Sentiment Analysis Using Deep Learning (Review 2)

Sima Shafaei & Adriana Ontiveros Gonzalez

Mar 31

1 Review Calendar Activities

Figure 1 shows the updated Gantt chart of the activities that had been planned for the project. The index of colors on the right side of the chart indicates the percentage of completion of the tasks, so those tasks 100% completed, are shown in lighter green, and those with 0% completion are shown in darker green.

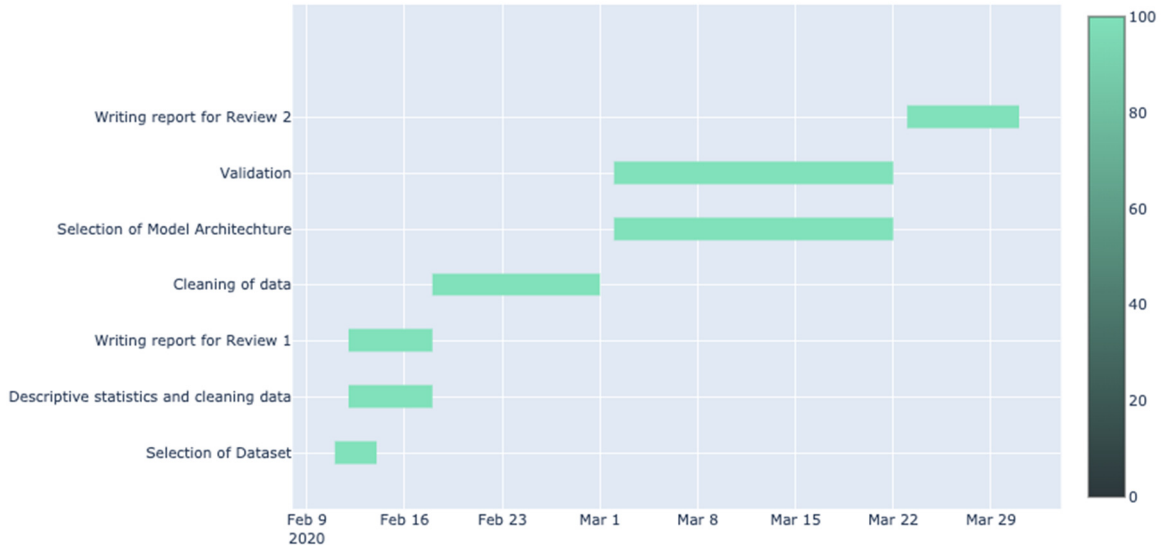
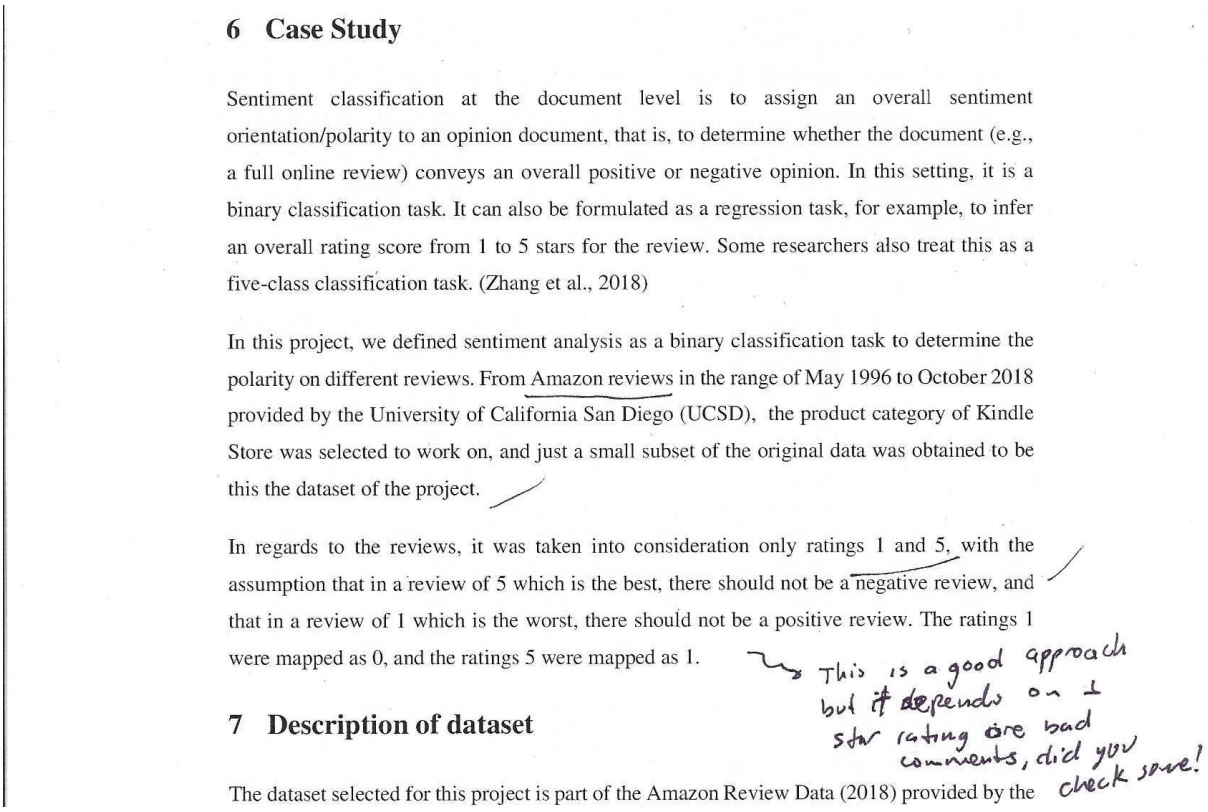


Figure 1. Gantt chart of activities for the Final Project.

2 Corrections for Review #1

Figure 2 shows the comments made for the Review #1, and as it was commented in section #7 of this review, the dataset selected for this project is part of the Amazon Review Data (2018) provided by the University of California San Diego (UCSD), which includes 233.1 million of Amazon's client's reviews divided by product categories, these reviews are in the range May 1996 to October 2018 (Ni, Li, & McAuley, 2019).

Based on Amazon, the rating system is 1 to 5 stars, where 5 stars mean the best rating and 1 star means the worst rating (Amazon.com, Inc, 2020). To verify, also a sample of the dataset was obtained and the reviews with 1 star were negative and the ones with 5 stars were positive.



6 Case Study

Sentiment classification at the document level is to assign an overall sentiment orientation/polarity to an opinion document, that is, to determine whether the document (e.g., a full online review) conveys an overall positive or negative opinion. In this setting, it is a binary classification task. It can also be formulated as a regression task, for example, to infer an overall rating score from 1 to 5 stars for the review. Some researchers also treat this as a five-class classification task. (Zhang et al., 2018)

In this project, we defined sentiment analysis as a binary classification task to determine the polarity on different reviews. From Amazon reviews in the range of May 1996 to October 2018 provided by the University of California San Diego (UCSD), the product category of Kindle Store was selected to work on, and just a small subset of the original data was obtained to be this the dataset of the project.

In regards to the reviews, it was taken into consideration only ratings 1 and 5, with the assumption that in a review of 5 which is the best, there should not be a negative review, and that in a review of 1 which is the worst, there should not be a positive review. The ratings 1 were mapped as 0, and the ratings 5 were mapped as 1.

This is a good approach but it depends on 1 star rating are bad comments, did you check some!

7 Description of dataset

The dataset selected for this project is part of the Amazon Review Data (2018) provided by the

Figure 2. Comments for Review #1.

An additional modification to Review #1 that was decided to be done, was that for the cleaning of the text, as it was seen that the word "book" had the same frequency in both, the positive and negative reviews, that word was taken out from the reviews, with the purpose of trying to get a model as simple as possible.

3 The advance of the Project – Proposed Activities

In this section, the description of the model that was used will be given, including the description of the metrics and its mathematical formulation.

Before the model was created, the reviews were split stratified on the y-axis which is the feature containing the polarity, this is binary and set to 1 for the positive reviews and 0 for the negative reviews.

The complete and cleaned set was split to get from it the 10% for testing, and the remaining 90%, was split on 90% from it for the training and the remaining 10% was for the validation. In the end,

75,483 reviews were for training, 9,319 were for testing and 8,387 were for the validation. (Figure 3)

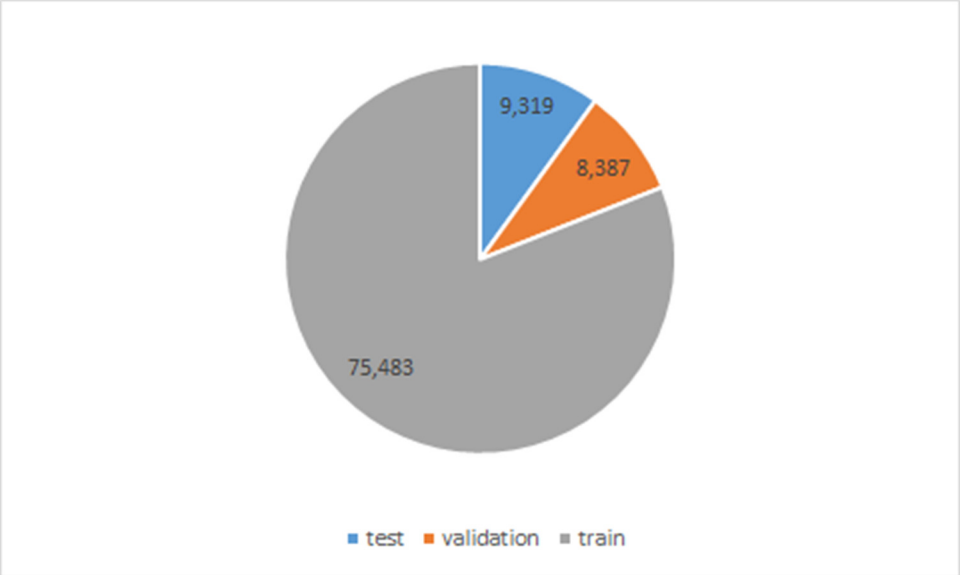


Figure 3. Number of samples in train, test and validation sets

The words included in the reviews had to be tokenized, so Keras was used, and the first 2,000 most common words included in the training reviews were tokenized, so the Tokenizer() function was called and the number of words was set to 2,000, then this tokenizer was fit on the training set and the word index was created, finally, the training reviews which were strings were then converted into lists of integer indices. This same fit on the training reviews was used to convert the validation and the test reviews into a list of integer indices.

3.1 Description of the Model

The complete architectural model used for this project is shown in Figure 4. In this section, it will be explained why each part of the model was selected, and in the next section, why each hyperparameter was selected. It is important to mention that this model was tried to be done as simple as possible.

First, the function of the sequential model was called (Sequential()), this is a linear stack of layers and the most common type of model. After this, the Embedding() layer function was called, and it creates a dictionary by mapping the integer indices created during the tokenizing to dense vectors,

and two attributes are given, the number of maximum words index, which was set to 2,000 (of most frequent words), and the dimensionality, which was set to 45, and this was based on the average length of the reviews from the training dataset.

Furthermore, based on related works it was decided to use a Bidirectional Long Short-Term Memory (LSTM). An LSTM is a type of Recurrent Neural Networks (RNN), and it was designed for modeling more accurately temporal sequences and their long-range dependencies. The main advantage of this network is that it is able to remember the sequence of past data i.e. words in our case in order to make a decision on the sentiment of the word. LSTM architecture was deeply explained in Review #1. In regards to the Bidirectional LSTM, it splits into two directions (forward states, for positive time direction and backward states, for negative time direction) the neurons of a regular LSTM, the inputs of the opposite direction states, are not connected to the output. What happens when the two-time directions are being used, is that the information from the past and future of the current time frame can be used (Pal, Ghosh, & Nag, 2018).

Then, the global max-pooling operation in 1D is applied to be used for temporal data, this helps the architectural model, as the total number of parameters is reduced and with this, we try to minimize the overfitting (Lin, Chen, & Yan, 2014). The layers of the global max-pooling are used to reduce the spatial dimensions of a 3D tensor, so based on the input of the max-pooling layer, the global max-pooling output will be the maximum amount in the max-pooling layer.

Then the dropout rate used for the layers is added, and finally, one layer with the sigmoid activation function is implemented.

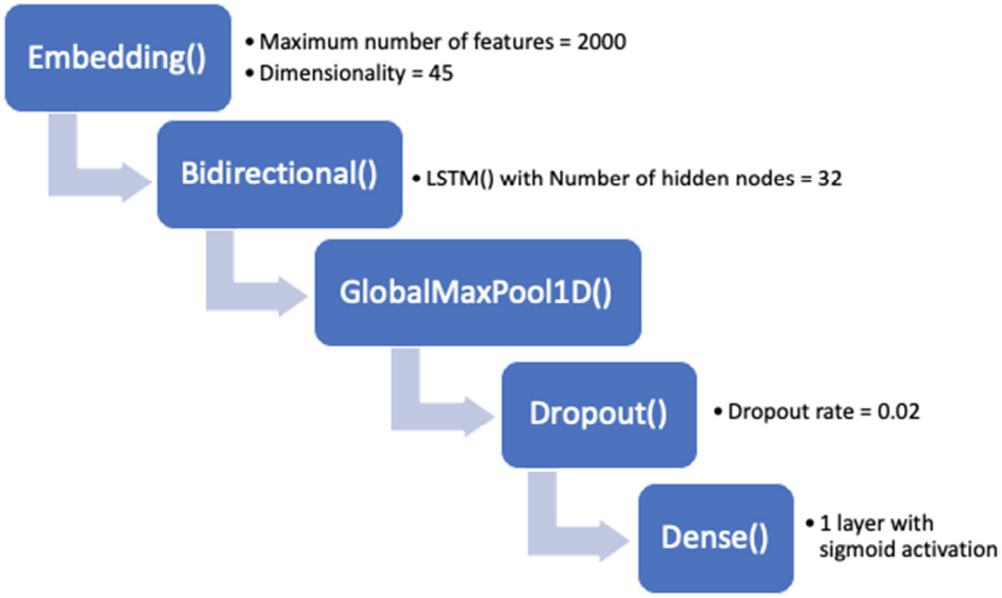


Figure 4. The architectural model used.

3.2 Description of Metrics and Mathematical Formulation

For the tuning of the hyperparameters that were used at the end, a grid search with cross-validation was applied to the training dataset, and to reduce the time of the process, between two to three options to each parameter were selected.

The hyperparameters to be validated were the following:

- The number of hidden units for the LSTM: This is the dimensionality of the output layer, and it refers to the dimension of the hidden state vector or the state output from an RNN cell (TensorFlow, 2020). Three options were used to select this number, 8, 18 and 32 hidden nodes, but these three were placed in an array to check each of them in a "for loop," and the grid search with cross-validation was inside this "for loop."
- Dropout rate: This is a regularization to prevent overfitting in a Neural Network, and it works by randomly dropping out neurons during every epoch (TensorFlow, 2020). The dropout rate should be between 0 and 1, and in this case, two small options were selected, 0.02 and 0.05, these numbers were placed in the grid search with cross-validation.

- Number of Epochs: An epoch is one full training cycle on the given set, once it finishes the first epoch, it starts with the second one. Two options were selected, 3 and 5, and these numbers were placed in the grid search with cross-validation.
- Batch size: Before all the internal parameters of the model are updated, the batch size provides the number of training samples to work through. Two options were selected, 8 and 16, and these numbers were placed in the grid search cross-validation.

Also, the optimizer that was used was Adaptive Moment Estimation (Adam), as it has demonstrated to work well in practice, and it is also favorably compared to other optimization methods (Kingma & Ba, 2014).

In regards to the loss during the compile, the binary cross-entropy was selected, which calculates the cross-entropy loss between the predicted labels and the true ones. And for the metrics, the accuracy was selected.

The cross-validation of the grid search was set to 3-folds, and once the "for loop" containing the grid search finished, the best score obtained, which indicates the mean cross-validated score of the best parameter, was 0.9316, and the best parameters obtained were 32 hidden units, 3 epochs, with a batch size of 8 and dropout rate of 0.02.

The architectural model was created with this tuned hyper parameters, was then compiled and fit with the training dataset and then validated with the validation dataset. Its results are shown in Figure 5, it can be seen that the accuracy of the model kept increasing on every epoch, and its loss kept decreasing on every epoch, although the difference between the second and third epoch was minimal.

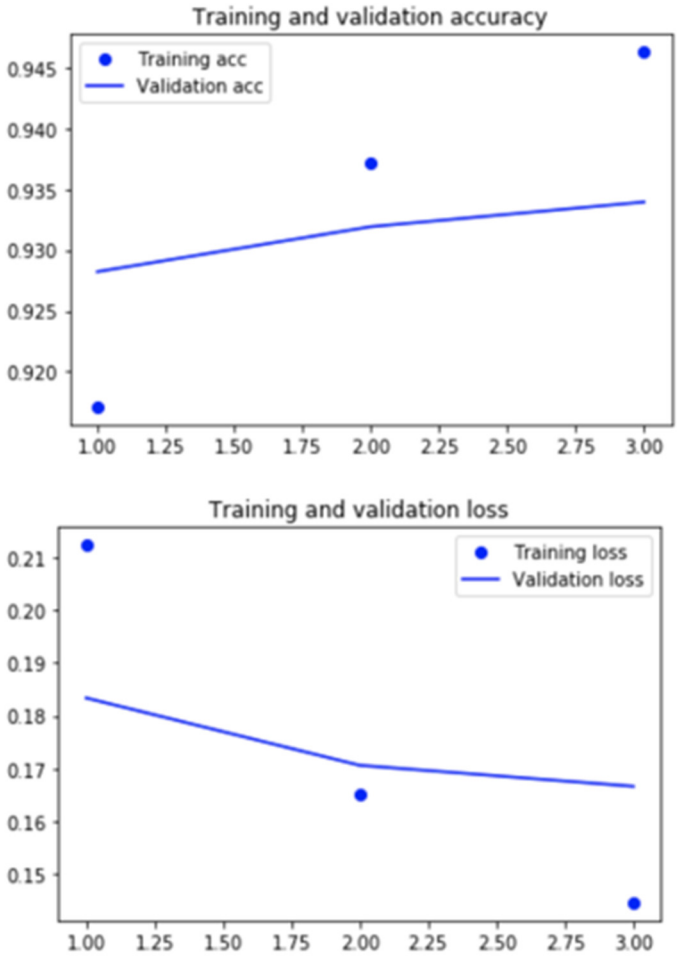


Figure 5. Training and validation losses and accuracies.

The same fitted model was then used to predict the test dataset and the results were compared with the labels, obtaining a 94% accuracy, which is pretty good, and as this dataset is balanced, we can rely on this result. Also, by observing the confusion matrix on Figure 6, it can be seen that all the numbers look similar, and comparing between the true labels and the predicted labels, for both of the clases, only the 6% of the samples for each of the classes were predicted incorrectly.

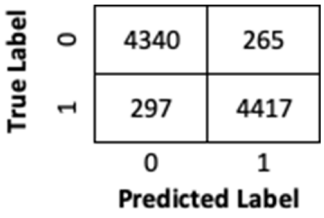


Figure 6. Confusion matrix.

4 Possible difficulties

Popular problems of sentiment analysis classification are sarcasm, negations, word ambiguity, and multipolarity. In sarcastic text, people express their negative sentiments using positive words. This fact allows sarcasm to easily cheat sentiment analysis models unless they're specifically designed to take its possibility into account. Moreover, in linguistics, negation is a way of reversing the polarity of words, phrases, and even sentences. Researchers use different linguistic rules to identify whether negation is occurring, but it's also important to determine the range of the words that are affected by negation words. Word ambiguity is another pitfall you'll face working on a sentiment analysis problem. The problem of word ambiguity is the impossibility to define polarity in advance because the polarity for some words is strongly dependent on the sentence context. Finally, sometimes, a given sentence or document (or whatever unit of text we would like to analyze) will exhibit multi-polarity. In these cases, having only the total result of the analysis can be misleading, very much like how an average can sometimes hide valuable information about all the numbers that went into it. Knowing about each of these challenges will help us avoid possible problems and can significantly increase sentiment analysis accuracy in a classification model. There are different solutions to overcome these problems. We can change the structure of the network and use a more complex and deep network to learn sarcasm, negations, word ambiguity, and multi-polarity. Word embedding layer in network architecture can capture context of a word in a document, semantic and syntactic similarity and relation with other words but it is not enough for above sentiment analysis problems. In this case, we may face resource limitation especially in the case of larger datasets. Another solution is improving the preprocessing step to choose a better primary vocabulary. In fact, one of the major challenges in this problem is choosing the right text features. Our preprocessing steps are very simple. We extract 2000 most repetitive words using the Tokenizer() function in Keras, without considering any above problems. Moreover, selecting "2000" as the parameter of this function is something that needs more evaluation. To improve this set we can add a pre-processing step that considers other effective criteria for selecting these vocabularies alongside the "frequency of words". For example, in our implementation, we remove the word "book" to improve the final result because this word appears equally in both positive and negative sets. But this initial vocabulary set can still be improved with some statistical and linguistic analysis.

5 References

Amazon.com, Inc. (2020). Help & Customer Service. Retrieved from Amazon: https://www.amazon.com/gp/help/customer/display.html?nodeId=201889150

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*.

Lin, M., Chen, Q., & Yan, S. (2014). Network in Network.

Ni, J., Li, J., & McAuley, J. (2019). *Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained* Aspects. https://doi.org/10.18653/v1/d19-1018

Pal, S., Ghosh, S., & Nag, A. (2018). Sentiment Analysis in the Light of LSTM Recurrent Neural Networks. International Conference on Information Technology and Applied Mathematics.

TensorFlow. (2020, 3 7). *tf.keras.layers.Dropout*. Retrieved from TensorFlow: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout

TensorFlow. (2020, 3 7). *tf.keras.layers.LSTM*. Retrieved from TensorFlow: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM