Performance Assessment D213 – Advanced Data Analytics  
Task II

Doug Haunsperger

College of IT, Western Governors University

May 10, 2024

# Part I. Rese**arch Questi**on

## A1. Question Proposal

I propose to research the question, “Can we successfully detect review sentiment on a dataset using a sentiment analysis model trained on a different domain?”

## A2. Goal

The goal of this analysis would be to develop a neural network model that is trained on reviews of items in certain domains (restaurants, movies) and then show that it can be used to detect sentiment in reviews of items in a novel domain (in this case, consumer goods).

## A3. Type of Neural Network

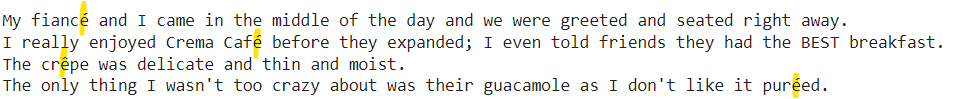
To research this question, I have chosen to use a Recurrent Neural Network (RNN), and more specifically, a type of RNN called Long Short Term Memory (LSTM). RNNs are “neural networks whose connections between neurons form a directed cycle” (Zhang et al., 2018). A model of this type “can use its internal ‘memory’ to process a sequence of inputs,” such as a review – which can be expressed as a sequence of words or word-like tokens (Zhang et al., 2018). My model includes a bidirectional LSTM layer. According to Zhang et al, LSTM was developed to deal with the tendency of RNNs to have vanishing or exploding gradients, while making an RNN bidirectional helps to identify dependencies both backward and forward in the text sequence to determine meaning (2018).

# Part II. Data Preparation

## B1. Data Exploration

I began by searching each review corpus for non-standard characters. Specifically, I searched for any characters that are encoded in Unicode by more than one byte (that is, they are not one of the basic 255 ASCII characters or control codes. Code and results can be seen in section B1 of the attached Jupyter notebook ‘D213\_PA2\_final.ipynb’. The output of this search on the Yelp dataset, with the non-ASCII characters highlighted, is shown in Figure 1.

Figure



I decided to remove all characters except for ASCII alphabetical and spaces. This also had the effect of removing punctuation and punctuation-based emoji such as “:)”.

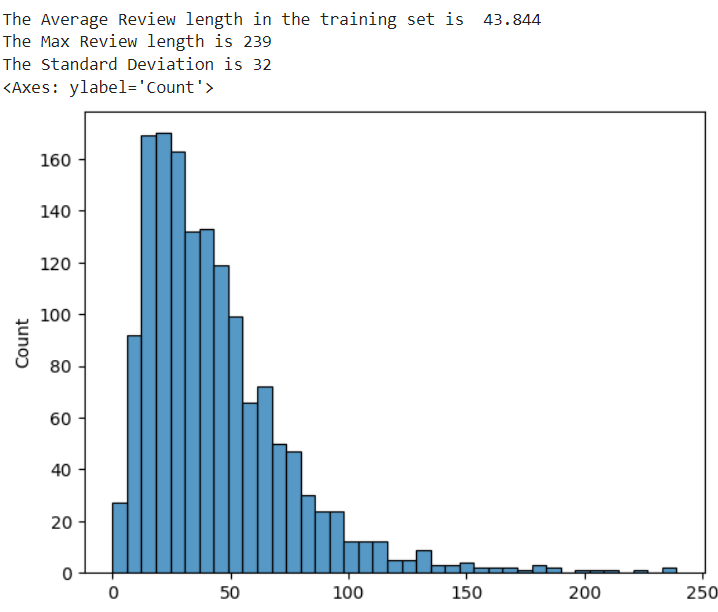
I next performed the rest of the pre-processing steps, including removing stopwords and lemmatization, prior to doing tokenization and determining vocabulary size. These steps are described in detail in section B5.

I concatenated the Yelp and IMDB data sets and split them into training and test sets. With a training set encompassing 1500 reviews, the vocabulary size was 3207 unique words. Including the ‘<unk>’ (‘unknown’) token, there were a total of 3208 elements in the vocabulary list.

The Google TensorFlow development team propose a heuristic rule of the fourth root of the vocabulary size as the word embedding dimension (2017). For a vocabulary size of 3208, this leads to an embedding dimension of 8. Patel and Bhattacharyya found that model performance tends to increase quickly as embedding dimension increases up to a point at which it levels off (2017). I tested doubling the embedding dimension to 16, but did not see significant accuracy improvement. The increased number of parameters to train had a noticeable impact on time to train, so I left the embedding dimension at 8.

For a maximum sequence length, I examined the distribution of review lengths in terms of the tokenized sequences. This is shown in Figure 2.

Figure



The distribution has a long right tail. The majority of reviews have fewer than 50 tokens, and nearly all have fewer than 100. I chose to cut off the sequence length at the mean plus 2 standard deviations, which equated to 107 tokens.

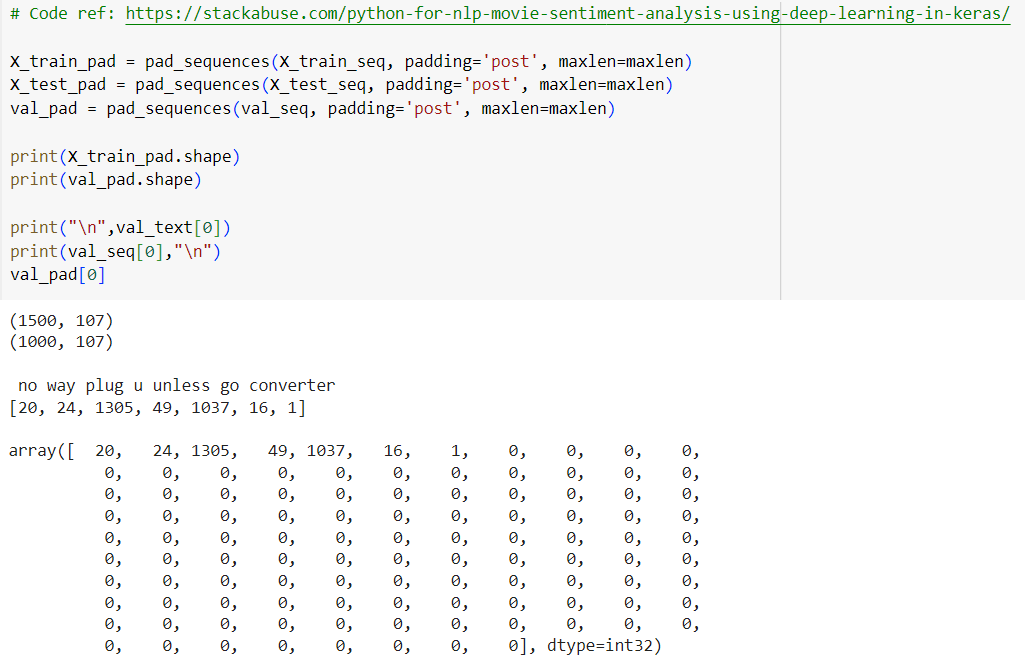
## B2. Tokenization

Tokenization is the process of breaking up a string of text into smaller chunks (words, n-grams, or word fragments, depending on application). It “turns an unstructured string (text document) into a numerical data structure suitable for machine learning” (Menzli, 2023). The process builds a dictionary of tokens and associated numeric values then translates the input text into arrays of numbers.

## B3. Padding

Since the neural network model expects fixed-length sequences (because the first layer of the model must have as many neurons/nodes as the sequence has tokens), sequences longer than the chosen maximum length are truncated, and sequences shorter than the maximum length are padded with zeroes. See Figure 3 for an example tokenized and padded sequence from the validation (Amazon) set. I have chosen to place the padding at the end of the sequence, but since the first hidden layer of the model is a bidirectional LSTM (which feeds the sequence in front-to-back and back-to-front), there should be little difference. I verified this empirically and saw little change in model accuracy between pre- and post-padding. The TensorFlow Guide recommends using post-padding “in order to be able to use the CuDNN implementation of the layers” (Zhu & Chollet, 2023).

Figure



## B4. Categories of Sentiment

The provided data sets categorize sentiment simply as ‘positive’ (1) or ‘negative’ (0). Since these are the labels the model is trained to recognize, I will use these same categories. There is a single output node in my network, with a sigmoid activation function. Softmax is typically chosen for classification tasks (Wu, 2021). For a binary classification problem, Wu shows that sigmoid is mathematically equivalent to softmax, but only requires one output node rather than two, with a concomitant reduction in needed computational resources (2021).

## B5. Preprocessing Steps

These are the steps I followed to prepare the datasets for analysis:

1. Import the list of stopwords from NLTK
2. Change the “’t” contraction to “not” in the dataset
3. Remove “no” / “not” / “nor” from the stopwords set, based on recommendations by Aleti (2020).
4. Split the review sentences into words using NLTK word\_tokenize.
5. Remove any words that appear in the revised stopwords list.
6. Lemmatize the remaining words using NLTK WordNetLemmatizer.
7. Rejoin the words into text sequences (they will be re-split using the Keras Tokenizer later)
8. Concatenate Yelp & IMDB datasets and do a random 75-25 train-test split. Per Brownlee, common training split percentages range from 50 to 80% (*Train-Test Split,* 2020).
9. Set aside the Amazon dataset as the validation data.

The Yelp, IMDB, and Amazon datasets each have 1000 reviews. Therefore, the train set has 1500 rows, the test set has 500, and the validation set is the processed Amazon set and remains with 1000 rows.

## B6. Prepared Data

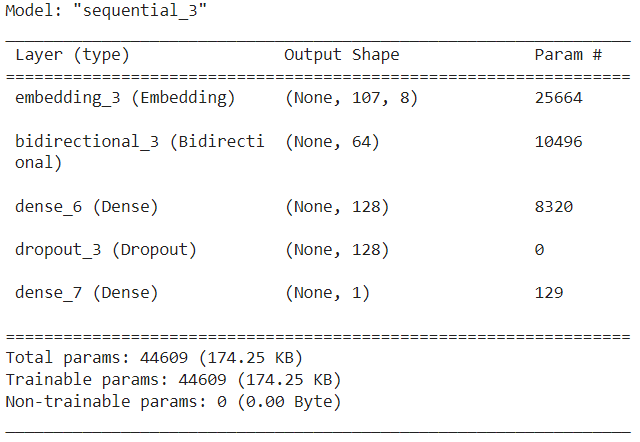
The tokenized data sets have been attached as ‘traindata.csv’, ‘testdata.csv’, and ‘valdata.csv’. The numeric transformed data sets (post-sequencing and padding) have been attached as ‘traindata\_pad.csv’, ‘testdata\_pad.csv’, and ‘valdata\_pad.csv’.

# Part III. Network Architecture

## C1. Model Summary

Figure 4 is a screenshot of the model summary output.

Figure



## C2. Network Architecture Description

As seen in Figure 4, my final model includes the following:

1. Embedding layer – sequence length of 107, embedding dimension of 8. 25,664 trainable parameters (weights and biases).
2. A bidirectional LSTM layer with 32 nodes in each direction. 10,496 trainable parameters.
3. A densely-connected layer of 128 nodes. 8,320 trainable parameters.
4. A dropout layer to drop 50% of the dense layer outputs to prevent overfitting.
5. A final densely-connected output layer with 1 node. 129 trainable parameters.

## C3. Hyperparameters

### Activation Functions

The LSTM layers use the default activations functions (hyperbolic tangent for the cell state activation and sigmoid for the node output). Per the Keras API (<https://keras.io/api/layers/recurrent_layers/lstm/>), this is required to enable the GPU-based cuDNN implementation.

The hidden Dense layers use the rectified linear activation function. Brownlee gives several advantages of the ‘relu’ activation for non-recurrent layers, such as computational simplicity and better results in deep networks versus the traditional sigmoid or ‘tanh’ functions (*ReLU,* 2020).

The output Dense node uses sigmoid, as discussed in section B4.

### Number of Nodes per Layer & Optimizer

I experimented with various combinations of number and types of layers after the embedding layer with varying numbers of nodes, testing for accuracy on the test dataset. Using the Adam optimizer, all models I tried performed acceptably well, from a low of 0.766 on a model with 2 128-node Dense layers with a 0.5 Dropout layer between each, to a high of 0.820 on my chosen model.

I also briefly experimented with using the Stochastic Gradient Descent optimizer, but at least at the default learning rate, the accuracy results were far below any of the models I trained with Adam.

### Loss Function

Since the end goal is classifying reviews as a binary choice (positive or negative), I chose to use the binary cross-entropy loss function. This is the standard function used in deep learning to do binary classification. The function “quantifies the dissimilarity between probability distributions, aiding model training by penalizing inaccurate predictions” (Saxena, 2023).

### Stopping Criteria

I used an early stopping monitor with a patience setting of 2, i.e. if the monitored loss of the test dataset fails to decrease for 2 epochs, the training process halts.

### Evaluation Metric

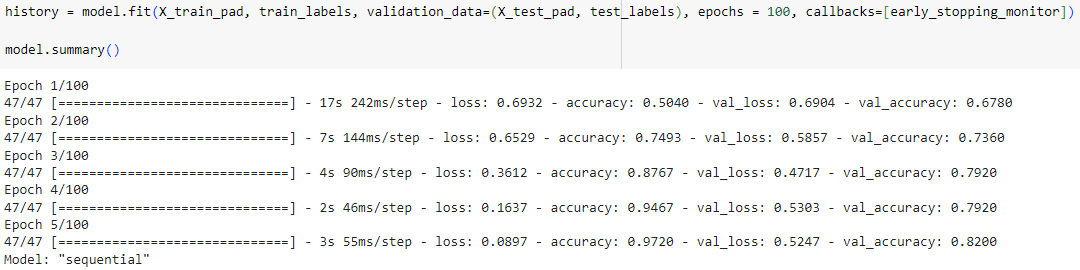
While building the model, training the model, and choosing hyperparameters, I evaluated measured accuracy on the test dataset. This gives an easily understandable figure of merit for the model performance.

# Part IV. Model Evaluation

## D1. Stopping Criteria

The reason to use stopping criteria is to judge when the model has been trained “well enough” to minimize loss on the test dataset and not overfit on the training dataset. Figure 5 is a screenshot of the training process, showing that the training halted after just 5 epochs due to the lack of improvement in ‘val\_loss’ (i.e. the calculated loss on the test set) since epoch 3.

Figure



## D2. Model Fitness

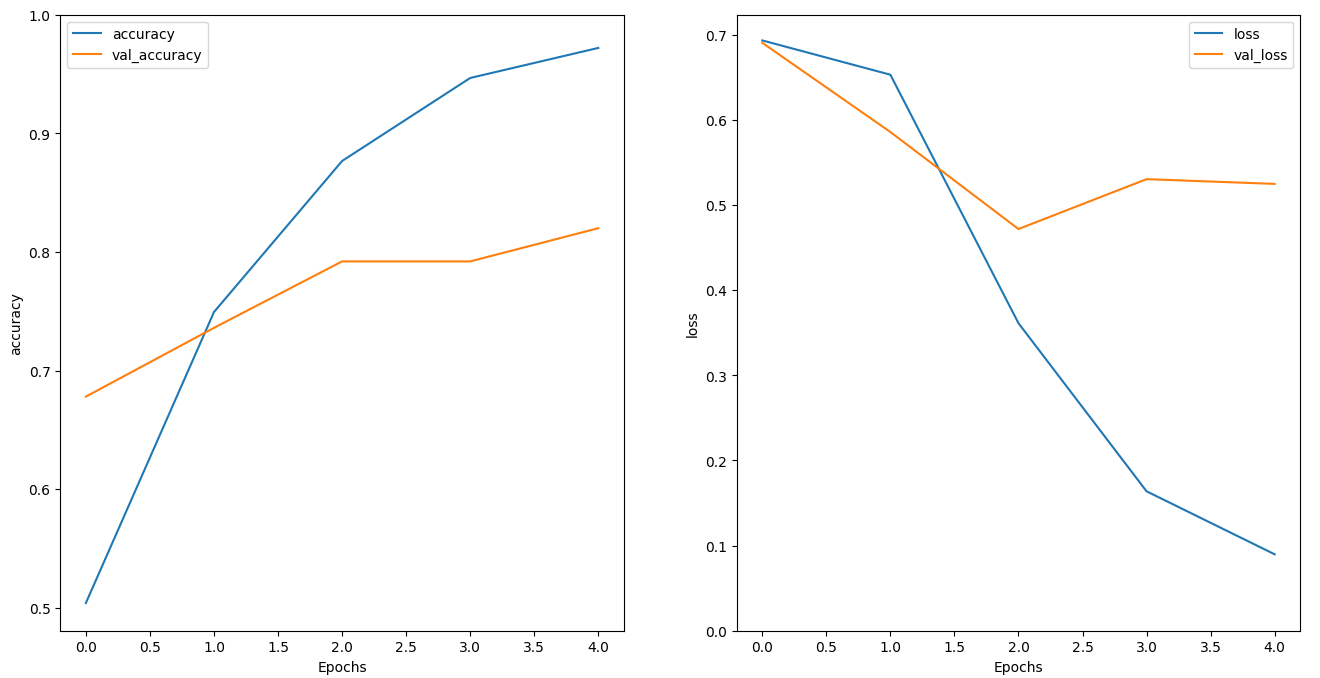
As discussed earlier, I use the binary cross entropy loss function to train and evaluate the model. A perfect fit to the data would result in a loss calculation of 0. The model quickly reduces the loss on its training data, reaching a value under 0.1 in just 5 epochs. Brownlee gives five methods to regularize a neural network to help prevent overfitting - activity regularization, weight constraint, dropout, noise, and early stopping (*Overfitting,* 2019). To prevent the model from overfitting on training data, I have chosen to use two of these methods – dropout and early stopping. Early stopping was discussed in section D1. Dropout “approximates training a large number of neural networks with different architectures in parallel” (Brownlee, *Dropout*, 2019). Some fraction of densely-connected node outputs are randomly ignored. This forces “nodes within a layer to probabilistically take on more or less responsibility for the inputs” (Brownlee, *Dropout*, 2019).

The model after its final training epoch had a loss of 0.0897 on the training data but 0.5247 on the test data.

## D3. Training Process Visualizations

Figure 6 shows graphs of accuracy and loss on the training set (blue line) and test set (orange line).

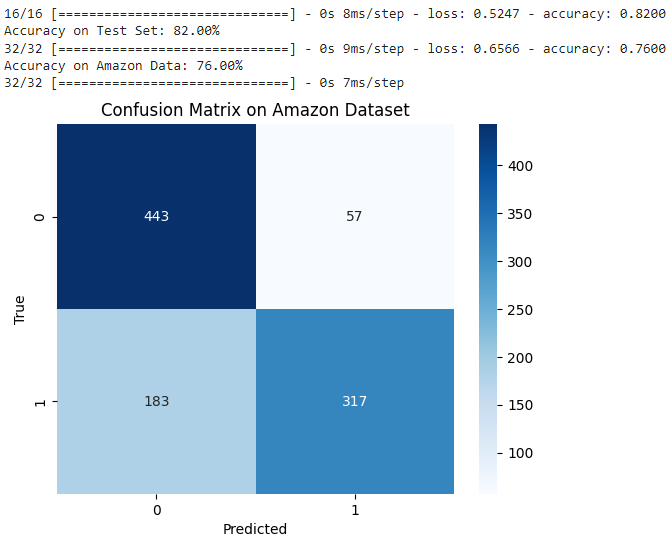
Figure



## D4. Predictive Accuracy

As previously mentioned, the trained model has an 82% accuracy on the test set. Evaluated against the never-before-seen Amazon data set, the model has a 76% predictive accuracy. Figure 7 shows a confusion matrix graphically illustrating the performance against the Amazon dataset. The model performs very well on negative reviews, correctly categorizing 443 out of 500. On positive reviews, it did less well, correctly identifying only 317 out of 500.

Figure



# Part V. Summary & Recommendations

## E. Code

The model is saved into the keras format with a single line of code:

model.save('sent\_analysis.keras')

The ‘sent\_analysis.keras’ model is attached to the submission. The full code for the project is attached as a Jupyter notebook.

## F. Network Functionality

The network starts with an embedding layer that places the 107-element vectors created by the tokenization of the input data into an 8-dimensional space. This feeds into a bidirectional LSTM layer, and from there to a densely-connected layer, the functionality of which were discussed in section A3. The final output dense layer (single node) uses its sigmoid activation to produce a single binary output value – positive or negative.

## G. Recommendation

This model seems to be reasonably useful at evaluating the sentiment of reviews, particularly negative ones, even in a domain that it was not originally trained on. It may be possible to increase accuracy further with additional training data.

# Part VI. Reporting

## H. Jupyter Notebook Report

## The PDF output of my Jupyter notebook is attached as ‘D213\_PA2\_DSH.pdf’. The Jupyter notebook itself is attached as ‘D213\_PA2\_final.ipynb’.

## I. Third-party Code Sources

Sources are cited in comments within the code, in the places where the code is used.

## J. References

Aleti, S. (July 25, 2020). *Don't blindly remove STOPWORDS for a Sentiment Analysis Model.* DEV Community. <https://dev.to/sunilaleti/don-t-blindly-remove-stopwords-in-sentiment-analysis-3nok>

Brownlee, J. (August 6, 2019). Machine Learning Mastery. *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks*. <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/>

Brownlee, J. (August 6, 2019). Machine Learning Mastery. *How to Avoid Overfitting in Deep Learning Neural Networks*. <https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/>

Brownlee, J. (August 20, 2020). Machine Learning Mastery. *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/>

Brownlee, J. (August 26, 2020). Machine Learning Mastery. *Train-Test Split for Evaluating Machine Learning Algorithms*. <https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/>

Menzli, A. (August 11, 2023). MLOps Blog. *Tokenization in NLP: Types, Challenges, Examples, Tools*. <https://neptune.ai/blog/tokenization-in-nlp>

Patel, K. and Bhattacharyya, P. (November 2017). *Towards Lower Bounds on Number of Dimensions for Word Embeddings*. Proceedings of the 8th International Joint Conference on Natural Language Processing. <https://aclanthology.org/I17-2006.pdf>

Saxena, S. (September 13, 2023). Analytics Vidhya*. Binary Cross Entropy/Log Loss for Binary Classification*. <https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/>

TensorFlow Team. (November 20, 2017). Google Developer Blog. *Introducing TensorFlow Feature Columns*. <https://developers.googleblog.com/en/introducing-tensorflow-feature-columns/>

Wu, Z. (June 7, 2021). *Sigmoid or Softmax for Binary Classification.* https://ecwuuuuu.com/post/sigmoid-softmax-binary-class/

Zhang, L., Wang, S. & Liu, B. (January 2018). *Deep Learning for Sentiment Analysis: A Survey*. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. <https://arxiv.org/ftp/arxiv/papers/1801/1801.07883.pdf>

Zhu, S. and Chollet, F. (July 24, 2023). TensorFlow Core Guide. *Understanding masking & padding.* <https://www.tensorflow.org/guide/keras/understanding_masking_and_padding#padding_sequence_data>