D213 Task 2

Benjamin McDaniel

Western Governors University

Table of Contents

[Research Question 3](#_Toc97022992)

[Data Preparation 3](#_Toc97022993)

[Feature Engineering & Data Exploration 4](#_Toc97022994)

[Balancing the Data Set 6](#_Toc97022995)

[Vocabulary & Embedding Length 7](#_Toc97022996)

[Tokenization 7](#_Toc97022997)

[Padding Process 9](#_Toc97022998)

[Categories of Sentiment 9](#_Toc97022999)

[Final Data Preparation Steps 10](#_Toc97023000)

[Vectorization and Embedding 11](#_Toc97023001)

[Network Architecture 13](#_Toc97023002)

[Model Evaluation 17](#_Toc97023003)

[Model Evaluation 19](#_Toc97023004)

[References 21](#_Toc97023005)

# Research Question

The research question that will be answered is: Can a convolutional neural network with integrated long short-term memory layer (LSTM) be used to predict the sentiment of verified customer submitted Amazon reviews of pet supplies?

The goals of this analysis include pre-processing Amazon customer reviews provided by (Ni et al., 2019), into cleaned vectorized tokens using the pre-trained Google News vectors negative 300 word2vec model. The pre-trained word2vec model was provided by (Google, 2013). Create the vectorized input suitable for a convolutional neural network (CNN) with integrated LSTM layer for text sentiment classification. The final goal is to evaluate the model on test data to determine accuracy.

A CNN with integrated long short-term memory layer has been identified as the type of neural network that is to be used in this analysis. As described by (Hagiwara, 2021), CNNs have been widely used in tasks that involve computer vision and have recently been applied to natural language processing (NLP) in classification tasks.

# Data Preparation

Exploratory data analysis was conducted on the data set provided by (Ni et al., 2019), as a “small” subset of reviews, specifically the Pet supplies category.

The original data set was reduced to include only reviews on products that were labeled as a “Verified” purchase. This reduced the original data set from 2,098,325 reviews to 1,929,042. The data set was further reduced to three features, and they were renamed to ensure conformity with pythonic conventions. The remaining features were named “reviewer\_id”, “overall”, and “review\_text”. Duplicate entries were identified and then dropped from the data set.

## Feature Engineering & Data Exploration

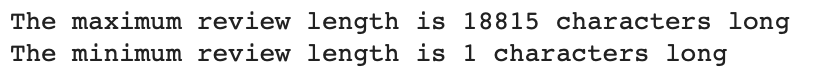
Several new features were engineered to explore the data more deeply. The first of such was the target feature for this analysis named “labels”. This feature is composed of three values based on the values contained in the “overall” feature. The “overall” feature provides the number of stars out of five assigned by a reviewer to a product review. If the “overall” value is below 3, the “labels” feature is -1. If the “overall” value is above 3, the “labels” feature is 1. When the “overall” value is 3, the “labels” feature has a value of 0. In turn label values indicate positive, neutral, or negative reviews as 1,0, and -1 respectively. The distribution of the labels was found to be unbalanced in favor of positive reviews. Below is a visualization of the label distribution. Next to this visualization is the numerical frequency of occurrence of each class, with the proportion of each class below.

Chart, bar chart

Description automatically generated Text

Description automatically generated

The next feature created was “review\_length” and is a value that represents the number of characters in the customer review of the product. The maximum review length was found to be 18,815 characters long and the minimum review length was found to be one character long. The output of this determination is found below.



A visualization of the univariate distribution of review length is below.

Chart

Description automatically generated Text

Description automatically generated

The feature “num\_words” was created as a count of words in each review text. A visualization of the distribution for this feature is below along with the descriptive statistics that were generated.

Chart

Description automatically generated Text

Description automatically generated

The feature “num\_all\_cap\_words” was created to visualize a count of all words found in reviews that were composed of all upper-case letters.

Chart

Description automatically generated Text

Description automatically generated with medium confidence

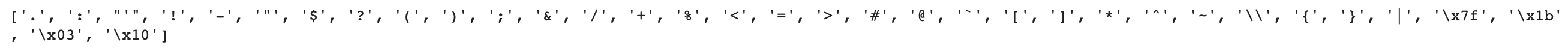
The feature “num\_characters” was developed as a count of characters that include punctuation and special characters that are not digits or letters found in each review text. This feature was created by first creating a feature named “characters” which contains the non-letter or digit characters found in each review. The “num\_characters” feature is a count of those characters.

Chart

Description automatically generated Text

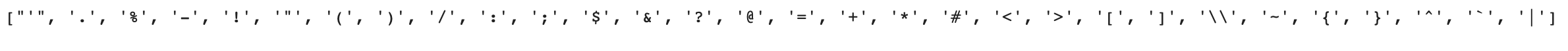
Description automatically generated with low confidence

The characters that were found in the full data set can be seen in the image below. One set of characters of note that were found were web links that needed to be converted into text and tags stripped before further processing.

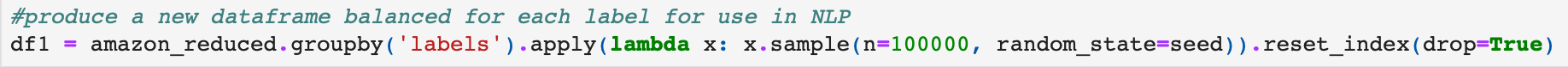


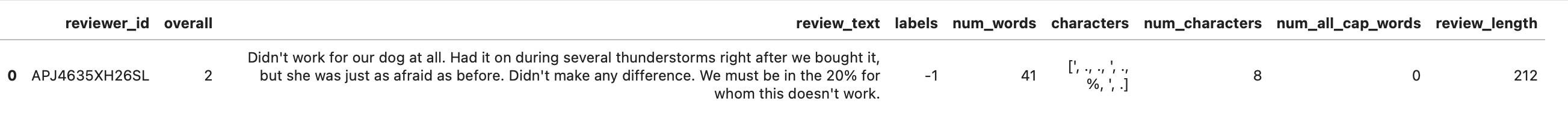
## Balancing the Data Set

The data set was found to be unbalanced with most of the reviews being of the positive label class of 1 as noted during the data exploration. To train a machine learning model more effectively, the data set was subsampled to ensure that the target classes were equal. The resulting data set of 300,000 reviews had the following unique characters within the review texts.



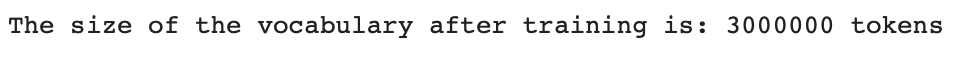
The line of code used to balance the data set follows with the first row of the resulting data frame.





## Vocabulary & Embedding Length

The Vocabulary used was taken from the Google News pre-trained word2vec model and consists of 3,000,000 words. This was confirmed after loading the model and the output is below.



The proposed word embedding length is 113. This was determined by taking the mean number of tokens per review and adding three times the standard deviation of the number of tokens per review. The descriptive statistics were calculated to determine this figure and the output is found below.

Text

Description automatically generated

## Tokenization

The goals of the tokenization process as outlined by (Vajjala et al., 2020), is to break up the text into individual words or sentences. In this analysis, the goal is to break each review down into individual words or tokens. To accomplish this task, the text also needs preliminary processing to retrieve the most informative tokens to be modeled. This includes making all the words in the text lowercase as well as the removal of digits, punctuation, html remnants, and special characters. English stop-words will also be removed as they are not as informative as other less commonly used words as summarized by (Kedia & Rasu, 2020). Furthermore, the resulting words will be stemmed for more efficient application in word representation. Stemming removes the suffixes from different variations of a base word. This allows each form of the base word to be represented in the same way during natural language processing.

A function was written to process and tokenize the review text for this analysis and is found below. The package used to remove html remnants was BeautifulSoup. The stemmer utilized was the SnowballStemmer from nltk.snowball. The word tokenizer utilized was from nltk.tokenize. The stop-words utilized came from nltk.corpus. The Pandas library was also utilized to perform regex functions on a pandas series.

Text

Description automatically generated

This function was executed as a call to create a new feature column within the data-frame. The code generated is found below.

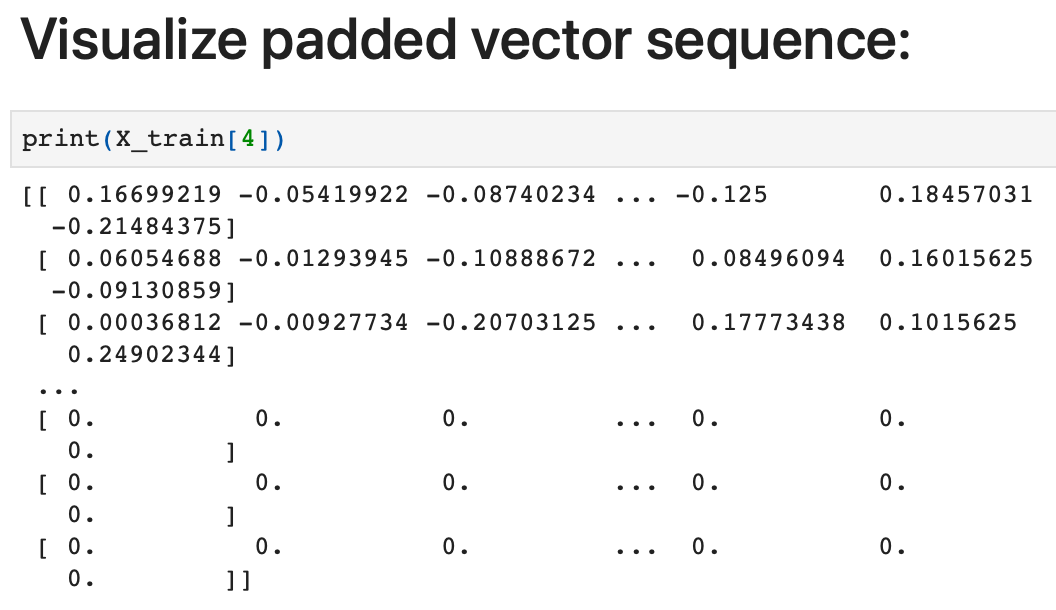
Graphical user interface, application

Description automatically generated

Further exploration of the new feature was conducted, and an additional feature was created named “num\_tokens”. As described in the vocabulary and embedding length section of this paper, this feature was used to determine the most appropriate embedding length of 113.

## Padding Process

The padding process that was used to standardize the length of sequences included adding 0.0 values at the end for each sequence that was shorter than 113. Meaning that the padding occurs post text sequence where necessary. Below is a screenshot of a padded sequence for this analysis.



## Categories of Sentiment

For this analysis there are three categories of sentiment that have been identified and labeled. A customer review is identified as 1 (positive), 0 (neutral), or -1 (negative). These categories are contained in the “labels” feature column of the pandas data-frame. To create the target variable this feature is split into three encoded columns. The code used to accomplish this is below along with a screenshot of the resulting new features.

Text

Description automatically generated with low confidence

Graphical user interface

Description automatically generated with low confidence

Because there are three classes of sentiment in the target variable the final dense layer of the neural network will require three nodes and the use of the “softmax” activation function.

## Final Data Preparation Steps

A pretrained word2vec model was loaded as described above using the following line of code.



Feature and target variables were then assigned to X and y respectively.

Graphical user interface, text, application

Description automatically generated

A function was written and used to split the data into Training, Validation, and Test sets. The function used is below and was influenced by code found in (Géron & Demarest, 2019).

Text

Description automatically generated

The function was executed with the line of code below. The size of the training set is 80% of the data, the validation and test set are both 10% of the size of the data set. The code output is found immediately below the code and returns the shape of the resulting splits.

Graphical user interface, text, application

Description automatically generated

The data set is now prepared for vectorization and use in a neural network. The split data set was saved as comma separated value files for future use and submission with this report. This was executed with the pandas to\_csv() function below.

Text

Description automatically generated

## Vectorization and Embedding

Once prepared, the resulting feature variable sets were converted into a vectorized matrix post padded with zeros where needed using the pre-trained word2vec model. This was accomplished using a custom written function influenced by code found in (Kedia & Rasu, 2020), and (Matteson et al., 2022).

Graphical user interface, text, application, email

Description automatically generated

This function was used for final transformation of the data sets prior to fitting to the neural network. Each set was converted to a Numpy array of float 32 type while executing the above function named make\_vectors().

Graphical user interface, text, application, email

Description automatically generated

# Network Architecture

The type of network used is a Convolutional Neural Network (CNN) with integrated Long Short-term Memory (LSTM) layer. Below is the output of the model summary from TensorFlow.

Table

Description automatically generated

This neural network is constructed of a total of 19 layers. The model architecture can be described as a CNN-LSTM as it incorporates two convolutional layers or “convets” and an integrated LSTM layer. This model fits this description as it features one or more convets stacked with pooling layers and a single LSTM or RNN layer that returns its full output sequence as summarized by (Chollet, 2021).

As the embedding is complete, the sequential model construction begins with the addition of the first one dimensional convolutional input layer. The one-dimensional convolution was selected due to the dimensionality of text data as described in (Kedia & Rasu, 2020).

Variables were assigned to be used in the construction of the layers and are found below.

Text

Description automatically generated

The layers were added in the order below beginning with the first convolutional layer with the assigned input shape 113 x 300. The “relu” activation function was selected for hidden dense layers, the LSTM layer, and the convolution layers. The rectified linear unit function (relu) will ensure that the product of each node will not have a negative value moving forward. Max pooling layers were assigned a pool size of 5 capturing the maximum value of the 5x5 matrix in each pool. The number of nodes assigned to each of the dense hidden layers were 200 for the first four and 100 for the last two. The dense output layer featured 3 nodes, one for each class of the target variable. The six dropout layers feature a 20% dropout rate for each instance which will assist in ensuring that the model will not overfit the training data. The first convolutional layer features 32 filters and a kernel size of 5 to reduce the output of the data to the succeeding layers. The second convolutional layer features 100 filters and a kernel size of 5 to further capture patterns in the data. The LSTM layer features 128 nodes and return\_sequences set to true so that the output maintains its shape for passing along to the next dense layer of the network. The last pooling layer selected was a GlobalMaxPooling1D layer in place of a flattening layer.

The final dense layer or output layer features 3 nodes for the output with the “softmax” activation function. The “softmax” activation was selected as it is a multiclass classification problem. This model features a total of 405,883 total parameters and of those they are all trainable.

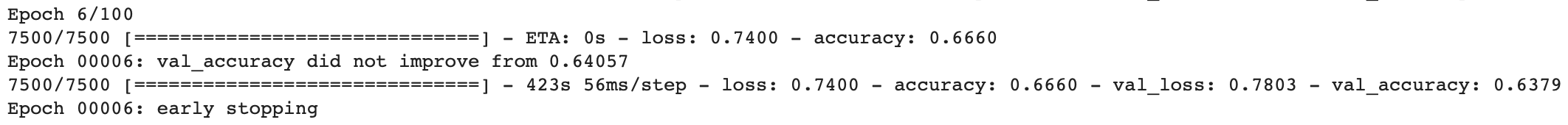
The model was then compiled using categorical cross entropy as the loss function because the problem is a non-binary classification problem. Adaptive moment estimation (ADAM) was selected as the optimizer for this model. As described by (Géron & Demarest, 2019), the ADAM optimizer combines attributes of momentum optimization and root mean square propagation (RMSprop). Specifically, it tracks the exponential decay of past gradients and the exponential decay of average past squared gradients. Accuracy was selected as the metric of interest in the training, validation, and test sets. The stopping criteria selected was validation set accuracy or “val\_accuracy” with a mode of “max” and patience set to three. This will monitor the validation accuracy of each epoch and if the validation accuracy fails to improve after three epochs the training will stop early. I have also set the model to save the best weights collected from the model training that returns the highest validation accuracy. These weights are saved to file so that the model can be loaded again when needed. The fit model has been assigned to the variable named history for ease in retrieving the metrics of choice from each epoch moving forward.

Graphical user interface, text, application

Description automatically generated

# Model Evaluation

The impact of assigning stopping criteria rather than expressly running the training for a specific number of epochs resulted in model training that was les computationally expensive in the long run. Determining the number of epochs prior to training the model would have resulted in either selecting a large arbitrary number of epochs to run to guarantee that the best model weights would have been captured or continuously training the model with increasing number of epochs. These options are less efficient than creating stopping criteria based on the selected metric. Below is a screenshot of the final training epoch that indicates early stopping.

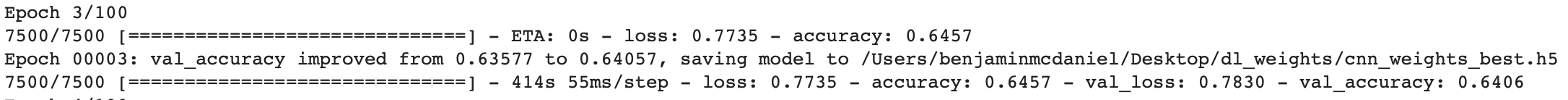


Visualization of the models training process were generated for both loss and accuracy that compare the training values with the validation values. These visualizations are found below and the epochs on the x axis are zero indexed.

Chart, line chart

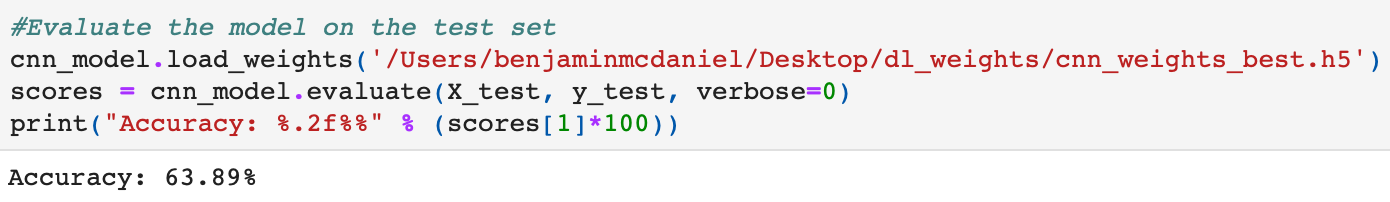
Description automatically generated 

The values represented in visualizations of loss and accuracy indicate that the best fit model weights occurred in the third epoch of training as indicated below.



This output indicates that on the validation set the model predicted classes with 64% accuracy. This is an informative model based on the loss values provided in the output. A model that was uninformative, meaning that its performance is like taking a best guess at labels would have had a loss score of 1.1. As this is not the case the model is informative and better than taking a guess at the class labels. The primary measures taken to address overfitting of the training data were the integration of dropout layers. As described by (Chollet, 2021), a dropout layer will randomly zero out the input nodes of a layer to disrupt incidental correlations in the training data that each layer is exposed to. In this instance each dropout layer randomly zeroed out 20% of the input units of a layer. The same dropout rate was used for each dropout layer so that the model could propagate learning error as it trains on the data.

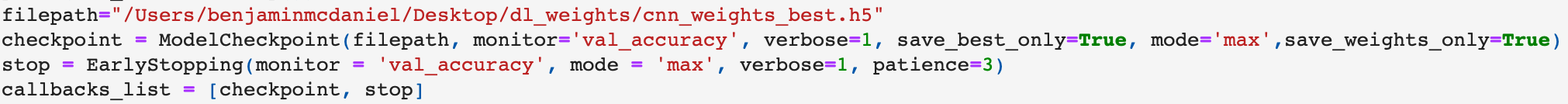
The predictive accuracy of the trained network was assessed on a test set that was hold out from the initial splitting of the data set. Below is the output of this accuracy evaluation and the line of code used to generate it.



The output indicates that the model achieves an accuracy of 63.89% on the test set. This is very close to the accuracy determined on the validation set which was 64%.

# Model Evaluation

The best model weights were saved as the variable checkpoint later assigned to callbacks when the model was fit. The model training and was successfully called back to assess accuracy on the test set. The following lie of code was used to accomplish this.



The model was also expressly saved to file with the following line of code.

Graphical user interface, text, application, email

Description automatically generated

The functionality of the convolutional neural network is related to its architecture in that the shape of the output data from each layer needs to be of the appropriate shape to be accepted by each successive layer in the model. The model was designed with this in mind. Specifically, the inclusion of an LSTM layer required that the return sequences parameter be set to true. Had it not the model would have failed to move data onto the next layer effectively. Furthermore, the output from convolutional layers is dimensionally reduced from the form of its input which needed to be considered when adding layers to the model.

The results of this analysis indicate that a CNN with integrated LSTM can be used to predict label classes of Amazon pet supply review data. The recommendation moving forward is to train a custom word2vec model based on text found in a larger set of Amazon reviews to build a vocabulary to provide more target specific input data. Use this vocabulary to vectorize the input to a CNN with integrated LSTM. The final recommendation is to train the model on a much larger balanced data set of reviews to increase accuracy.

# References

Matteson, D. S., Stoffer, D., Becker, D., Hall, I., & Halder, B. (2022). *Advanced Data Analytics*. Data Camp. Retrieved 2022, from <https://app.datacamp.com/learn/custom-tracks/custom-advanced-data-analytics>

Chollet, F. (2021). *Deep learning with python. second edition*. Manning.

Kedia, A., & Rasu, M. (2020). *Hands-on Python Natural Language Processing: Explore tools and techniques to analyze and process text with a view to building real-world Nlp applications*. Packt.

Vajjala, S., Majumder, B., Gupta, A., & Surana, H. (2020). *Practical natural language processing: A comprehensive guide to building real-world Nlp sysems*. O'Reilly Media.

Hagiwara, M. (2021). *Real-World Natural Language Processing*. Manning Publications.

Ni, J., Li, J., & McAuley, J. (2019). *Justifying recommendations using distantly-labeled reviews and fined-grained aspects*. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. Retrieved February 22, 2022, from https://cseweb.ucsd.edu//~jmcauley/pdfs/emnlp19a.pdf

Géron, A., & Demarest, R. (2019). *Hands-on machine learning with scikit-learn and tensorflow:*

*Concepts, tools, and techniques to build Intelligent Systems*. O'Reilly.

Google (2013, July 29). *Google code archive - long-term storage for google code project hosting.* Google Code Archive. Retrieved February 17, 2022, from https://code.google.com/archive/p/word2vec