidden states and the other makes use of information from future. The output y_i at each time step is calculated based on the two RNNs working in different directions. Fig.5 [23] shows an architecture of the bidirectional RNN network. This network is particularly useful in predicting future word in the sentence. In our study, future word token in the review might affect weights put on the previous word, and we use such network to establish a connection between them.

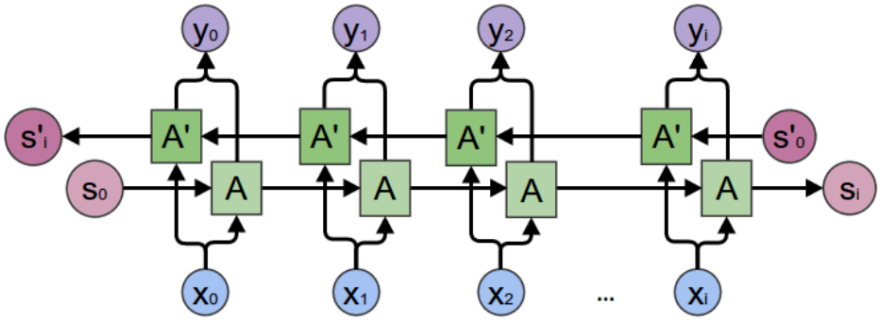

Figure 5: Bidirectional recurrent neural network structure [23].

2.8 Long Short-Term Memory

Long short-term memory network (LSTM) [24] is a special type of RNN, which could be used to deal with vanishing gradient problem [21]. It is flexible since it can control what information to forget and what to carry onto the next time step. Fig.14 [29] shows a cell structure of a LSTM network at time step t. LSTM has hidden states $h(t_i)$ and cell states $C(t_i)$, where cell states are used to store new information. At each time step, a sigmoid function f transforms the combination of outputs from previous hidden state $h(t-1)$ and current input $x(t)$ into a value in the range between 0 and 1, where 0 means delete all information from the old cell state $C(t-1)$ and 1 means keep all information. This is the forget gate showing on the left of the cell in Fig.14. The second operation is to apply sigmoid function, called a input gate i, on $h(t-1)$ and $x(t)$ again to decide what values to update from $h(t-1)$ and $x(t)$. The third operation proceeds as that a tanh function creates a vector of new values $\hat{C}$ which will be added to the cell

state at next stage. Combining i and $\hat{C}$ together, we will know what information to update for time step t. The old cell state is then updated by adding $f * C(t-1)$ with $i * \hat{C}$. The last step is to decide outputs from current time step. This is done by applying a sigmoid function o to $x(t)$ and $h(t-1)$ to decide what to output. This is called the output gate. And then a tanh function is placed on the new cell state $C(t)$ which is then multiply with the value from output gate to get the outputs $h(t)$ for time step t. In this study, we will try both RNN with one directional

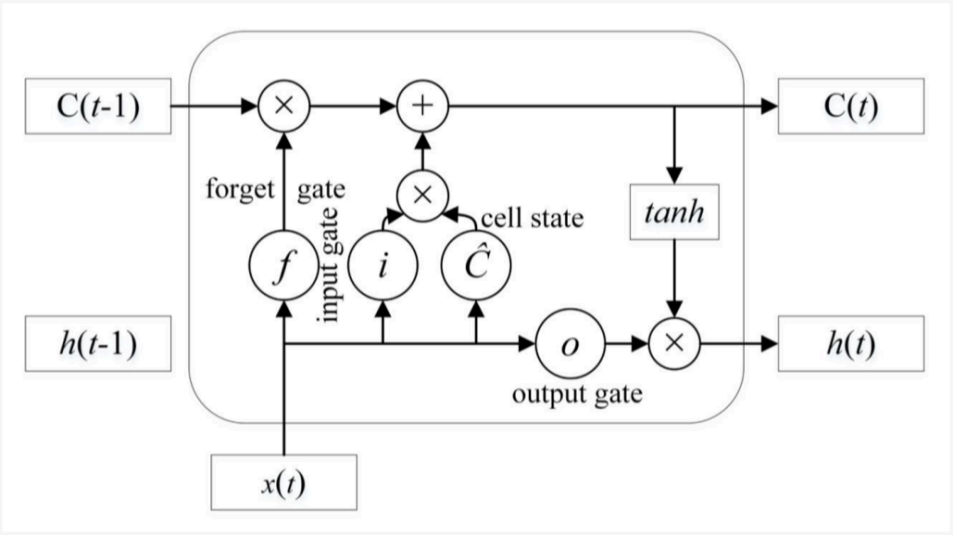

Figure 6: Recurrent neural network with LSTM cell structure [29].

LSTM and RNN with bidirectional LSTM models and see which performs better.

3 Data

The dataset used in this study is Amazon fine food reviews dataset that can be downloaded from Kaggle [4]. The original dataset contains 10 columns including information such as product ID, user ID and helpfulness of reviews. I drop all columns except the text and score columns since my main goal is to classify sentiment involved in the review text into positive and negative polarity. I rename the two columns with reviews and rating respectively.

Reviews with rating 3 are treated as neutral attitude and will not be considered in this project Therefore, they can be deleted. Reviews with rating greater and equal to 4 will be classified as positive sentiment and encoded with value 1. Reviews with rating less and equal to 2 will be treated as negative and encoded as 0. Duplicated reviews and rows with null values are also checked and removed.

Before data cleaning and splitting the dataset, number of data involved in each class are counted. The left-hand side of Fig.7 shows number of data in each class before sampling. We can see that there are almost 6 times more positive class data than negative one. If we proceeded with such highly imbalanced dataset, then any results would be severely bias towards the majority class, in our case, the positive one. For simplication, I intend to use a balanced dataset and hence I decide to down sample the majority class to have same number of data as the minority class. The right-hand side of Fig.7 indicates the data are balanced after down sampling. Originally there are 568454 samples in the dataset and after pre-processing with the above steps, the resulting data frame has 114168 numbers of data.

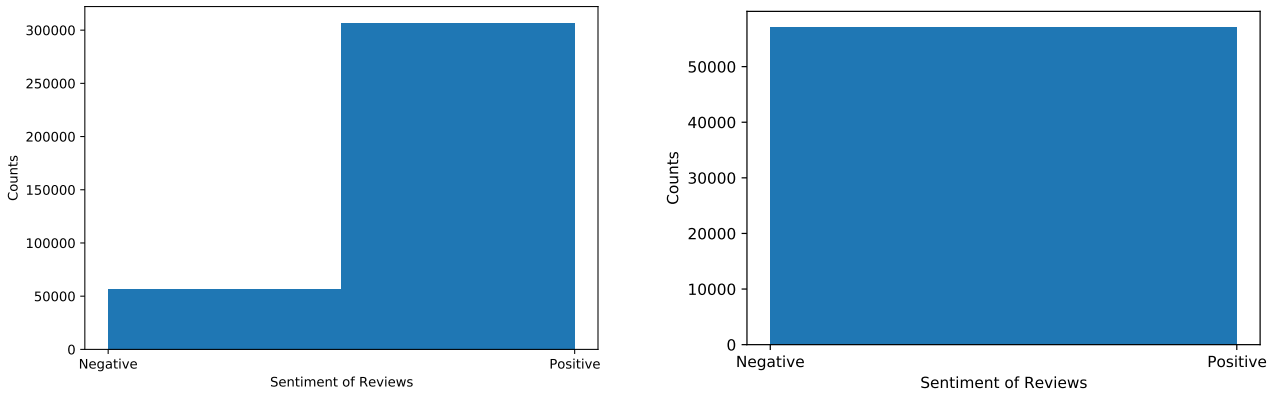

Figure 7: Histogram of number of data on each class before down sampling (left) and after down sampling (right).

Next is to clean the review text. This plays an important role on natural language processing as clean reviews not only reduce noise of the data but also increase the speed of model runs and make training more efficient. To proceed, I first tokenise, lowercase, remove hashtags, punctuations, numbers and non-English characters from the text. I also expand the contracted words such as 'aren't' back to 'are not' and customise a list of stop words such that it does not include 'not' and 'no' since these two are important negation words where their removal could turn negative meaning into positive. The customised stopwords are then removed. Finally, I lemmatise word tokens so that each word is expressed in its base form. We conduct lemmatisation instead of stemming because stemming only removes the last few characters of words that might result in spelling errors and meaningless words [25]. The word tokens are then joined together and ready for train and test split. Here is an example of an original review and a clean review:

Original - *I bought these to put my puppies vitamins in and he just doesn't like them!*
Clean - *bought put puppy vitamin not like*

Next, I first split the dataset into training and testing sets based on the 70-30 splitting ratio. The training set is again splitting into a real training set and a validation set that can be used to tune parameters during training. The dataset size for training, testing and validation sets are: 73067, 18267 and 22834.

Word clouds can be generated to gain a view on the most frequent occurring words in positive and negative classes on training set. Fig.8 shows the negative (left) and positive (right) class word clouds in the training dataset. We can see that words like 'make', 'use' and 'one' appear to be the most frequently used words in both classes. Positive words such as 'well', 'good' and 'love' are shown on the negative reviews as well. This can be caused by looking only at a single word for each pass through the reviews, where the network cannot catch bigrams expression such as 'not good' and 'not well'. We hope that the RNN with LSTM network, which I am going to implement in next section, can catch the negative expression prior to the positive words in the negative review. One other possibility is that the review contains both good and negative comments, but negative factors dominates the positive one.

After data pre-processing and having a glimpse at the training dataset, we are ready to proceed to the implementation in next section.

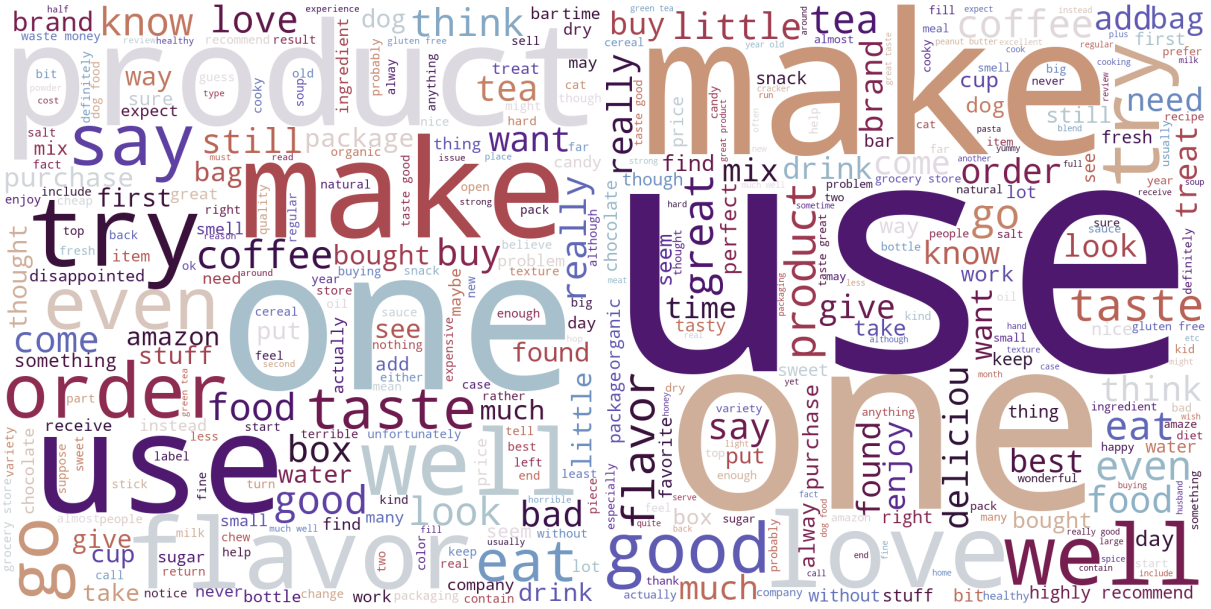

Figure 8: Word clouds for reviews with negative sentiment (left) and positive sentiment (right) in training dataset.

4 Results

4.1 Model Prediction

Firstly, I implement a machine learning method - navie Bayes classifier to the training dataset. We make use of MultinomialNB() function with smoothing parameter 1.0 from sklearn library to fit the model. The text is transformed into a sparse vector using TF-IDF prior to fitting the classifier. This yields 85.71% accuracy on testing dataset and it is fast to train and produce high accuracy.

We then turn to construct deep learning models. For all the deep learning models, we use dense vector form to represent our reviews. Each review is padded into a sequence of length 100. I choose to use sequence length of 100 since as shown in Fig.9, majority of the reviews have number of words less than 200 and the average number of tokens per review is 41.8. Counting number of reviews with token length less than 100, there appears to be 68456 out of 73067 being true. Therefore, it is reasonable to set sequence length to 100 and this can reduce noise on training.

The first model we considered is a multi-layer perceptron (MLP) model. The model contains one embedding layer for word embedding, one flatten layer to reshape output from the embedding layer and one dense layer with output dimension 64 and one output dense layer. The outputs from the penultimate layer are passed onto the last fully connected dense layer by applying sigmoid activation function. The network is trained on mini-batches, each with size 128 and is run for 20 epochs. The optimiser we choose to minimise the binary cross-entropy loss is Adam (adaptive moment estimation) [26].

We first implement the model with random initial embedding weights, which yields accuracy of 67.79% with loss 1.06. Since this is the simplest deep learning model considered in this project and it is super-fast to train, we will use it as our baseline model.

We next consider using pre-trained model GloVe [cite 13] for word embedding. I choose embedding dimension to be 100. All other layers are the same as our baseline model. This results in

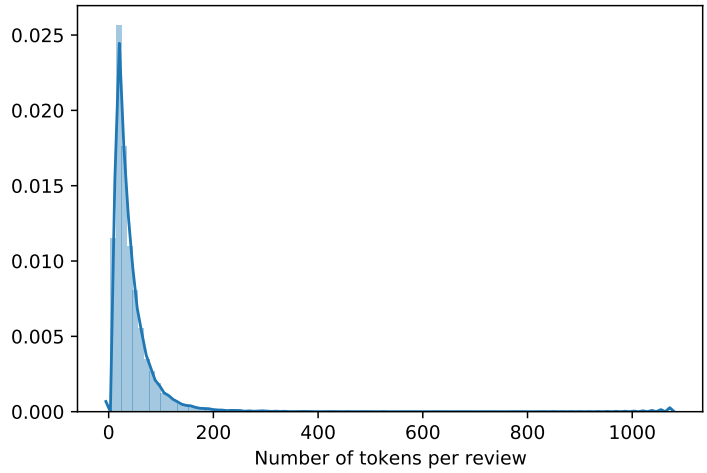

Figure 9: Distribution of review length.

76.86% test accuracy with loss 1.67. We can see that by using a pre-trained embedding model, the accuracy is increased by about 9%. However, the loss has also increased. Fig.10 shows accuracy (left) and loss (right) on train and validation sets during training. We can see that the model starts over-fitting at about the very first epoch where the validation loss is larger than the training loss and it starts to shoot up very quickly. The training accuracy achieves almost 100% after 20 epochs and is much larger than the validation accuracy.

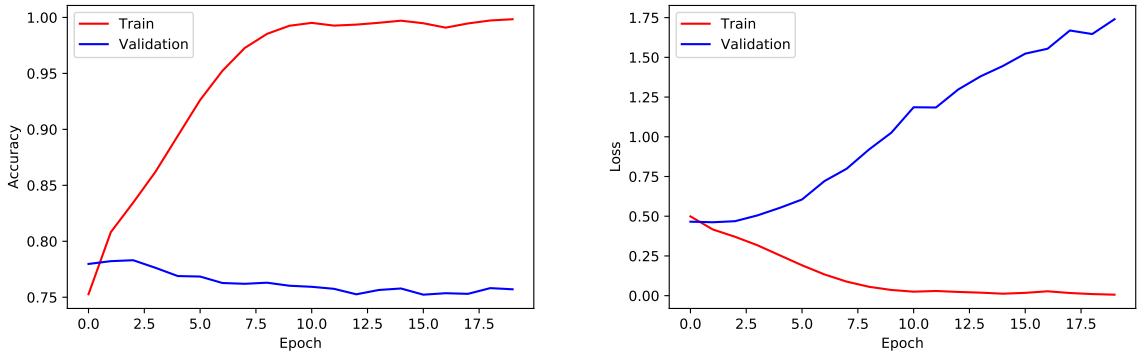

Figure 10: Accuracy (left) and loss (right) for train and validation datasets during training for MLP model with pre-trained GloVe embedding.

To reduce over-fitting, I add a dropout layer after embedding layer with rate 0.3 and one after the first dense layer with rate 0.5 on the network. This yields 78.76% test accuracy with loss 0.47. The accuracy has improved by about 2% and loss has been significantly reduced from 1.67 to 0.47. Fig.11 shows plots of accuracy and loss for training and validation set. Compare to Fig.10, we can see the gap between training and validation loss is narrower and also the model starts over-fitting at a later epoch. This indicates that dropout layer is an effective regularisation technique to employ for over-fitting and I will add dropout layers for models introduce later.

We also try to fit different optimisers such as root mean square propagation (RmsProp) [27] and adaptive gradient algorithm (AdaGrad) [28] for training MLP with GloVe and dropout defined above. Table 1 shows the testing accuracy and loss for each of the three optimisers. We can see

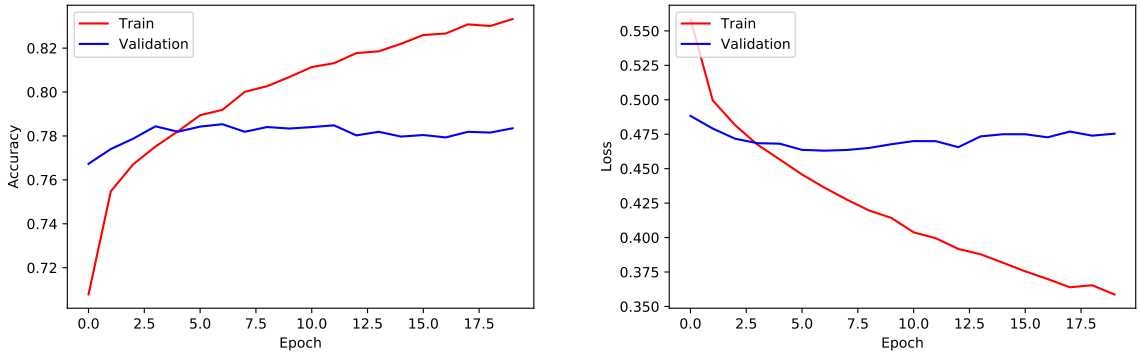

Figure 11: Accuracy (left) and loss (right) for train and validation datasets during training for MLP model with dropout and pre-trained GloVe embedding.

that AdaGrad gives the highest accuracy and lowest loss. The training process has been run for multiple times to check whether this occurs purely by chance and the result showing that highest accuracy is always achieve by AdaGrad in this study. Fig.12 offers an intuitive view on the training (solid lines) and validation loss (dashed lines) for the three optimisers. We can see that AdaGrad has the lowest validation loss and closest to its training loss. Hence, I decide to use AdaGrad as our optimiser for training for rest of the models.

	Testing Accuracy	Testing Loss
RmsProp	0.781773	0.500891
AdaGrad	0.790335	0.455012
Adam	0.788167	0.466988

Table 1: Results of running MLP models with three different optimisers.

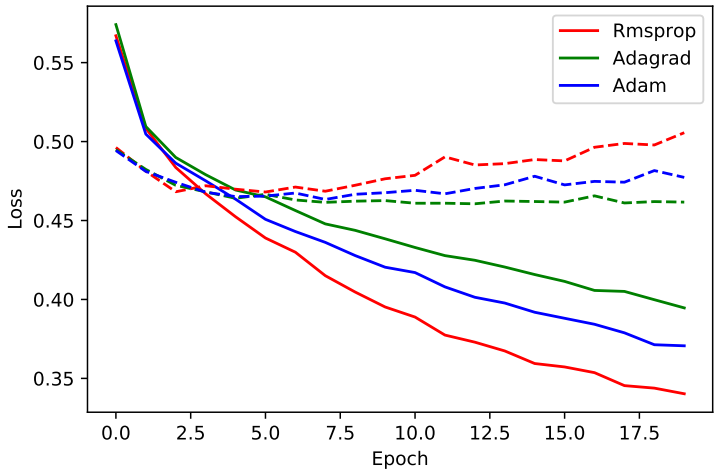

Figure 12: Training loss (solid lines) and validation loss (dashed lines) for MLP model with three different optimisers.

Next, I construct an RNN with LSTM for classification. The output dimension of LSTM layer is set to 128 first, and dropout is applied inside the LSTM layer with dropout rate 0.2. We add a dropout layer with rate 0.5 after the LSTM layer as well for more regularisation. The entire

model is run for 10 epochs with mini-batch size 128. The resulting test accuracy is 89.21% with loss 0.28.

In order to select the best model, we fine tune the LSTM network output dimension. We run the model with output dimensions 64, 128, 256 and 512 each for 10 epochs with mini-batch size 128. Table 2 shows a table of results for the 4 dimensions. We can see from Fig.13 that the accuracy is increasing as the number of output dimension increases. Notice that the loss is decreasing for the first three but for the last one it has a slight bounce back. We run the same algorithm for several times, the loss is very similar for the last two models and their relative relationship appears randomly by chance, but the accuracy is always higher for the 512-output dimension model. Furthermore, doubling the output dimension requires 1.5 times more time to train the model. For example, it takes about 40 mins to run LSTM network with output dimension 256 but takes 60 mins to run the 512-output dimension model using TPU on Google colab. Hence, there is a trade-off between model accuracy and model training time.

	Testing Accuracy	Testing Loss
Dim 64	0.888697	0.281951
Dim 128	0.892091	0.276355
Dim 256	0.896623	0.260134
Dim 512	0.897762	0.269170

Table 2: Results of running RNN with LSTM models, where LSTM output dimension varies.

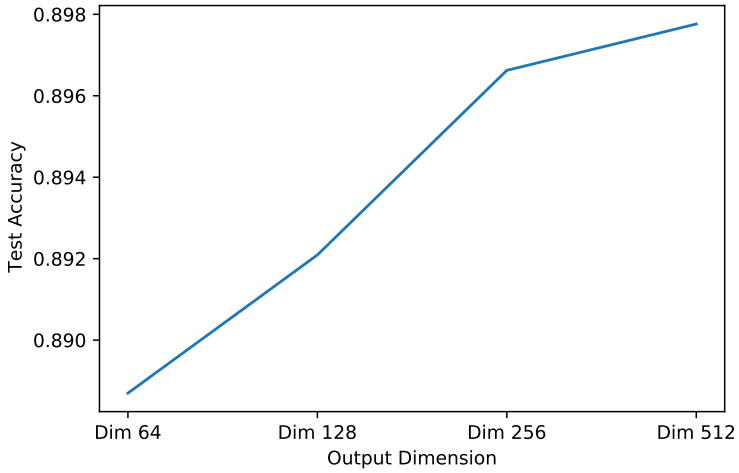

Figure 13: Testing accuracy of running RNN with LSTM models, where LSTM output dimension varies.

Fig.14 shows the accuracy and loss for training and validation sets for RNN with LSTM (512) model. The model does not suffer from significant over-fitting problem.

We now turn to train RNN with bidirectional LSTM cells. Almost the same setting as that in single directional LSTM model except that this time we use a bidirectional layer with two LSTM cells wrapped inside. The output dimension of each LSTM cell is set to 512 and the model is run for 10 epochs with mini-batch size 128. This leads to 89.86% test accuracy with loss 0.27. The running time for the bidirectional LSTM model is 1.5 times more than the one directional LSTM model. Fig.15 summarises the training process.

Finally, we implement a convolutional neural network under the architecture designed by Kim

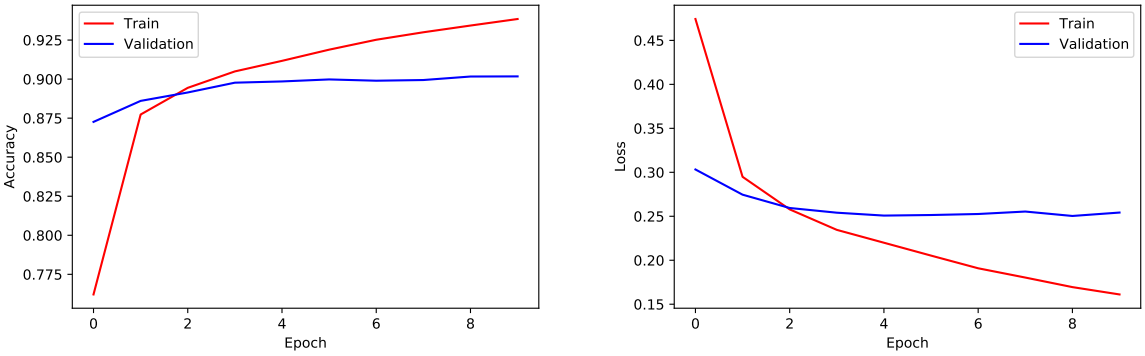

Figure 14: Accuracy (left) and loss (right) for train and validation datasets during training for RNN with LSTM cell, where output dimension of LSTM cell is 512.

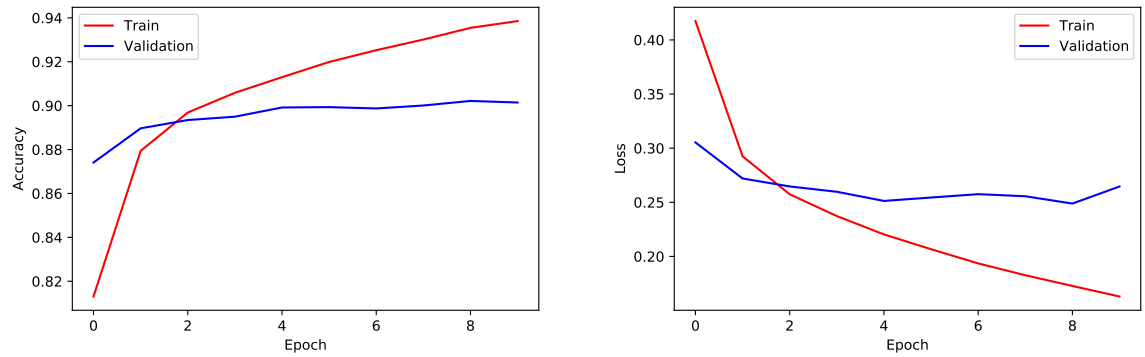

Figure 15: Accuracy (left) and loss (right) for train and validation datasets during training for RNN with bidirectional LSTM cells.

[15] and hyperparameters chosen by Zhang [18]. One difference is that I do not use Word2Vec [13] for pre-trained vector embedding but GloVe [14]. The model is trained for 20 epochs with mini-batch size 50. This model leads to test accuracy of 88.67% with loss 0.31. Compare to RNN with LSTM model and the bidirectional LSTM model, the CNN is approximately 6 times faster than the former and 11 times faster than the latter, but with only 1% test accuracy drop. Hence, CNN could be employed if we do not have high computer capacity.

We also increase the number of filters from 128 to 256 and the resulting test accuracy is 88.57% with loss 0.31. Change filter sizes from (3, 4, 5) to (5, 6, 7) yields 88.84% accuracy and 0.31 loss. Fig.16 gives a visualisation of accuracy and loss during training for the CNN model with filter sizes (5, 6, 7). It also does not suffer from huge over-fitting problem.

We summarises all the results of test accuracy and loss for model introduced above in Table 3.

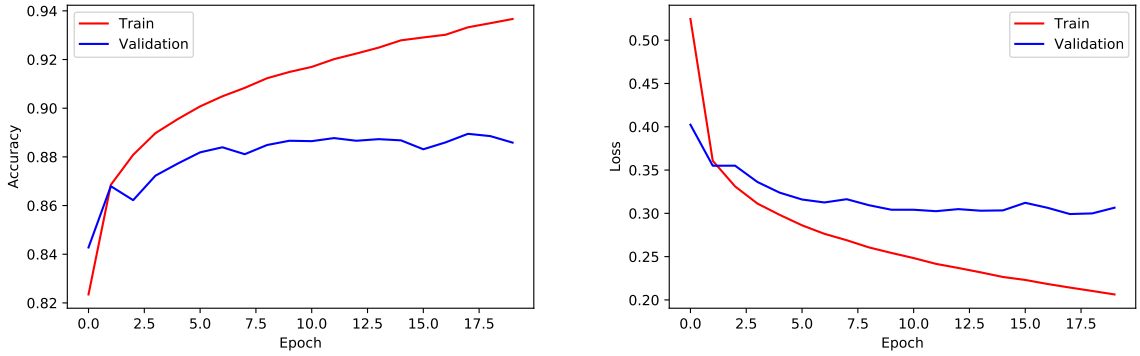

Figure 16: Accuracy (left) and loss (right) for train and validation datasets during training for CNN with filter sizes (5, 6, 7).

	Testing Accuracy	Testing Loss
Navie Bayes	85.71%	-
MLP	67.79%	1.06
MLP with GloVe	76.86%	1.67
MLP with Dropout	78.76%	0.47
MLP with AdaGrad	79.03%	0.46
RNN with LSTM(512)	89.78%	0.27
RNN with Bi-LSTM	89.86%	0.27
CNN	88.84%	0.31

Table 3: Summary of results for testing accuracy and loss for all models introduced.

4.2 Model Assessment

Apart from using test accuracy as our model measurement, we could judge whether a model is capable of conducting classification by looking at confusion matrix and receiver operating characteristic (ROC) curve. Confusion matrix offers an intuitive visualisation of numbers of data being classified into each class. From the confusion matrix, precision, recall and F1 score can also be calculated. As for the ROC curve, it is a curve showing the relative relationship between true positive rate (TPR) and false positive rate (FPR). In general, a skilful model shall have high TPR and low FPR, whereas a non-skilful model would have almost the same TPR and FPR as it conducts classification by random guessing [30]. This is more intuitive by looking at plots of ROC curve later. The area under ROC curve, called AUC, evaluates the model performance numerically. It calculates the area under the ROC curve and in general, the higher the value, the better the model. We will assess the performance of RNN with bidirectional LSTM and CNN models introduced before, since they give the highest accuracy among all models.

Fig.17 gives the confusion matrix of the bidirectional LSTM model (left) and the CNN model (right). We can see that both of them give almost the same numbers of true predictions (predict negative as negative and positive as positive). However, bidirectional LSTM has more false positive (FP) number of data, showing on upper right corner, than false negative (FN) one, indicated on lower right corner. This leads to higher recall (0.91) than precision (0.89). The F1 score, which is a balanced measurement between recall and precision, is 0.90. As for the CNN model, it has higher FN than FP which leads to higher precision (0.90) than recall (0.88) with F1 score being 0.89. Hence, RNN with bidirectional LSTM performs slightly better

than CNN if compared the F1 score.

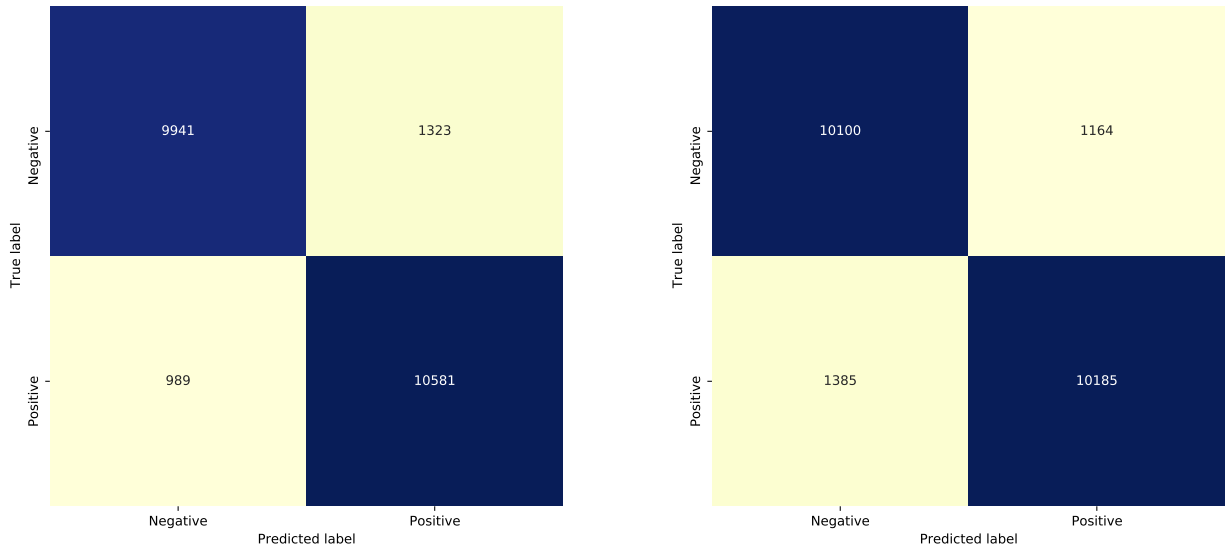

Figure 17: Confusion matrix for bidirectional LSTM model (left) and CNN model (right).

Next, we look at the ROC curve for both models. The left-hand side of Fig.18 shows the ROC curve for bidirectional LSTM network and the right-hand side for the CNN model. We stated before that a skilful model generally has high TPR and low FPR, which translates to graphical language is that the curve should travel from lower left corner, through the upper left corner and arrive at the upper right corner [30]. Both plots indicate these features. To evaluate numerically, AUC for these models are 0.96 and 0.95 respectively. Again, it confirms the fact that they have good ability for classification and RNN with bidirectional LSTM performs slightly better.

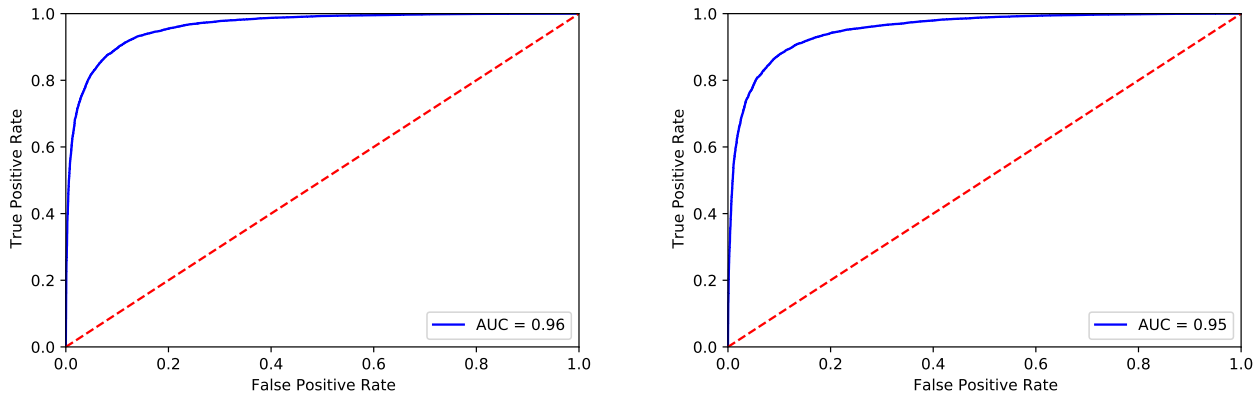

Figure 18: ROC curves for bidirectional LSTM model (left) and CNN model (right).

5 Conclusion

To sum up, this study explores various modern machine learning and deep learning models to conduct classification for sentiment analysis on Amazon fine food reviews dataset. We have found that using a pre-trained model for embedding improves model accuracy by about 9% as compared to random initialisation. Also, recurrent neural network with bidirectional long short-term memory cells and convolutional neural network are proven to be efficient tools for sentiment classification based on text. They achieve 89.86% and 88.86% accuracy on the classification task, which are approximately 22% more accurate than baseline multi-layer perceptron model. Both models give high F1 score (0.90 and 0.89 respectively) and AUC values (0.96 and 0.95 respectively) with almost perfect fit of ROC curves, which indicate they are skilful models to use for sentiment classification.

Further study could be placed on fine-tuning model hyperparameters and ensemble modelling to achieve higher prediction accuracy. Moreover, we have down sampled to achieve balanced dataset in this study. Future research could investigate into imbalanced dataset as most of the real-world datasets are imbalanced.

Overall, the results support the notion that recurrent neural network with LSTM and convolutional neural network are good techniques to use for sentiment analysis.

References

[1] Goodfellow I, Bengio Y, Courville A. Deep learning. The MIT Press. 2016.

[2] Collobert R, Weston J, Bottou L, Karlen M, Kavukcuoglu K, and Kuksa P. Natural language processing (almost) from scratch. Journal of Machine Learning Research, 2011.

[3] Goldberg Y. A primer on neural network models for natural language processing. Journal of Artificial Intelligence Research, 2016.

[4] J. McAuley and J. Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. WWW, 2013.

[5] Z.S.Harris. 1981. Distributional Structure. In: HizH. (eds) Papers on Syntax. Synthese Language Library (Text and Studies in Linguistics and Philosophy), vol 14. Springer, Dordrecht.

[6] H. P. Luhn, "A Statistical Approach to Mechanized Encoding and Searching of Literary Information," in IBM Journal of Research and Development, vol. 1, no. 4, pp. 309-317, Oct. 1957, doi: 10.1147/rd.14.0309.

[7] Sparck Jones, K. (1972), "A Statistical Interpretation of Term Specificity and its Application in Retrieval", Journal of Documentation, Vol. 28 No. 1, pp. 11-21.

[8] Narasimha Murty, M.; Susheela Devi, V. (2011). Pattern Recognition: An Algorithmic Approach. ISBN 978-0857294944.

[9] Zhang, Lei, Wang, Shuai, Liu, Bing. (2018). Deep Learning for Sentiment Analysis : A Survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 10.1002/widm.1253.

[10] Han, Jun; Morag, Claudio (1995). "The influence of the sigmoid function parameters on the speed of backpropagation learning". In Mira, Jos; Sandoval, Francisco (eds.). From

Natural to Artificial Neural Computation. Lecture Notes in Computer Science. 930. pp. 195201

[11] R.Hahnloser, R.Sarpeshkar, M.A.Mahowald, R.J.Douglas, H.S.Seung. (2000). "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit". Nature. 405 (6789): 947951.

[12] J.Brownlee, 2019. How To Use Word Embedding Layers For Deep Learning With Keras. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/ [Accessed 9 May 2020].

[13] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NIPS 2013.

[14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

[15] Y.Kim. Convolutional neural networks for sentence classification. In Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL 2014), 2014.

[16] Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuglu, P. Kuksa. 2011. Natural Language Processing (Almost) from Scratch. Journal of Machine Learning Research 12:24932537.

[17] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. CoRR, abs/1207.0580.

[18] Y. Zhang, B. Wallace. 2015. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. CoRR,abs/1510.03820.

[19] Colah.github.io. 2015. Understanding LSTM Networks – Colah's Blog. [online] Available at: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ [Accessed 11 May 2020].

[20] J.L.Elman. Finding structure in time. Cognitive Science, 1990.

[21] Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press, 2001.

[22] M. Schuster, K.K.Paliwal. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 1997.

[23] Colah.github.io. 2015. Understanding LSTM Networks – Colah's Blog. [online] Available at: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ [Accessed 11 May 2020].

[24] Cite 23: S. Hochreiter, J.Schmidhuber. Long short-term memory. Neural Computation, 9(8): 1735-1780, 1997

[25] Machinelearningplus.com. 2019. Lemmatization Approaches With Examples In Python Machine Learning Plus. [online] Available at: https://www.machinelearningplus.com/nlp/lemmatization-examples-python/ [Accessed 11 May 2020].

[26] K.Diederik, J.Ba. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.

[27] Hinton, Geoffrey. "Lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude" (PDF). p. 26. [Accessed 11 May 2020].

[28] Duchi, John; Hazan, Elad; Singer, Yoram. (2011). Adaptive subgradient methods for online learning and stochastic optimization. JMLR. 12: 21212159.

[29] M.Liu, Y.Huang, Z.Li, B.Tong, Z.Liu, M.Sun, F.Jiang, H.Zhang. 2020. The Applicability of LSTM-KNN Model for Real-Time Flood Forecasting in Different Climate Zones in China. Water. 12. 440. 10.3390/w12020440.

[30] Jason Brownlee. 2018. How to Use ROC Curves and Precision-Recall Curves for Classification in Python. [online]. Available at: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/ [Accessed 11 May 2020].