**Part I:  Research Question**

*Research Question:*

Can we use sentiment analysis to predict whether the writer of a review feels positive or negative towards the subject of the review?

*Goals:*

The objectives of this data analysis are to analyze a set of reviews from multiple sources (IMDB movie reviews, Amazon product reviews, and Yelp reviews), and use them to build a model that can classify the sentiments of future reviews into a “positive” or “negative” category. This can help us quickly understand customers’ emotions towards certain products, movies, or business.

*Methods:*

We can use Python computing language to perform text classification tasks and produce sentiment analysis predictions. Specifically, we will use Pandas to read and manipulate the text data. We will also utilize Python’s Natural Language Toolkit (NLTK) to transform and clean the text, as well as the TensorFlow and Keras libraries to tokenize and further transform text. Most importantly, TensorFlow/Keras will also be used to build our neural network that will predict the sentiment analysis of reviews. We will train a Recurrent Neural Network (RNN) to create these predictions.

**Part II:  Data Preparation**

Tokenizing the words in our data is a vital preprocessing step that helps prepare this text for analysis. Tokenization is a “method to segregate a particular text into small chunks or tokens” (*Sharma et al., 2021*). These tokens are necessary to transform our data into vectors and allow text data to exist in numerical form, and thus understood by machines. Tokenization can occur at a sentence, character, or word level, and can be done using a variety of different Python packages and methods. In my analysis, I used TensorFlow’s Keras package to tokenize the text.

We also have a need to “pad” our text data. According to Tensorflow’s website, padding “comes from the need to encode sequence data into contiguous batches: in order to make all sequences in a batch fit a given standard length, it is necessary to pad or truncate some sequences” (*Masking and Padding with Keras*). In other words, we will need to add padding to the reviews that are less than our maximum sequence length of 42 words. In this model, the padding occurs at the end of the review, referred to as “post”-padding, which TensorFlow recommends when working with RNN layers. Below is a screenshot of what one of our padded sequences looks like. Note, the zeroes are our padding, while the positive integers represent actual words in this review.

array([ 2, 113, 392, 34, 2, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0], dtype=int32)

There are two categories of sentiment in this dataset: positive, which is represented as a 1, and negative, represented as a 0. Because this is a binary classification problem with a range of probabilities between 0 and 1, the final layer of our network will use a sigmoid activation function (*Janakiev*).

One initial step of any data analysis is to explore the dataset to find anomalies. After combining these three datasets, I found the vocabulary size of all reviews was approximately 5270 words, the longest review was 73 words, and the average review was approximately 12 words. I used regex as well as character tokenization (in the Keras library) to identify non-English characters in the reviews, since the quality of the text will be of great significance. While there were no emojis in the reviews, I was able to find and identify seven instances of “smiley faces”. There were also a few encoding errors and non-English letters in the text, such as *é*, *å*, and *ê*.

Because I wanted to create a set of word tokens that would offer the most utility to our model, I opted to remove most punctuation marks and English *stopwords* (which are established by NLTK), such as “what”, “they”, and “will”. Stopwords typically “have little lexical content, and their presence in a text fails to distinguish it from other texts” (*Bird et al., 2009*). I also decided to use *stemming* to prepare the data, which refers to a “natural language processing technique that lowers inflection in words to their root forms, hence aiding in the preprocessing of text, words, and documents for text normalization” (*Sharma, 2021*). The removal of stopwords and text stemming were aided by the nltk.corpus and nltk.stem packages (specifically, the stopwords and PorterStemmer modules).

After preparing and preprocessing the text data, the vocabulary size of the reviews was approximately 4251 words, the longest review was 42 words, and the average review was approximately 7 words. The maximum sequence length was accordingly set to 42, because this is the number of words in the longest review, which would allow us to not truncate any of our reviews. According to (*Introducing TensorFlow Feature Columns*, 2017), a general rule of thumb for word embedding length should be the 4th root of the number of categories (or vocabulary size). For our dataset, this calculated to about 8.

The final step to prepare the data before building our model was to split the data into training and test sets. This allows us to train our model on one set of reviews, and then be tested on a separate set, which in turn gives us a tool to measure our model’s fit and accuracy. The training set was 80% of the data (corresponding to 2400 reviews), while the testing data was 20% (600 reviews).

**Part III:  Network Architecture**

The summary of our model is below:

Model: "sequential\_100"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding\_69 (Embedding) (None, 42, 8) 34016

flatten\_42 (Flatten) (None, 336) 0

dense\_193 (Dense) (None, 8) 2696

dense\_194 (Dense) (None, 1) 9

=================================================================

Total params: 36,721

Trainable params: 36,721

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

There are four layers in this model: an embedding layer, a reshaping (flatten) layer. And two dense layers. This set of layers holds a total of 36, 721 parameters. One dense layer is “hidden” and has 8 nodes and uses a rectified linear unit (ReLU) activation function, and the other dense layer is our “outer” layer, with one node, and uses a sigmoid function. I chose these activation functions because of industry consensus: according to data scientist Nikolai Janakiev, it is generally common to use a ReLU function for hidden layers and a sigmoid function for the output layer in binary classification problems (*Janakiev*), as our problem is. The relatively small number of nodes per layer is functional and efficient for what is a correspondingly small dataset.

We’re using the binary\_crossentropy loss function because it is better for dealing with probabilities, and this is a binary classification problem in which the model outputs a probability (*Text Classification with Movie Reviews*). The optimizer I chose was the Adam optimizer, which is simple to implement and low on memory/computational requirements. This optimizer produced the overall best results, and is said to be the general best choice (*Giordano, 2020*).

I implemented Keras’ EarlyStopping class as stopping criteria. Specifically, I monitored the validation loss (val\_loss) to stop at its minimum point. I learned from the first set of accuracy and loss graphs that both training and validation accuracy values plateau early and stay relatively stable, so there is no sensible accuracy value to use as stopping criteria. Training loss comes down fast at first and then slowly evens out, while validation loss starts by decreasing with more epochs, hits a valley, and then comes back up. Since we want to minimize loss, we want to pinpoint the number of epochs where that validation loss hits its minimum value. The metric that I will be using the evaluate the model is validation accuracy. This will tell us exactly how often the model correctly predicted the sentiment on our validation/test set of data, which should give us a good idea of how well it will perform when -predicted the sentiment of future, external reviews.

**Part IV:  Model Evaluation**

Stopping criteria, such as early stopping, is used to prevent overfitting a model. Early stopping can detect when metrics such as loss stop improvement and cut off the model from continuing to train on that number of epochs. In contract, setting a predefined number of epochs will train the model on that long, regardless of whether the model is gaining from it. That sort of “overtraining” causes overfitting. The final epoch that was trained on using early stopping is below:

Epoch 2/20

75/75 [==============================] - 0s 2ms/step - loss: 0.0082 - accuracy: 0.9987 - val\_loss: 0.7546 - val\_accuracy: 0.7717

A line graph of the loss (both training and validation) and accuracy (training and validation) for our initial model (without stopping criteria) is below.

Chart, line chart

Description automatically generated

After implementing the stopping criteria which forces the model to stop training after six epochs, the model appears to be properly fit and moderately accurate. According to Tensorflow, if the validation loss or accuracy is moving in the wrong direction (increasing loss or decreasing accuracy), the model is overfitting. If the validation metric begins to stagnate while the training metric continues to improve, the model is close to overfitting (*Overfit and Underfit*). In the graph below, we can see that the validation loss is indeed beginning to stagnate while the training loss is continuing to decrease, thus we are close to overfitting, but still moving in the right direction. The early stopping criteria addresses the overfitting issue, as we can see the model was clearly overfit with 20 epochs (by looking at the above line graph). Our final model has a validation loss of 0.45 and accuracy of almost 0.80. This sort of accuracy is not ideal, but it’s also not bad. It means we can accurately predict the sentiment of a review about 80% of the time.

Chart, line chart

Description automatically generated

**Part V:  Summary and Recommendations**

Our goal was to build a neural network capable of predicting the sentiment of reviews, and we achieved that goal. We preprocessed and prepared our text data into a workable format, and then transformed into vectors that could be understood by our Keras model’s initial embedding layer. That data was flattened into a dimension that was filtered through one dense layer and spit out to a final dense outer layer that gives us a prediction of whether a review is positive or negative. It is not a perfect network, but certainly is a tool that we could implement. However, I would recommend gathering more reviews to build a more accurate predictor of sentiment. I would personally feel more comfortable if we had a model that accurately predicted sentiment at least 90% of the time. Gathering more data in the form of reviews will give us a larger network to train on, and likely a more accurate model.

References

Bird, S., Klein, E., & Loper, E. (2009). 2. Accessing Text Corpora and Lexical Resources. In *Natural language processing with python*. essay, O'Reilly. Retrieved from https://www.nltk.org/book/ch02.html.

Giordano, D. (2020, July 26). *7 tips to choose the best optimizer*. Medium. Retrieved January 20, 2022, from https://towardsdatascience.com/7-tips-to-choose-the-best-optimizer-47bb9c1219e

*Introducing tensorflow feature columns*. Google Developers. (2017, November 20). Retrieved January 20, 2022, from https://developers.googleblog.com/2017/11/introducing-tensorflow-feature-columns.html

Janakiev, N. (n.d.). *Practical text classification with python and keras*. Real Python. Retrieved January 20, 2022, from https://realpython.com/python-keras-text-classification/

*Masking and padding with Keras:* TensorFlow. (n.d.). Retrieved January 20, 2022, from https://www.tensorflow.org/guide/keras/masking\_and\_padding

*Overfit and underfit*. TensorFlow. (n.d.). Retrieved January 20, 2022, from https://www.tensorflow.org/tutorials/keras/overfit\_and\_underfit

Sharma, P. (2021, January 1). *Keras tokenizer tutorial with examples for Beginners*. MLK - Machine Learning Knowledge. Retrieved January 20, 2022, from https://machinelearningknowledge.ai/keras-tokenizer-tutorial-with-examples-for-fit\_on\_texts-texts\_to\_sequences-texts\_to\_matrix-sequences\_to\_matrix/

Sharma, P. (2021, November 25). *An introduction to stemming in Natural Language Processing*. Analytics Vidhya. Retrieved January 20, 2022, from https://www.analyticsvidhya.com/blog/2021/11/an-introduction-to-stemming-in-natural-language-processing/

*Text Classification with Movie Reviews*. TensorFlow. (n.d.). Retrieved January 20, 2022, from https://www.tensorflow.org/hub/tutorials/tf2\_text\_classification