


12-2023

REVIEW CLASSIFICATION USING NATURAL LANGUAGE PROCESSING AND DEEP LEARNING

Brian Nazareth

California State University – San Bernardino

Follow this and additional works at: <https://scholarworks.lib.csusb.edu/etd>

 Part of the [Data Science Commons](#), [Numerical Analysis and Scientific Computing Commons](#), and the [Other Computer Sciences Commons](#)

Recommended Citation

Nazareth, Brian, "REVIEW CLASSIFICATION USING NATURAL LANGUAGE PROCESSING AND DEEP LEARNING" (2023). *Electronic Theses, Projects, and Dissertations*. 1821.
<https://scholarworks.lib.csusb.edu/etd/1821>

This Project is brought to you for free and open access by the Office of Graduate Studies at CSUSB ScholarWorks. It has been accepted for inclusion in Electronic Theses, Projects, and Dissertations by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.

REVIEW CLASSIFICATION
USING NATURAL LANGUAGE PROCESSING AND DEEP LEARNING

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Brian Nazareth
December 2023

REVIEW CLASSIFICATION

USING NATURAL LANGUAGE PROCESSING AND DEEP LEARNING

A Project

Presented to the

Faculty of

California State University,

San Bernardino

by

Brian Nazareth

December 2023

Approved by:

Dr. Yan Zhang, Advisor, Computer Science and Engineering

Dr. Jennifer Jin, Committee Member

Dr. Khalil Dajani, Committee Member

© 2023 Brian Nazareth

ABSTRACT

Sentiment Analysis is an ongoing research in the field of Natural Language Processing (NLP). In this project, I will evaluate my testing against an Amazon Reviews Dataset, which contains more than 100 thousand reviews from customers. This project classifies the reviews using three methods – using a sentiment score by comparing the words of the reviews based on every positive and negative word that appears in the text with the Opinion Lexicon dataset, by considering the text's varying sentiment polarity scores with a Python library called TextBlob, and with the help of neural network training. I have created a neural network model that learns from the review stars and then compare the neural network's performance against both the Opinion Lexicon and TextBlob's classification methods. We see that the accuracy of the Opinion Lexicon classification method is 64.38% while the accuracy with TextBlob's classification method is 65.71% and the neural network model achieves an accuracy of 96.46%. The model would help brands for future reviews left by customers by classifying them as positive, negative, or neutral.

ACKNOWLEDGEMENTS

I sincerely extend my thanks to my project committee, Dr. Yan Zhang – Advisor, Dr. Jennifer Jin – Committee Member, and Dr. Khalil Dajani – Committee Member.

I would like to thank my friends and family for their continuous support.

TABLE OF CONTENTS

ABSTRACT.....	iii
ACKNOWLEDGEMENTS.....	iv
LIST OF TABLES.....	vii
LIST OF FIGURES.....	viii
CHAPTER ONE: INTRODUCTION.....	1
CHAPTER TWO: LITERATURE REVIEW.....	3
Existing Methods.....	3
Previous Work.....	4
CHAPTER THREE: PROPOSED SYSTEM.....	9
Proposed System.....	9
System Specifications.....	12
System Design.....	13
CHAPTER FOUR: DATA AND DATA PREPROCESSING.....	17
Data Collection.....	17
Text Cleanup.....	17
CHAPTER FIVE: METHODOLOGIES.....	21
Classification with Opinion Lexicon.....	21
Classification with TextBlob.....	22

Classification with Neural Network	23
CHAPTER SIX: EXPERIMENTAL RESULTS.....	31
Threshold Determination.....	31
Classification Model Evaluation.....	34
Classification Model Comparison.....	47
CHAPTER SEVEN: USER INTERFACES.....	56
CHAPTER EIGHT: CONCLUSION.....	58
APPENDIX A: CODE.....	60
REFERENCES.....	74

LIST OF TABLES

Table 1. Opinion Lexicon Threshold Test Cases.....	32
Table 2. TextBlob Threshold Test Cases.....	33
Table 3. Positive Sentiment Test Case.....	47
Table 4. Positive Sentiment Test Case.....	48
Table 5. Positive Sentiment Test Case.....	49
Table 6. Negative Sentiment Test Case.....	49
Table 7. Negative Sentiment Test Case.....	50
Table 8. Negative Sentiment Test Case.....	51
Table 9. Neutral Sentiment Test Case.....	51
Table 10. Neutral Sentiment Test Case.....	52
Table 11. Neutral Sentiment Test Case.....	53

LIST OF FIGURES

Figure 1. Block Diagram of Model.....	10
Figure 2. Confusion Matrix for Opinion Lexicon Classification.....	36
Figure 3. Confusion Matrix for TextBlob Classification.....	40
Figure 4. Confusion Matrix for Neural Network Classification.....	43
Figure 5. Homepage of the Application.....	54
Figure 6. Sentiment Prediction Output from User Input.....	55
Figure 7. Translated Sentiment Prediction Output from User Input.....	56

CHAPTER ONE

INTRODUCTION

Amazon reviews play a crucial role in the online shopping experience, offering potential buyers valuable insights and feedback from other customers on products they are considering. Reviews can range from simple star ratings to profound and detailed comments about a product's quality, usability, efficiency features, and performance. Amazon's review ecosystem is a colossal compilation of diverse opinions and evaluations, which makes it an indispensable resource for shoppers seeking to make informed purchase decisions. To help ensure authenticity, Amazon's review system includes a "verified purchase" feature that provides additional confidence for buyers. Reviews on Amazon are a powerful tool for consumers, manufacturers, and retailers alike, as they help build customer satisfaction and trust. Reviews manifest as an influential tool for both consumers and manufacturers. They facilitate the cultivation of customer satisfaction, loyalty, and trust, which are paramount in the competitive realm of online retail. These reviews can help provide valuable insights for businesses and this can help them refine their products and services, thereby adopting a customer-centric ecosystem infused with quality and excellence.

Natural Language Processing (NLP) is a field of artificial intelligence that focuses on enables computers to understand, interpret, and generate human language^[1]. NLP combines linguistics, computer science, and artificial intelligence techniques to create algorithms and models to process and analyze

natural language data. These algorithms and models are typically trained on large text datasets such as books, news articles, research papers, and social media posts. Machine learning techniques are used to improve the accuracy and performance of NLP models. The field of NLP is growing rapidly with numerous exciting developments and applications. As digital text keeps on growing, NLP will become increasingly important for making sense of and leveraging this data. By processing and analyzing large volumes of textual data, NLP can provide valuable insights and will be capable of automating tasks that would otherwise require human intervention.

In this project, I will be using Natural Language Processing's key feature – sentiment analysis. With sentiment analysis, the existing reviews can be classified as Positive, Negative, or Neutral for companies to learn what worked with their product and what did not. Sentiment analysis can provide insights for common themes and sentiments in reviews, what products make the customer happy, and what issues could they be facing with a product. Sentiment analysis will help brands and manufacturers to analyze these reviews and look beyond the star ratings.

CHAPTER TWO

LITERATURE REVIEW

Existing Methods

Sentiment analysis has been an active area of research, with various approaches and techniques proposed to analyze and classify sentiments in textual data. Traditional approaches often relied on lexicon-based methods that used sentiment lexicons or dictionaries containing predefined sentiment polarity for words. These lexicons were manually curated or generated using linguistic heuristics and contained positive and negative words with their associated sentiment scores. Commonly used techniques to create lexicons include counting the occurrences of positive and negative words in a text or computing sentiment scores based on the aggregation of word-level sentiments.

Machine learning is a widely used approach in sentiment analysis. Algorithms are trained on labeled datasets to classify sentiments. Techniques such as Naive Bayes, Support Vector Machines (SVM), and Decision Trees have been used, with features derived from the text, such as syntactic patterns, or linguistic features. Feature engineering plays a crucial role in extracting relevant information from the text, while model training and optimization focuses on achieving high accuracy and generalization.

More recently, deep learning models have gained popularity in sentiment analysis. Recurrent Neural Networks (RNNs) like Simple RNN, Gated Recurrent United Neural Network (GRUNN), and Long Short-Term Memory (LSTM)

networks have been successfully used to capture sequential dependencies and contextual information in texts.

Previous Work

In the past few years, a vast number of projects have been done in the field of sentiment analysis.

- Pak and Paroubek proposed techniques to classify twitter reviews as positive, negative, or neutral. They used twitter API to collect tweets and analyze sentiments using Naive Based algorithms.^[2]
- Kiritchenko et al. wanted to improve the then existing Arabic sentiment lexicons since they had very low coverage. They then generated Arabic Translations of English Sentiment Lexicons by collecting Arabic tweets from Twitter and from the Arabic translation of NRC Emotion Lexicon.^[3]
- Park and Kim used a dictionary-based approach which consists of a list of predefined opinion words that were collected manually. They discovered that traditional dictionary-based approach is not enough and found the need to expand on the lexicon and build a new expansive thesaurus lexicon would increase the accuracy of the sentiment classification.^[4]
- Bautin et al. explored the concept of international sentiment analysis by using the Lydia text analysis system. The Lydia system

recognizes named entities in text and extract their temporal and spatial distribution and were able to analyze nine different languages. They felt the need to develop a system that would not lose the meaning of the word by simply translating the words to English and then performing sentiment analysis on them. They discovered that the calculated sentiment scores ended up being consistent across various languages without the need of a translator. They also proposed a cross-language analysis sentiment polarity score calculation that works across other cross-cultural comparisons.^[5]

- S.H. Muhammad et al. proposed the first large-scale human-annotated Twitter sentiment dataset for four of the most widely spoken languages in Nigeria which consisted of 30,000 annotated tweets per language. They introduced methods for human annotators to manually decide the sentiment with the subjectivity voting of a text – if it has three positive vote the sentiment is positive, if it has three negative votes, the sentiment is negative, if it has two positive or two negative and the third vote is the opposite then they consider the majority vote to be of that sentiment. They also created sentiment lexicons for three Nigerian languages based on the NaijaSenti dataset. For sentiment analysis, they opted to use multiple local variants of BERT (Bidirectional Encoder

Representations from Transformers) – a multilingual variant of BERT trained on 104 languages called mBERT, a scaled-up mBERT called RemBERT which decouples embeddings and enable larger embedding output sizes during pre-training, a RoBERTa-style model called AfriBERTA which is trained on 11 African languages, mDEBERTaV3 and XLM-R.^[6]

- Yan et al. presented a unified generative framework for Aspect-based Sentiment Analysis (ABSA) which aimed to address multiple ABSA tasks within a single model. ABSA identifies aspect terms, sentiment polarities, and opinion terms in text. They proposed an approach where each task is a generative task. They use a pre-trained sequence-to-sequence model BART to solve all ABSA tasks.^[7]
- Singh et al. performed sentiment analysis with the help of the BERT model on Twitter data sets to understand the public sentiments and opinions regarding the impact of COVID-19. One data set is collected by tweets from all around the world, and the other data set contains tweets made from accounts in India. The sentiment analysis is done by considering the polarity scores of the TextBlob library.^[8]
- Lyu et al. studied the sentiments in the COVID-19 vaccine-related discussion on Twitter tweets from the day WHO declared COVID-

19 a pandemic on March 11, 2020, to January 31, 2021. The dataset consisted of 1,499,421 unique tweets from 583,500 different users. The data consisted of discussions about vaccinations as the countries of the world progressed towards manufacturing its first vaccine. The resulting data showed the trust emotion reaching its peak on November 9, 2020 after Pfizer announced that its vaccine is 90% effective. They used syuzhet, which is a popular R package for sentiment and emotion analysis, and the National Research Council of Canada Emotion Lexicon for the dictionary. The increasingly positive sentiment around COVID-19 vaccines and distant amount of trust implies a higher acceptance of COVID-19 vaccines compared to the previous ones.^[9]

- Rustam et al. performed sentiment analysis on tweets relating to COVID-19 using multiple machine learning methods. They used RF which is used for classification and regression problems that generates several trees and performs voting between them to make a majority decision. They used XGBoost which is a Gradient Boosting classifier that assigns weight to each sample, and has regularization techniques to control over-fitting of data. They used SVC which is a linear model used for sentiment analysis. They used Extra Trees Classifier (ETC) which trains and fits the number

of weak learners randomized on decision trees and boosts the prediction accuracy. They used the TextBlob library to find the sentiment polarity scores. They also used feature-extraction techniques such as TF-IDF, BoW and a combination of TF-IDF and BoW. They conclude that ETC is the best performer of them all with an accuracy of 93%, outperforming RF and XGBoost who both showed an accuracy of 92%.^[10]

- Nandwani and Verma discuss the various approaches of sentiment analysis and found while the lexicon-based approach performs well in both sentiment and emotion analysis, the dictionary-based approach is more adaptable and easier to apply.^[11]

CHAPTER THREE:

PROPOSED SYSTEM

Proposed System

This project aims to develop a model that accurately categorizes Amazon reviews as positive, negative, or neutral. It will employ several machine learning techniques that can help classify sentiment in Amazon reviews with high accuracy. Such sentiment analysis is useful for both consumers looking to explore a product or service and marketers seeking to gauge public sentiment about their business.

To achieve accurate sentiment classification, my proposed sentiment analysis system consists of preprocessing techniques, lexical resources, and deep learning models. The initial step is text preprocessing, which involves lowercasing, removing punctuation, tokenizing, eliminating irrelevant words by removing stopwords, and handling word contractions. We utilize existing sentiment lexicons like Opinion Lexicon to identify positive and negative words and assign sentiment scores to the text. These lexicons provide valuable resources for sentiment classification. In addition to using opinion sentiment lexicons, we will also use TextBlob which uses its own internal sentiment analysis capabilities to find the text's emotional tone. TextBlob's internal analysis can offer an enhanced accuracy and classification compared to Opinion Lexicon.

We will then calculate sentiment scores for each text by aggregating the sentiment polarities of individual words using Opinion Lexicon and TextBlob. In

addition, we employ advanced sentiment analysis models like LSTM (Long Short-Term Memory)^[12] networks to capture contextual information and sequential dependencies within the text for more accurate sentiment classification. The sentiment analysis system is trained on a labeled dataset, where sentiments are categorized as positive, negative, and neutral.

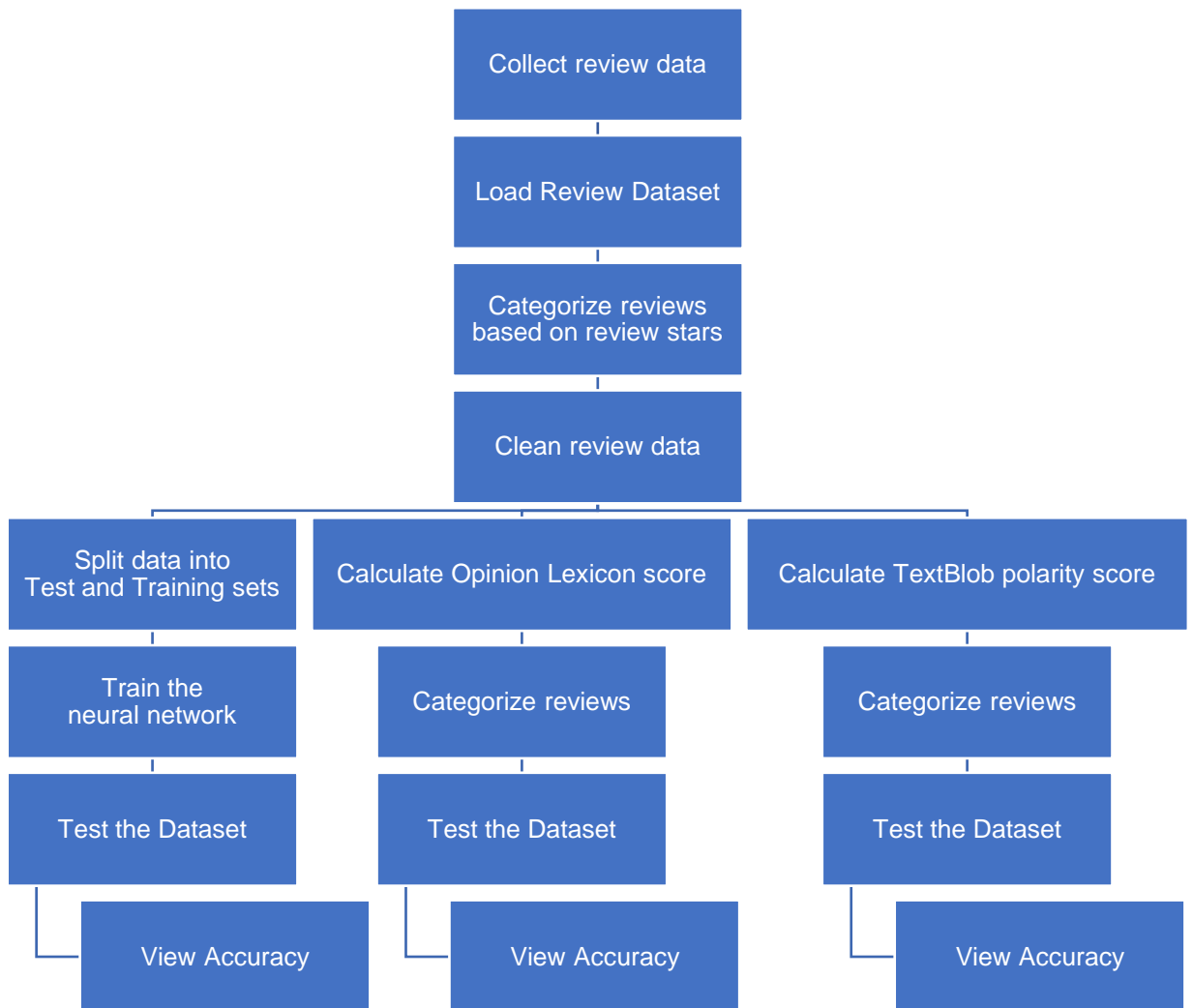


Figure 1. Block Diagram of Model

The block diagram in Figure 1 illustrates the different phases involved in working with the training model. I gathered review data from Amazon for select products using a Google Chrome browser extension. The raw reviews are carefully collected and compiled into a structured dataset for further analysis. The loaded data is processed to ensure data consistency by removing any whitespaces and punctuations and converting all text to lowercase. Next, the review text is categorized with three methods - by classifying the reviews based on the review stars on Amazon, by classifying the reviews using the sentiment score based on Opinion Lexicon, and by classifying the reviews using the sentiment polarity score of the text based on TextBlob.

Once the text is categorized, we will train the neural network with TensorFlow Keras' Sequential model. The neural network will be trained on the classification labels of the review star method and will learn what texts are positive, negative, or neutral. After the training and testing we will evaluate each classification's performance with Confusion Matrix metrics.

System Specifications

Hardware Specifications

	Minimum	Recommended
Processor	Intel i5 7500 or AMD Ryzen 3 3300X	Intel i7 10700K or AMD Ryzen 9 3900X
Graphics Card	NVIDIA GTX 1050	NVIDIA GTX 2080 Ti or newer
RAM	8 GB	16 GB
Disk Space	5 GB*	5 GB*
	*Includes Libraries. Dataset size may also vary	*Includes Libraries. Dataset size may also vary

Software Specifications

Operating System	Windows 10 or higher
Languages	Python 3.10
IDE	PyCharm, Jupyter Notebook
Framework	Flask
Libraries	TensorFlow, Keras, Pandas, TextBlob, NLTK, Scikit-Learn, Matplotlib

System Design

Module 1. User

- Users can view the dataset
- Users can input text
- Users can select Predict Sentiment
- Users can select Translate Sentiment

Module 2. System

Data Collection:

- The system collects customer reviews from a certain source, such as an online marketplace or review platform.
- The data is typically in text, accompanied by star ratings or other sentiment indicators.

Data Preprocessing:

- The collected data undergoes preprocessing to clean and prepare it for analysis.
- Text cleaning techniques are applied to remove noise, such as punctuation, stopwords, and convert the text to lowercase.
- Word contractions are expanded to their full forms for better analysis.
- The cleaned text is stored in a separate column in the dataset.

Sentiment Analysis:

- The system performs sentiment analysis on the preprocessed text to determine the sentiment expressed in each review.
- The sentiment is categorized as positive, negative, or neutral based on the star ratings associated with the reviews.
- A sentiment score is calculated for each review using Opinion Lexicon.
- The sentiment score helps in quantifying the sentiment of the reviews.
- A sentiment polarity score is calculated for each review using TextBlob.
- The sentiment polarity score helps in quantifying the sentiment tone of the reviews.

Training Data Preparation:

- The system prepares the data for training a sentiment classification model.
- The reviews and their corresponding sentiments are split into training and testing sets.
- The target sentiments are encoded using one-hot encoding for further processing.

Tokenization and Embedding:

- The system tokenizes the text data using a tokenizer.

- Word embedding is performed to convert the text tokens into numerical vectors.
- A pre-trained word embedding model, GloVe, creates an embedding matrix.
- The embedding matrix maps words to their corresponding vector representations.

Model Architecture:

- The system designs a deep-learning model for sentiment classification.
- The model architecture includes an embedding layer, recurrent layers (such as LSTM), and dense layers.
- The embedding layer utilizes the pre-trained embedding matrix to capture semantic information.
- The recurrent layers help capture sequential dependencies and understand the text's context.
- The dense layers perform the final classification and generate predictions.

Model Training and Evaluation:

- The prepared training data and model architecture train the sentiment classification model.
- The model is trained using the training set, with a specified number of epochs and batch size.

- The model's performance is evaluated during training using a validation set to monitor the loss and accuracy.
- Early stopping and model checkpointing techniques are applied to prevent overfitting and save the best model.

Model Deployment and Usage:

- Once the model is trained and evaluated, it can be saved for future use.
- The saved model can be loaded and deployed to classify the sentiment of new customer reviews.
- The deployed model inputs the preprocessed text, performs tokenization and padding, and generates sentiment predictions.

System Output and Reporting:

- The system outputs the predicted sentiment labels for the customer reviews.
- The predictions can be used to analyze and understand the sentiment trends in the collected reviews.
- Reports and visualizations can be generated to present insights, such as sentiment distribution, sentiment changes over time, or sentiment comparison across different products or categories.

CHAPTER FOUR

DATA AND DATA PREPROCESSING

Data Collection

I used a Google Chrome Extension that downloads review data of a single product on Amazon.com to a single CSV file. The extension had a limit of 1,000 reviews per product, so I had to manually select more than 80-90 products and download the reviews of each of them. I had to manually merge the reviews into one single CSV file since the extension did not allow me to select and download reviews from multiple products at a time.

The review data consists of redundant columns that will not make of any use to our project like Review Summary, Product ID, Reviewer ID, Reviewer Name, Verified Purchase, Style, UNIX Time, and Votes. We will only consider the “Overall” column, which is the stars left by the customer, and the “Review Text” column. We will make a copy of these two columns into a new Pandas Dataframe.

Text Cleanup

Text cleaning aims to prepare text data for analysis and machine learning tasks, where clean and consistent data is essential. This implements various text cleaning techniques such as removing punctuation and stopwords and converting text to lowercase. I used the following text cleaning techniques to preprocess the data before moving to classifying them:

- Converting to lowercase: It is important to ensure that all text is in the same case to maintain consistency and avoid duplicate entries.
- Word contractions: The contractions library is used to expand common word contractions such as “can’t” to “cannot”, “won’t” to “will not” “didn’t” to “did not” “you’re” to “you are”, “I’m” to “I am”, etc. This ensures that similar phrases are treated as identical.
- Removing punctuation: The `remove_punctuation()` function takes a text input and removes all punctuation. The `string.punctuation` list of punctuations include: “!, ", #, \$, %, &, ', (,), *, +, ,, -, ., /, :, ;, <, =, >, ?, @, [, \,], ^, _, ` , {, |, }, ~”
- Removing stopwords: We use the NLTK Stopwords library to define a set of English stopwords. The `remove_stopwords()` function removes all stop words from the text. Stop words are words that do not add much meaning to the words are removed with this function. The input text is split into words and any words that appear in the Stopwords set are filtered out. The NLTK Stopwords library includes: “*other, which, you, hasn, above, and, nor, yours, here, out, so, during, in, d, we, off, that'll, there, are, on, should've, for, them, do, it, having, he, wouldn't, from, while, all, the, aren't, if, ours, was, ll, our, or, each, just, not, is, won, how, be, don, their, yourself, between, ain, mightn't, what, very, isn, both, further, whom, too, its, my, mustn, will, you're, no, didn't, shouldn, more, ve,*”

y, through, by, you've, him, it's, below, doesn't, as, can, until, ma, been, his, mustn't, you'd, an, again, once, those, re, into, hadn't, herself, a, only, under, why, down, needn, of, doing, few, because, m, itself, am, who, should, at, to, hasn't, don't, isn't, needn't, o, shouldn't, hers, doesn, haven't, against, yourselves, being, same, wasn't, themselves, this, hadn, won't, himself, most, theirs, shan't, they, weren, up, had, that, wasn, ourselves, she's, were, couldn, her, some, aren, did, before, such, your, myself, shan, wouldn, me, has, didn, i, does, but, with, after, any, couldn't, about, these, she, over, where, s, weren't, you'll, when, own, haven, t, than, then, now, have, mightn.” Additionally, we also remove any words consisting only of digits.

- Removing whitespaces: We remove any leading or trailing whitespaces from the text to ensure consistency.

Before cleaning the text, an example review looks like this:

“I had two 27in monitors that pump out heat big time. I needed another for work, so I bought one of these. Nice, light, bright, and no heat. After a week I bought another for my gaming PC. They're beautiful. I didn't get the ones with the adjustable stand. I didn't need that and they're perfect.Pro tip: They're shipped in the box that shows what's inside, so the 2nd time I checked the box in checkout for Amazon to pack it in one of their boxes.”

After preprocessing the text, the review will look like this:

“two 27 in monitors pump heat big time needed another work bought one
nice light bright heat week bought another gaming pc beautiful get ones
adjustable stand need perfectpro tip shipped box shows inside 2nd time checked
box checkout amazon pack one boxes“

CHAPTER FIVE

METHODOLOGIES

Classification with Opinion Lexicon

Lexicon refers to a person's vocabulary, language, or branch of knowledge. A predefined dictionary of words labeled positive, negative, or neutral is used in lexicon-based sentiment analysis. To determine the sentiment of a sentence, it is tokenized, and each token is matched with the available words in the model. Opinion Lexicon is a list of around 6800 words with positive and negative connotations created by Minqing Hu and Bing Liu^[13]. It can be used to analyze sentiment in text data. We will create two lists, `pos_words` and `neg_words`, by extracting the positive and negative words from the Opinion Lexicon. We will be comparing the words in each review text with the words that appear in these lists to find if they have a positive or negative sentiment.

`pos_words` contains a list of 2006 positive words, such as "love," "happy," "excellent," "amazing," "beautiful," etc. These words have a positive sentiment and can be used to identify positive sentiment in text data. On the other hand, the `neg_words` list contains approximately 4783 negative words, including words like "hate," "sad," "terrible," "disgusting," "ugly," etc. These words have a negative sentiment and can be used to identify negative sentiment in text data.

We will create a method that will classify the review according to the three sentiment scores. The function `get_sentiment_score()` will calculate the sentiment score for each text. We will do this by first tokenizing each sentence of

the text using the TreebankWord tokenizer. Then we will be counting the number of positive and negative words in each sentence. For each sentence in the text, it will increment the score by 1 for each positive word that appears in the text and decrement the score by 1 for each negative word and then takes the average score of all the sentences in the text.

The function then returns the total score. The result is a float value between -1 and 1, where negative values indicate a negative sentiment and positive values indicate a positive sentiment. Neutral sentiments have a score of 0.

Classification with TextBlob

TextBlob^[14] is a versatile Python library for Natural Language Processing that facilitates a wide range of text analysis tasks, including sentiment analysis, part-of-speech tagging, tokenization, and more. It is built on the shoulders of NLTK and offers an intuitive way for NLP tasks.

We will be using the sentiment analysis feature of TextBlob to determine the sentiment and tone of the review. Unlike lexicon-based sentiment analysis which relies on predefined dictionaries, TextBlob employs machine learning techniques to analyze text sentiment. TextBlob sentiment analysis generates two key metrics: polarity and subjectivity. Polarity determines the sentiment's orientation, ranging from -1 (negative) to 1 (positive), while subjectivity determines the opinion of the text. We will only consider Polarity in our project as

we are more inclined towards classifying the reviews as Positive, Negative, or Neutral.

We will create a function similar to that of the Opinion Lexicon sentiment scores that will classify the review according to the sentiment polarity scores. The function `calculate_sentiment_polarity()` will calculate the sentiment polarity for each text using its own internal sentiment analysis methods. TextBlob will determine the sentiment of the text by first tokenizing the text into sentences and words. It then calculates the sentiment polarity for each sentence, and then these sentence-level scores are aggregated to produce an overall polarity score for the entire text. The result is a float value between -1 and 1, where negative values indicate negative sentiment and positive values indicate positive sentiment. Neutral sentiments have a score of 0.

Classification with Neural Network

Using the pre-trained classification labels (in our case – positive, negative, and neutral), neural networks can be trained to enhance the classification's accuracy and consistency. The neural network architecture consists of multiple layers, including input layers, hidden layers, and an activation function. Each layer consists of neurons or nodes that are interconnected, enabling information flow and transformation.

The first layer of the neural network model is the embedding layer, which converts tokenized words into numerical vectors and is capable of capturing the words' semantics. This allows the model to recognize patterns and relationships

in the text. Then we have a recurrent neural network (RNN) layer, in our case we have the Long Short-Term Memory (LSTM) layer. This layer processes sentences into numerical sequences of a fixed length, which allows that model to capture temporal dependencies and patterns within the text. The model also has a Dense layer with a sigmoid activation function, which is added for further refinement and learning and can adapt to complex representations of the text. The sigmoid activation maps the output values between 0 and 1, which is used in binary classification tasks.

The training phase is where the model learns to make classification by adjusting its weights based on the classification data from the review stars. In this phase, we will use the loss function and the optimizers we have set when we compile the model. For the loss function we use Categorical-crossentropy, a loss function that is commonly used for multi-class classification tasks where the targets are one-hot encoded. This function calculates the cross-entropy loss between the ground truth labels and the prediction distribution.

Then we will use the Adam optimizer to minimize the loss function. Adam is an adaptive learning rate optimization algorithm that adapt the learning rates during training and is faster than most optimization algorithms.

We begin by setting up a tokenizer object using the `Tokenizer()` class from Keras' preprocessing.text module.^{[15][16]} This object fits the preprocessed review text and builds a vocabulary of words that will be used to represent each review as a sequence of integers.

The `word_index` object will return a dictionary containing the word-to-index mappings learned by the tokenizer during the fitting step. This dictionary can convert new text data into sequences of integers that can be fed into a machine-learning model. Saving tokenizers is a good practice when working with text data in machine learning workflows, as it allows the same vocabulary and word-to-index mappings to be used consistently across different model runs.

We then save the tokenizer object created using the Pickle library. This object contains the vocabulary and word-to-index mappings learned from the preprocessed reviews. It will convert new text data into sequences of integers that can be fed into a machine-learning model.

We will perform a train-test split on the preprocessed review data and the encoded sentiment labels in the final DataFrame, using the `train_test_split()` function from the scikit-learn's `model_selection` module.^[17] This will split the data into a training set and testing set. The split is performed with a test size of 0.3 – meaning 30% of data will be used for testing. The resulting split data is stored in four variables: `X_train`, `Y_train`, `x_test`, and `y_test`.

To ensure good performance and overall accuracy of the model, we need to perform a preprocessing step called Sequence Padding on our data.

Sequence padding is a preprocessing technique that converts all texts into numerical sequences, and it ensures all sequences in a batch have the same length. This is important because most deep learning architectures require a fixed-length input. Padding is a technique used to make all sequences the same

length for processing by an LSTM. Padding affects the way the networks function and can make a big difference in the way the models perform and have varying accuracies.

To perform Sequence Padding, we define a function `sequence_padding()` that uses the provided tokenizer to convert the input sentences into numerical sequences using Keras' `texts_to_sequences` function from the `Tokenizer` class. It then pads and truncates these numerical sequences to ensure they all have the same length. These padded and truncated numerical vector sequences are stored in the `train_set` and `test_set` variables.

We will then label data by converting them to One-Hot Encoding vectors for the softmax function in the neural network. One-hot encoding is a technique that converts categorical variables into binary vectors. The categorical values are first mapped to integer values, and then each integer value is represented as a binary vector.

Then the `read_glove_vector()` function reads a pre-trained GloVe vector file^[18] and returns a dictionary where each word is a key that maps to its corresponding vector representation. GloVe (Global Vectors for Word Representation) is an unsupervised learning algorithm that generates word embeddings.^[19] The algorithm maps words into a space where the distance between words is related to semantic similarity. GloVe is trained on word co-occurrences from a corpus like Wikipedia. For example, "cat" and "dog" tend to occur with similar other words.

In this function, we begin by initializing two data structures - an empty set called `words` and a dictionary called `word_to_vec_map`. Word2vec^[20] is a natural language processing (NLP) technique that employs a neural network model to grasp word relationships within extensive text data. This trained model can identify synonymous words and offer word suggestions for incomplete sentences.^[21] It creates a representation of each word in the vocabulary into a binary vector.

The `read_glove_vector` function opens the GloVe file in read mode and iterates over each line. It splits each line into words and retrieves the current word and its vector representation, converting it from a list of strings to a numpy array of float64 data type. The current word and its vector representation are added to the `word_to_vec_map` dictionary.

We will initialize the embedding matrix with the size of the vocabulary and the length of the embedding vector. We will iterate through each word in the vocabulary that we learned in the previous steps and get its corresponding embedding vector from `word_to_vec_map`. If the pretrained vector exists for the word, we assign the corresponding row of the embedding matrix to that vector. This embedding matrix will be used later as an input to the embedding layer of the neural network for training.

After classifying each text, we create a training model using TensorFlow's Keras library for natural language processing. The model architecture includes three layers: an embedding layer, a long short-term memory (LSTM) layer, and

three dense layers with different activation functions. Let us look at what these layers do:

- **Embedding layer:** The embedding layer is an essential component of a neural network model, as it is the first layer that receives the input data. Its primary function is to map the integer-encoded reviews to a fixed-size vector space. They take one-hot word vectors as inputs and output a dense vector of a specified dimensionality. Each dimension represents a latent feature of the category. This way, the embedding layer can help the neural network learn the semantic relationships between words in the text data. In this specific model, we use pre-trained GloVe word embeddings.
- **LSTM layer:** The Long Short-Term Memory (LSTM) layer is a Recurrent Neural Network (RNN) layer well suited to sequential data^[22]. The LSTM layer is used to learn the temporal dependencies in the sequence of words in the reviews. We use a single LSTM layer in this model with 128 units. The number of units in the LSTM layer is a hyperparameter that we use to improve the performance of the model.
- **Dense layers:** After the LSTM layer, we add a dense layer with 64 units and a sigmoid activation function. The Dense connects every neuron from the previous layer to every neuron in the current layer.

This layer will take the output of the LSTM layer and will apply a non-linear transformation to the data.

- Output layer: The output layer is the final layer in a neural network that produces predictions. It takes inputs from the previous layers and performs calculations using its neurons to produce the output. The output layer has its own set of weights that are applied before the final output is derived. This model has an output layer 3 units with a sigmoid activation function. The output layer takes in the output of the second dense layer and produces a probability distribution over the 3 sentiment classes: positive, negative, and neutral. The softmax activation function is used in the output layer to ensure that the predicted probabilities sum up to 1.

Then we will create a function that creates a neural network model using the Keras Sequential API. The model is then compiled using three metrics: the 'categorical_crossentropy' loss function, the Adam optimizer, and the 'accuracy' metrics.

- The categorical_crossentropy loss function is frequently used in multi-class classification tasks. It calculates the difference between the predicted probabilities for each class and the true one-hot encoded class labels, resulting in a single scalar value representing the model's total loss on the training data.

- The Adam optimizer^[23] is a popular stochastic gradient descent (SGD) optimization algorithm used in deep learning models. It is an adaptive learning rate optimizer that adjusts the learning rate of each weight in the model based on the history of its gradient updates.^[24] This allows the optimizer to converge to the optimal set of weights more quickly and with less tuning than traditional SGD.
- The accuracy metric assesses the performance of classification model. It measures the percentage of correctly classified examples out of the total number of examples. The accuracy metric is used to evaluate how well the model can correctly classify text into one of three output classes.

We will train the neural network model using the Keras' fit() method. The model is trained on the training set and its corresponding one-hot encoded labels.

CHAPTER SIX

EXPERIMENTAL RESULTS

Threshold Determination

When we assign the Opinion Lexicon Sentiment scores or the TextBlob Polarity scores to a review text, we must make sure that the threshold that we set is close enough to what the original review's sentiment or the ground truth label is. Since both the Sentiment and Polarity scores of Opinion Lexicon and TextBlob, respectively, have a range of -1 and 1, we must decide on how big of a threshold should the Neutral sentiment be. Therefore, I decided to further test what threshold should we set.

Opinion Lexicon Test Cases

Let us first look at the different threshold test cases for the Opinion Lexicon classification method. The First is the original threshold that we set where if the score is 0 then that is a Neutral review. The Second is where the score is in between 0 and 0.01 for neutral reviews. The Third is where the score is in between 0 and 0.02 for neutral reviews. The Fourth is where the score is in between 0 and 0.05 for neutral reviews. The Fifth is where the score is in between 0 and 0.1 for neutral reviews. The Fifth is where the score is in between 0.1 and -0.1 for neutral reviews.

Table 1. Opinion Lexicon Threshold Test Cases

Threshold	Accuracy	Positive	Negative	Neutral
0 < Neutral > 0	64.38%	69055	18899	12008
0 < Neutral > 0.01	64.23%	68577	18899	12486
0 < Neutral > 0.02	63.21%	65539	18899	15524
0 < Neutral > 0.05	58.51%	52736	18899	28327
0 < Neutral > 0.1	46.82%	48270	18899	32793
-0.1 < Neutral > 0.1	39.14%	32793	3523	63646

In Table 1, we can see the number of Positive, Negative, and Neutral reviews of each of the thresholds and compare their performance with the Accuracy metrics to find which threshold suits better for this classification. We find that setting the threshold with the Neutral reviews having a score of 0 has a much higher accuracy than the other thresholds, and the accuracy keeps decreasing drastically each time we increase the threshold by an increment of 0.01.

TextBlob Test Cases

Now we will look at the different threshold test cases for the TextBlob classification method.

Table 2. TextBlob Threshold Test Cases

Threshold	Accuracy	Positive	Negative	Neutral
0 < Neutral > 0	65.71%	79146	16123	4693
0 < Neutral > 0.01	65.62%	78068	16123	5771
0 < Neutral > 0.02	65.52%	76776	16123	7063
0 < Neutral > 0.05	64.46%	71806	16123	12033
0 < Neutral > 0.1	61.51%	61811	16123	22028
-0.1 < Neutral > 0.1	57.10%	61811	6936	31215

In Table 2, we can see the number of Positive, Negative, and Neutral reviews of each of the thresholds and compare their performance with the Accuracy metrics to find which threshold suits better for this classification. We find that setting the threshold with the Neutral reviews having a score of 0 has a much higher accuracy than the other thresholds.

Classification Model Evaluation

Confusion matrix is a table that visualizes the performance of the algorithm. It is a contingency table of two dimensions – “actual” and “predicted” values. We will use Confusion Matrix to evaluate the results of Opinion Lexicon Classification, TextBlob Classification, and the Neural Network Predictions. To calculate the confusion matrix, we use the `confusion_matrix` function from `scikit-learn`. This function takes the true sentiment labels and predicted sentiment labels obtained using the Opinion Lexicon sentiment scores as inputs. We will also use the function by taking the true sentiment labels and predicted sentiment labels obtained using the TextBlob sentiment scores as inputs. And then we will also use the function by taking the true sentiment labels and predicted sentiment labels obtained after training the neural network. The resulting confusion matrix provides us with information on how well the model has predicted the sentiment labels. It is a table with four cells representing the number of samples that belong to a particular combination of true and predicted sentiment labels. The four cells correspond to true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions.

- True Positive (TP): TP are the cases that accurately predict that the text is positive.
- True Negative (TN): TN are the cases that accurately predict that the text is negative.

- False Positive (FP): FP is when a sentiment was predicted as positive but is negative.
- False Negative (FN): FN is when a sentiment was predicted as negative but is positive.

The confusion matrix shows the performance of the classification models of Opinion Lexicon, TextBlob and the Neural Network. The matrix has three classes, 'positive,' 'negative,' and 'neutral,' on both the true and predicted axes. The matrix shows the number of instances that were classified into each category. The diagonal values show the number of correctly classified instances, while the off-diagonal values show the misclassifications.

We will calculate the performance metrics that will help assess the quality of each classification model. The key performance metrics include:

- Accuracy: The ratio of the correctly labeled subjects to the whole pool of subjects.
$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$
- Precision: Precision is also known as Positive Prediction Value (PPV). It is the ratio of the correctly positively labeled subjects to all the positively labeled subjects.
$$\text{Precision} = TP / (TP + FP)$$
- Recall: Recall is the ratio of correctly true labeled subjects to all true labeled subjects. It is also known as Hit Rate or True Positive

Rate (TPR).

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

- F1-score: It is the mean of precision and recall.

$$\text{F-1 score} = (2 * \text{TP}) / ((2 * \text{TP}) + (\text{FP} + \text{FN}))$$

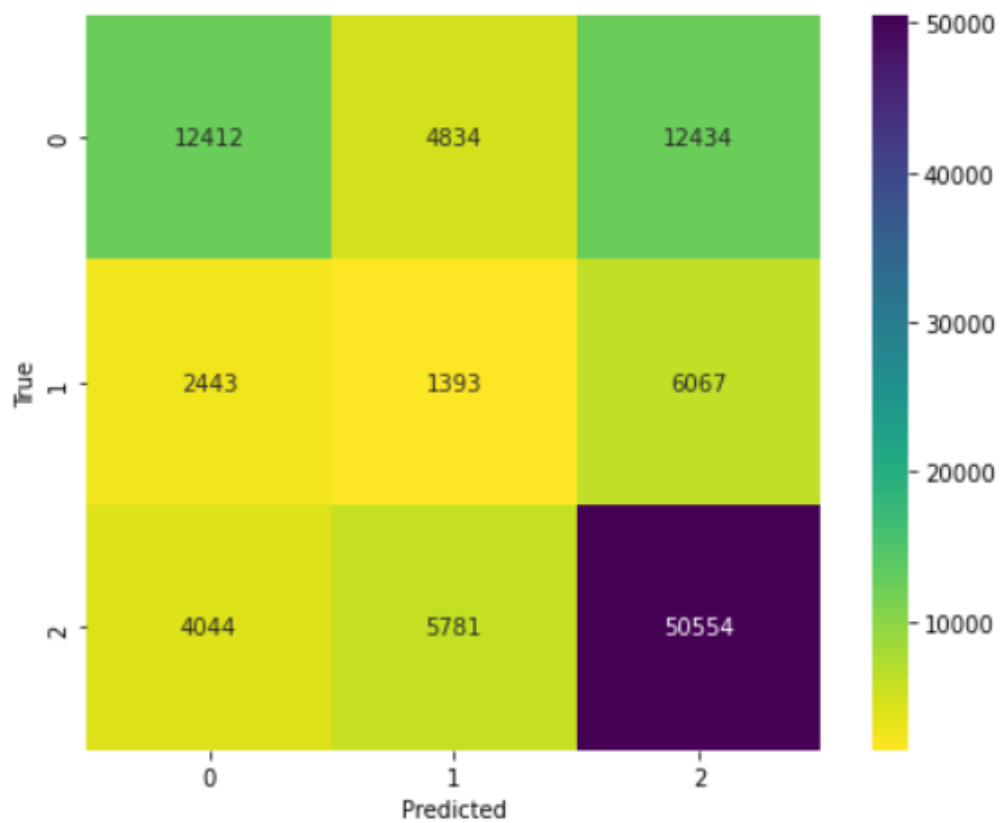


Figure 2. Confusion Matrix for Opinion Lexicon Classification

The confusion matrix, as we see in Figure 2, shows that the model better classified the positive and negative reviews than the neutral ones. The model

correctly classified 50,554 positive reviews and misclassified 9,825 positive reviews. Similarly, the model correctly classified 12,412 negative reviews and misclassified 17,268 negative reviews. On the other hand, the model only correctly classified 1,393 neutral reviews and misclassified 8,510 neutral reviews.

We will calculate the accuracy of this model by considering the True Positives, True Negatives, False Positives, and False Negatives for each category:

- True Positives (TP):
 - Negative: 12412
 - Neutral: 1393
 - Positive: 50554
- True Negatives (TN):
 - Negative: $1393 + 6067 + 5781 + 50554 = 63795$
 - Neutral: $12412 + 12434 + 4044 + 5781 = 34671$
 - Positive: $12412 + 4834 + 2443 + 1393 = 21082$
- False Positives (FP):
 - Negative: $2443 + 4044 = 6487$
 - Neutral: $4834 + 5781 = 10615$
 - Positive: $12434 + 6067 = 18501$
- False Negatives (FN):
 - Negative: $4834 + 12434 = 17268$
 - Neutral: $2443 + 6067 = 8510$

- Positive: $4044 + 5781 = 9825$

We then calculate the Accuracy, Precision, Recall and F-1 score performance metrics:

- Accuracy:
 - Accuracy (Negative) = $(12412 + 63795) / (12412 + 63795 + 6487 + 17268) = 0.7623 = 76.23\%$
 - Accuracy (Neutral) = $(1393 + 34671) / (1393 + 34671 + 10615 + 8510) = 0.6534 = 65.34\%$
 - Accuracy (Positive) = $(50554 + 21082) / (50554 + 21082 + 18501 + 9825) = 0.7166 = 71.66\%$
- Precision:
 - Precision (Negative) = $12412 / (12412 + 6487) = 0.6567 = 65.67\%$
 - Precision (Neutral) = $1393 / (1393 + 10615) = 0.1160 = 11.60\%$
 - Precision (Positive) = $50554 / (50554 + 18501) = 0.7320 = 73.20\%$
- Recall:
 - Recall (Negative) = $12412 / (12412 + 17268) = 0.4181 = 41.81\%$
 - Recall (Neutral) = $1393 / (1393 + 8510) = 0.1406 = 14.06\%$

- Recall (Positive) = $50554 / (50554 + 9825) = 0.8372 = 83.72\%$
- F-1 Score:
 - F1-Score (Negative) = $(2 * 12412) / ((2 * 12412) + (6487 + 17268)) = 0.5110 = 51.10\%$
 - F1-Score (Neutral) = $(2 * 1393) / ((2 * 1393) + (10615 + 8510)) = 0.1271 = 12.71\%$
 - F1-Score (Positive) = $(2 * 50554) / ((2 * 50554) + (18501 + 9825)) = 0.7811 = 78.11\%$

Then, the overall accuracy for Opinion Lexicon Classification

= $\Sigma(\text{True Positive}) / \text{Total Number of Reviews}$

= $(12412 + 1393 + 50554) / 99962$

= $0.6438 = 64.38\%$

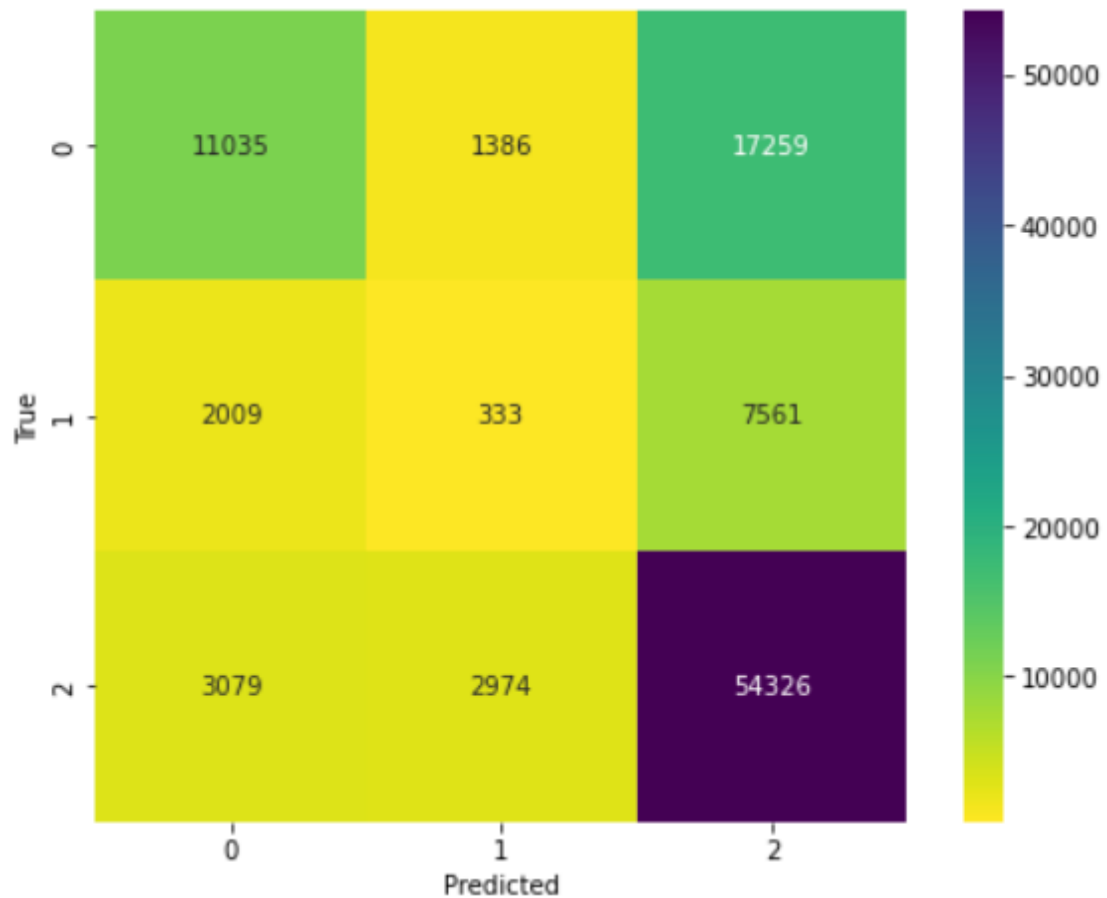


Figure 3. Confusion Matrix for TextBlob Classification

The confusion matrix, as we in see in Figure 3, shows that the model better classified the positive and negative reviews than the neutral ones. The model correctly classified 54,326 positive reviews and misclassified 6,053 positive reviews. Similarly, the model correctly classified 11,035 negative reviews and misclassified 18,645 negative reviews. On the other hand, the model only correctly classified 333 neutral reviews and misclassified 9,570 neutral reviews.

We will calculate the accuracy of this model by considering the True Positives, True Negatives, False Positives, and False Negatives for each category:

- True Positives (TP):
 - Negative: 11035
 - Neutral: 333
 - Positive: 54326
- True Negatives (TN):
 - Negative: $333 + 7561 + 2974 + 54326 = 65194$
 - Neutral: $11035 + 17259 + 3079 + 54326 = 85699$
 - Positive: $11035 + 1386 + 2009 + 333 = 14763$
- False Positives (FP):
 - Negative: $2009 + 3079 = 5088$
 - Neutral: $1386 + 2974 = 4360$
 - Positive: $17259 + 7561 = 24820$
- False Negatives (FN):
 - Negative: $1386 + 17259 = 18645$
 - Neutral: $2009 + 7561 = 9570$
 - Positive: $3079 + 2974 = 6053$

We then calculate the Accuracy, Precision, Recall and F-1 score performance metrics for each category:

- Accuracy:

- Accuracy (Negative) = $(11035 + 65194) / (11035 + 65194 + 5088 + 18645) = 0.7625 = 76.25\%$
- Accuracy (Neutral) = $(333 + 85699) / (333 + 85699 + 4360 + 9570) = 0.8606 = 86.06\%$
- Accuracy (Positive) = $(54326 + 14763) / (54326 + 14763 + 24820 + 6053) = 0.6911 = 69.11\%$
- Precision:
 - Precision (Negative) = $11035 / (11035 + 5088) = 0.6844 = 68.44\%$
 - Precision (Neutral) = $333 / (333 + 4360) = 0.0709 = 7.09\%$
 - Precision (Positive) = $54326 / (54326 + 24820) = 0.6864 = 68.64\%$
- Recall:
 - Recall (Negative) = $11035 / (11035 + 18645) = 0.3717 = 37.17\%$
 - Recall (Neutral) = $333 / (333 + 9570) = 0.0336 = 3.36\%$
 - Recall (Positive) = $54326 / (54326 + 6053) = 0.8997 = 89.97\%$
- F-1 Score:
 - F1-Score (Negative) = $(2 * 11035) / ((2 * 11035) + (5088 + 18645)) = 0.4818 = 48.18\%$

- F1-Score (Neutral) = $(2 * 333) / ((2 * 333) + (4360 + 9570)) = 0.0456 = 0.45\%$
- F1-Score (Positive) = $(2 * 54326) / ((2 * 54326) + (24820 + 6053)) = 0.7787 = 77.87\%$

Then, the overall accuracy for TextBlob Classification

$= \Sigma(\text{True Positive}) / \text{Total Number of Reviews}$

$= (11035 + 333 + 54326) / 99962$

$= 0.6571 = 65.71\%$

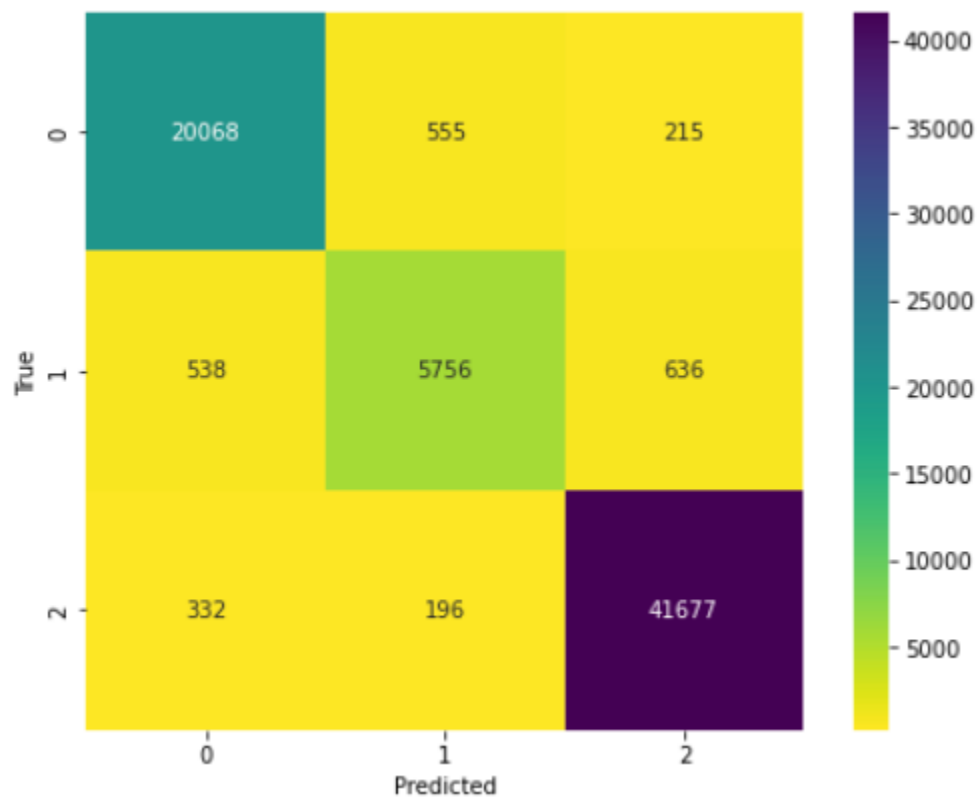


Figure 4. Confusion Matrix for Neural Network Classification

The confusion matrix, as we see in Figure 4, shows the model classifying the reviews by learning the review star labels. We use the training labels that we used on the `train_test_split` function and evaluate the true labels with the neural network's predicted labels. The model correctly classified 41,736 positive reviews and misclassified 469 positive reviews. Similarly, the model correctly classified 20,347 negative reviews and misclassified 491 negative reviews. On the other hand, the model only correctly classified 5,927 neutral reviews and misclassified 1,003 neutral reviews.

We will calculate the accuracy of this model by considering the True Positives, True Negatives, False Positives, and False Negatives for each category:

- True Positives (TP):
 - Negative: 20068
 - Neutral: 5756
 - Positive: 41677
- True Negatives (TN):
 - Negative: $5756 + 636 + 196 + 41677 = 48265$
 - Neutral: $20068 + 215 + 332 + 41677 = 62292$
 - Positive: $20068 + 555 + 538 + 5756 = 26917$
- False Positives (FP):
 - Negative: $538 + 332 = 870$
 - Neutral: $555 + 196 = 751$

- Positive: $215 + 636 = 851$
- False Negatives (FN):
 - Negative: $535 + 215 = 750$
 - Neutral: $538 + 636 = 1174$
 - Positive: $332 + 196 = 528$

We then calculate the Accuracy, Precision, Recall and F-1 score performance metrics:

- Accuracy:
 - Accuracy (Negative) = $(20068 + 48265) / (20068 + 48265 + 870 + 750) = 0.9768 = 97.68\%$
 - Accuracy (Neutral) = $(5756 + 62292) / (5756 + 62292 + 751 + 1174) = 0.9724 = 97.24\%$
 - Accuracy (Positive) = $(41677 + 26917) / (41677 + 26917 + 851 + 528) = 0.9802 = 98.02\%$
- Precision:
 - Precision (Negative) = $20068 / (20068 + 870) = 0.9584 = 95.84\%$
 - Precision (Neutral) = $5756 / (5756 + 751) = 0.8845 = 88.45\%$
 - Precision (Positive) = $41677 / (41677 + 851) = 0.9799 = 97.99\%$
- Recall:

- Recall (Negative) = $20068 / (20068 + 750) = 0.9639 = 96.39\%$
- Recall (Neutral) = $5756 / (5756 + 1174) = 0.8305 = 83.05\%$
- Recall (Positive) = $41677 / (41677 + 528) = 0.9874 = 98.74\%$
- F-1 Score:
 - F1-Score (Negative) = $(2 * 20068) / ((2 * 20068) + (870 + 750)) = 0.9612 = 96.12\%$
 - F1-Score (Neutral) = $(2 * 5756) / ((2 * 5756) + (751 + 1174)) = 0.8567 = 85.67\%$
 - F1-Score (Positive) = $(2 * 41677) / ((2 * 41677) + (851 + 528)) = 0.9837 = 98.37\%$

Then, the overall accuracy for Neural Network

$$= \Sigma(\text{True Positive}) / \text{Total Number of Reviews}$$

$$= (20068 + 5756 + 41677) / 69973$$

$$= 0.9646 = 96.46\%$$

Classification Model Comparison

The following tables illustrate the test cases for the different classification methods. The accuracy varies with Opinion Lexicon, TextBlob and Neural Network. We will look at the three different test cases of each classification with example reviews from the dataset.

Positive Test Cases

Table 3. Positive Sentiment Test Case

Method	Accuracy	Input	Expected Output	Actual Output
Opinion Lexicon	64.38%	We've only had this for one day. I bought it for my grandson. He absolutely LOVES it!! I even like it. The mouse is very nice and fits his hand well, but it would probably fit most hands just fine. The lights are really pretty, but not too bright so it wouldn't disturb anyone else in the room. It's perfect so far!! I can't believe we got this for such a reasonable price. I hope it holds up well and my Grandson gets a few years of use out of it. I would buy it again in a heartbeat!	Positive	Positive
TextBlob	65.71%		Positive	Positive
Neural Network	96.46%		Positive	Positive

As you can see from Table 3, the sentence produces the same result for the Opinion Lexicon, TextBlob, and Neural Network classification methods.

Table 4. Positive Sentiment Test Case

Method	Accuracy	Review Text	Expected Output	Actual Output
Opinion Lexicon	64.38%	I have purchased 3 drive so far, one 1 gig, and two 2 gig drives. I use these as backup for my three computers to store documents, pictures, and various other media. I use one to travel with with my laptop, I have found these to be fast and reliable, and very compact to travel with. I am considering buying some for my daughters for their home computers as they are always running out of space and trying to burn disks which takes a while.	Positive	Positive
TextBlob	65.71%		Positive	Neutral
Neural Network	96.46%		Positive	Positive

In Table 4 for the Positive Test Case, the sentence produces a Neutral result for the TextBlob method instead of “Positive.” This is because the sentiment polarity score calculation for each of these texts was not high enough that they could meet the threshold of a “Positive” result.

Table 5. Positive Sentiment Test Case

Method	Accuracy	Review Text	Expected Output	Actual Output
Opinion Lexicon	64.38%	So..... yes so..... it works, ok it works, but yes it is slow; if you don't mind and just need some extra space for all your crap then buy it, but if you are too busy to wait..... then don't buy it, simple as that.	Positive	Neutral
TextBlob	65.71%		Positive	Negative
Neural Network	96.46%		Positive	Positive

In Table 5, we see mixed results as the review that was left on Amazon had a 5-star rating, so it is a “Positive” review. With the Opinion Lexicon classification, we get a “Neutral” result as the sentiment score calculation does not go higher than 0 as it also has a lot of negative words that decrement the score. With TextBlob, we get a “Negative” result with its internal sentiment analysis methods consider the words “don’t” in the review.

Negative Test Cases

Table 6. Negative Sentiment Test Case

Method	Accuracy	Review Text	Expected Output	Actual Output
Opinion Lexicon	64.38%	We were unable to get this to work. Enclosed "manual" was almost unintelligible and made no sense. Unfortunately, I kept trying too long and was unable to return it. I	Negative	Negative
TextBlob	65.71%		Negative	Negative
Neural Network	96.46%		Negative	Negative

		will never use this seller again.		
--	--	-----------------------------------	--	--

As you can see from Table 6, the sentence produces the same result for the Opinion Lexicon, TextBlob, and Neural Network classification methods.

Table 7. Negative Sentiment Test Case

Method	Accuracy	Review Text	Expected Output	Actual Output
Opinion Lexicon	64.38%	The keyboard and mouse are fully functional and the mouse has a good feel for gaming. However, the ESC, and F1 through F7 keys have a bad paint / screen job. See attached photos. Probably still worth the price, but I would have preferred a little QC on the finish.	Negative	Positive
TextBlob	65.71%		Negative	Neutral
Neural Network	96.46%		Negative	Negative

In Table 7, however, the example produces a “Positive” result for Opinion Lexicon and a “Neutral” result for the TextBlob method instead of “Negative.”

Table 8. Negative Sentiment Test Case

Method	Accuracy	Review Text	Expected Output	Actual Output
Opinion Lexicon	64.38%	At first it worked great and I was so excited to have a touch screen. But today it stopped working	Negative	Positive
TextBlob	65.71%		Negative	Positive
Neural Network	96.46%		Negative	Negative

In Table 8, the example produces a “Positive” result for TextBlob method instead of “Negative” for both the Opinion Lexicon and TextBlob methods. This is because the review has two positive words.

Neutral Test Cases

Table 9. Neutral Sentiment Test Case

Method	Accuracy	Review Text	Expected Output	Actual Output
Opinion Lexicon	64.38%	Bought it 5 years ago. It has always squeaked when you lean back in it, but after a year or so the squeak gets really loud. After about three years the arm cushions will fall off. After about 4 years the faux leather starts to peel off. The cushion never flattened out for me like I've seen other reviewers have said, I think that has to do with the weight of the person, I'm 6'1" and weigh 190 lbs, and the cushion is	Neutral	Neutral
TextBlob	65.71%		Neutral	Neutral
Neural Network	96.46%		Neutral	Neutral

		still decent after 5 years.		
--	--	-----------------------------	--	--

In Table 9, the example produces a “Neutral” result for all three classification methods.

Table 10. Neutral Sentiment Test Case

Method	Accuracy	Review Text	Expected Output	Actual Output
Opinion Lexicon	64.38%	USED Seagate (STGY8000400) Desktop 8TB External Hard Drive HDDOn initial use, I clocked 25MB/s write speed. That is 1/4 or 1/6 the speed of all my other USB 3.0 drives. I saw the top review and thought I got lemons. I went through all the troubleshooting steps I could find online and the solution was Seagate's software update. I suspect the product's firmware was rolled back prior to resell.UPDATEFor unknown reasons, speed will drop back down to 25MB/s, and other times it will run at normal speeds. When I have large data to transfer, I just restart my computer and hope for normal speeds. Minus 2 stars for the random inconvenience.	Neutral	Negative
TextBlob	65.71%		Neutral	Negative
Neural Network	96.46%		Neutral	Neutral

In Table 10, we get a “Negative” result instead of the intended “Neutral” result for Opinion Lexicon and TextBlob classification as the sentiment score and the sentiment polarity score is below -0.1 for the text.

Table 11. Neutral Sentiment Test Case

Method	Accuracy	Review Text	Expected Output	Actual Output
Opinion Lexicon	64.38%	I got it in Rose Gold, it's very dark Rose Gold and I wish I would've just got pink. The mouse works perfectly, the lights are cool and I love that it's chargeable but it is NOT comfortable to use with it's very flat design. There is zero wrist or palm support lol my hand cramps after a lot of use. I think I'll give it a couple weeks to see if I can get used to it. If not I'll give it away and order one with a slight bump for my palm.	Neutral	Positive
TextBlob	65.71%		Neutral	Positive
Neural Network	96.46%		Neutral	Neutral

In Table 11, the text produces a “Positive” result for the Opinion Lexicon and TextBlob methods instead of “Neutral” as the review has more positive words for the sentiment score and sentiment polarity score calculation to sum up higher than 0.

CHAPTER SEVEN:

USER INTERFACES

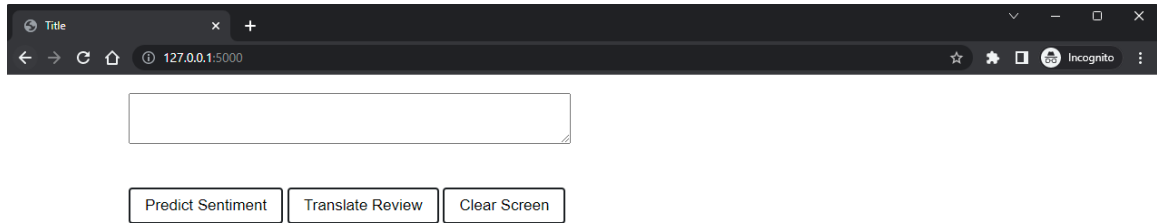


Figure 5. Homepage of the Application

Figure 4, as seen above, shows the project's user interface. The user interface is created using Flask's Frontend Framework.^[25] Here the user has 4 input options: a textbox to enter some text, a Predict Sentiment button, a Translate Review button, and a Clear Screen button to clear the screen of any

outputs. On the backend of the application, the trained neural network is loaded for the Predict Sentiment function to work. The Translate Review button uses the `translate()` method from the `iTranslate Python` library to translate and predict sentiments.

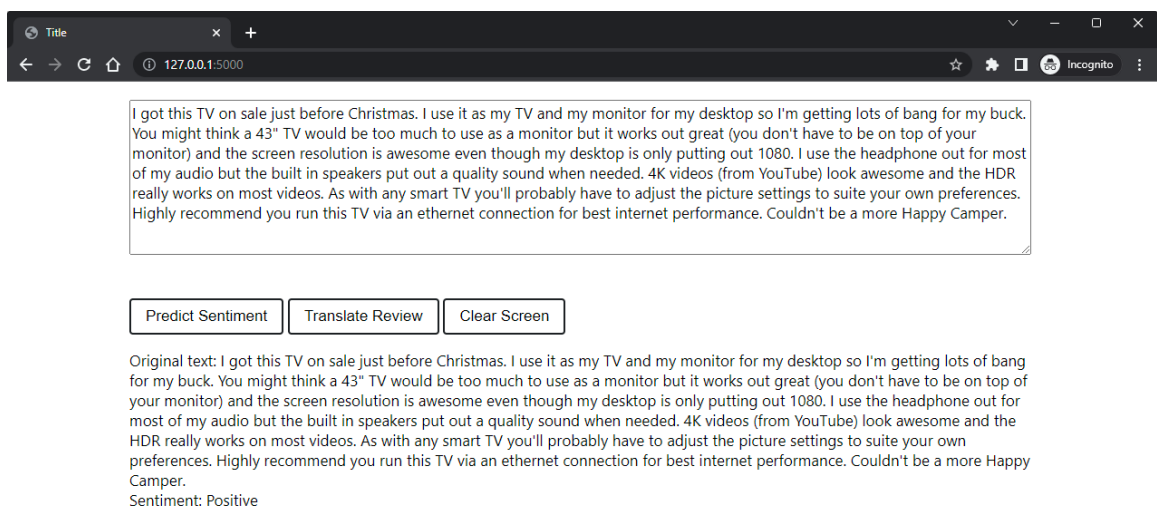


Figure 6. Sentiment Prediction Output from User Input

In Figure 5 we see that once the user has entered some text and clicked the Predict Sentiment button, it will display the appropriate sentiment. The

Predict Sentiment button call the predict_sentiment() function and take the input text from the user and predict the sentiment. In this example, the sentiment is displayed as Positive.

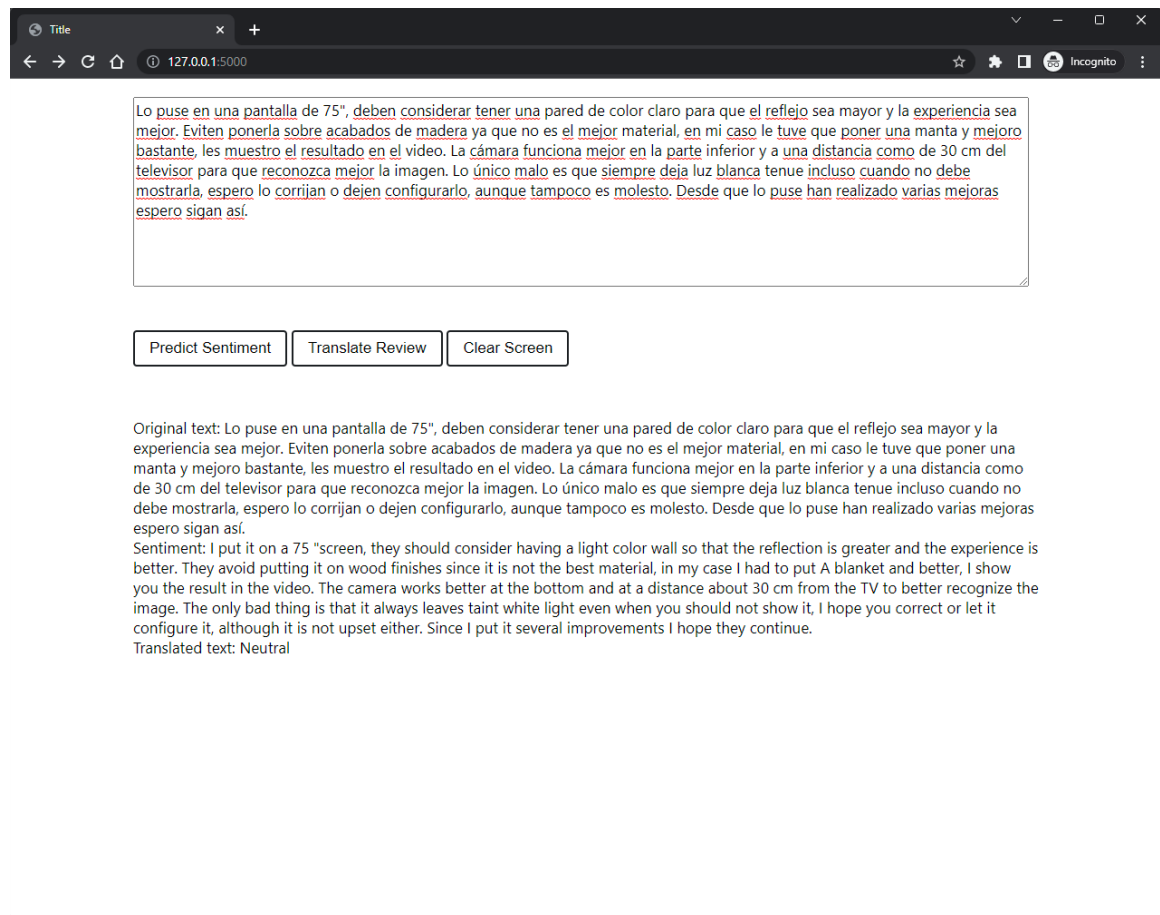


Figure 7. Translated Sentiment Prediction Output from User Input

In Figure 6 we see that once the user has entered some text and then clicked on the Translate Review button, it will display the original text that the

user entered, the text translated back to English, and then display the sentiment. This is done by calling the `translate()` method of the `iTranslate` Python library, and then move the translated text to the `predict_sentiment()` function to predict the sentiment.

CHAPTER EIGHT

CONCLUSION

We have classified the reviews using three approaches – Opinion Lexicon, TextBlob, and Neural Network Training. We see an accuracy of 64.38% when we classify the reviews with Opinion Lexicon, and we see an accuracy of 65.71% when we use TextBlob polarity scores to classify the reviews. We have also created a neural network model that learns from the review star classification method and achieves an accuracy of 96.46%. We have created an application that will successfully analyze reviews left by customers in the future and be able to classify those reviews as positive, negative, or neutral based on the neural network model.

Here are a few enhancements that could be made to the project:

- Implement methods for users to provide feedback on sentiment predictions, allowing the model to be trained continuously over time.
- Explore more advanced deep learning models, or transformer-based models like BERT, to capture more intricate relationships and context within the text.

As for the frontend of the project, certain enhancements come to mind that could be made:

- A simplified process for training the model inside the application instead of using Jupyter Notebook. This is not currently possible in

my project as the training sometimes fails or stops before reaching its' intended accuracy level.

- Alongside training inside the application, the user should also be able to select a custom dataset, either in CSV or JSON formats. This is also not possible in the project's current state for the same reason above, but it is possible if you manually specify in the ipynb notebook which CSV file the application should use in the "read_csv" line.
- The user should also be able to select custom models if they are training the model with newer datasets. This is also not possible in the project's current state for the same reason above, but it is possible if you manually specify in the code which model file the application should use in the "load_model()" method.

APPENDIX A:

CODE

Importing Libraries

```
import nltk
from nltk.tokenize import TreebankWordTokenizer
from nltk.tokenize import sent_tokenize
from nltk.corpus import stopwords, opinion_lexicon

from ipywidgets import interact, interact_manual, fixed
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, precision_score, recall_score, f1_score
from textblob import TextBlob
nltk.download('stopwords')
nltk.download('opinion_lexicon')

import string
from string import punctuation
import contractions

from tqdm import tqdm
tqdm.pandas()

pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)

from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.layers import Dense, Dropout, Embedding, LSTM, Bidirectional, BatchNormalization, Masking, Activation, SimpleRNN
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import Tokenizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.utils import np_utils
import tensorflow
import warnings
warnings.filterwarnings('ignore')
```

Loading the Dataset into a Pandas Dataframe

```
#read the data

reviews = pd.read_csv("bigdata.csv", low_memory=False)

#keep only targeted columns
col = ['overall', 'reviewText']
df = reviews[col].copy()

df.dropna(inplace=True)
```

Preprocessing the Data

```
stopwords = set(stopwords.words('english'))

def remove_punctuation(text):
    return text.translate(str.maketrans('', '', punctuation))

def remove_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in stopwords if not word.isdigit()])

#text cleaning function
def remove_noise(text):

    # Make Lowercase
    text = text.apply(lambda x: " ".join(x.lower() for x in x.split()))

    #word contractions
    text = text.apply(lambda x: " ".join([contractions.fix(i) for i in x.split()]))

    #remove punctuation
    text = text.apply(lambda x: " ".join(remove_punctuation(x) for x in x.split()))

    #remove stopwords
    text = text.apply(lambda x: " ".join(remove_stopwords(x) for x in x.split()))

    # Remove whitespaces
    text = text.apply(lambda x: " ".join(x.strip() for x in x.split()))

    # Convert to string
    text = text.astype(str)

    return text

#performing text cleaning
df['clean_review'] = df[['reviewText']].progress_apply(remove_noise,axis=1)
```

Categorize Reviews According to Review Stars

```
#categorized sentiment according to stars:
df['true_sentiment'] = df['overall'].apply(
    lambda x: "positive" if x >= 4 else ("neutral" if x == 3 else "negative"))
y_true = df['true_sentiment'].tolist()
```

Categorize Reviews According to Opinion Lexicon

```
pos_words = list(opinion_lexicon.positive())
neg_words = list(opinion_lexicon.negative())
```

```
#calculate sentiment score for each text against Treebank tokenizer
def get_sentiment_score(text):
    total_score = 0
    raw_sentences = sent_tokenize(text)

    for sentence in raw_sentences:
        sent_score = 0
        tokens = TreebankWordTokenizer().tokenize(text)

        for token in tokens:
            if token in pos_words:
                sent_score += 1
            elif token in neg_words:
                sent_score -= 1

        if len(tokens) > 0:
            sentence_score = sent_score / len(tokens)
            total_score += sentence_score

    return total_score
```

```
df['sentiment_score'] = df['clean_review'].apply(lambda x: get_sentiment_score(x))
```

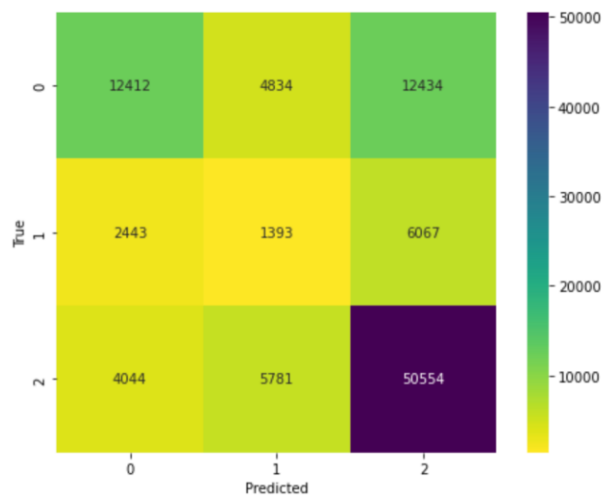
```
#categorized sentiment according to oplex score:
df['oplex_sentiment'] = df['sentiment_score'].apply(
    lambda x: "positive" if x > 0 else ("negative" if x < 0 else "neutral"))
```

Plotting the Confusion Matrix of Opinion Lexicon classified reviews and Star classification reviews

```
y_oplex_pred = df['oplex_sentiment'].tolist()

oplex_cm = confusion_matrix(y_true, y_oplex_pred)

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8,6))
sns.heatmap(oplex_cm, cmap='viridis_r', annot=True, fmt='d', square=True, ax=ax)
ax.set_xlabel('Predicted')
ax.set_ylabel('True');
```



Evaluating Performance metrics of Opinion Lexicon predictions

```
target_names = ['class 0', 'class 1', 'class 2']
accuracy = accuracy_score(y_true, y_oplex_pred)
precision = precision_score(y_true, y_oplex_pred, average='weighted')
recall = recall_score(y_true, y_oplex_pred, average='weighted')
f1 = f1_score(y_true, y_oplex_pred, average='weighted')
cr = classification_report(y_true, y_oplex_pred, target_names=target_names)
```

```
print('Accuracy: ', accuracy)
print('Precision: ', precision)
print('Recall: ', recall)
print('F-1 score: ', f1)
print('Classification Report: \n', cr)
```

```
Accuracy: 0.6438346571697245
Precision: 0.6486837257679936
Recall: 0.6438346571697245
F-1 score: 0.6361525782912014
Classification Report:
              precision    recall  f1-score   support

   class 0       0.66       0.42       0.51       29680
   class 1       0.12       0.14       0.13        9903
   class 2       0.73       0.84       0.78       60379

 accuracy          0.64       0.64       0.64       99962
  macro avg       0.50       0.47       0.47       99962
 weighted avg       0.65       0.64       0.64       99962
```

Categorize Reviews According to TextBlob

```
def calculate_sentiment_label(review):  
    blob = TextBlob(str(review))  
    sentiment = blob.sentiment.polarity  
    return sentiment
```

```
df['sentiment_polarity'] = df['reviewText'].apply(calculate_sentiment_label)
```

```
df['blob_sentiment'] = df['sentiment_polarity'].apply(  
    lambda x: "positive" if x > 0 else ("negative" if x < 0 else "neutral"))
```

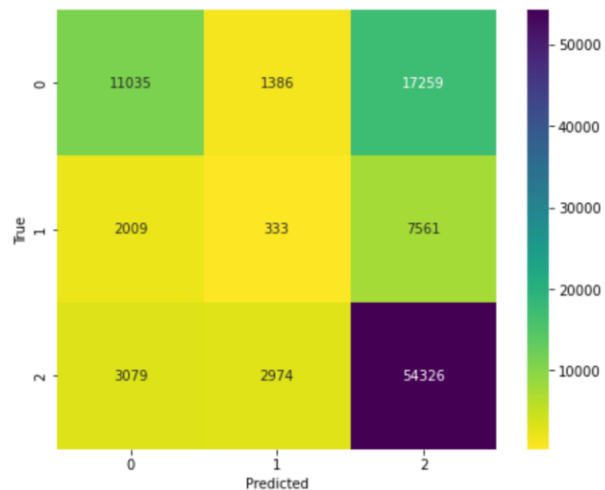
Plotting the Confusion Matrix of TextBlob classified reviews and Star

classification reviews

```
y_blob_pred = df['blob_sentiment'].tolist()
```

```
oplex_cm2 = confusion_matrix(y_true, y_blob_pred)
```

```
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8,6))  
sns.heatmap(oplex_cm2, cmap='viridis_r', annot=True, fmt='d', square=True, ax=ax)  
ax.set_xlabel('Predicted')  
ax.set_ylabel('True');
```



Evaluating the Performance metrics of TextBlob Classification

```
target_names = ['class 0', 'class 1', 'class 2']
accuracy = accuracy_score(y_true, y_blob_pred)
precision = precision_score(y_true, y_blob_pred, average='weighted')
recall = recall_score(y_true, y_blob_pred, average='weighted')
f1 = f1_score(y_true, y_blob_pred, average='weighted')
cr = classification_report(y_true, y_blob_pred, target_names=target_names)
```

```
print('Accuracy: ', accuracy)
print('Precision: ', precision)
print('Recall: ', recall)
print('F-1 score: ', f1)
print('Classification Report: \n', cr)
```

```
Accuracy: 0.6571897320981973
Precision: 0.624844788767698
Recall: 0.6571897320981973
F-1 score: 0.6179534730365019
Classification Report:
              precision    recall  f1-score   support

   class 0         0.68       0.37       0.48       29680
   class 1         0.07       0.03       0.05        9903
   class 2         0.69       0.90       0.78       60379

 accuracy                   0.66       99962
 macro avg              0.48       0.44       0.44       99962
 weighted avg           0.62       0.66       0.62       99962
```

Concatenating the reviews into a new “Final” Pandas Dataframe

```
r1 = df[df['true_sentiment']=='positive']
r2 = df[df['true_sentiment']=='negative']
r3 = df[df['true_sentiment']=='neutral']

final = pd.concat([r1,r2,r3])
```

Encoding the Review star Sentiment label columns

```
#encoding the target sentiments
lb = LabelEncoder()
final['encoded'] = lb.fit_transform(final['true_sentiment'])
```

Saving the reviews into a new CSV file

```
final.to_csv("new_data.csv",index=False)
```

Reading the saved dataset and verifying the value counts

```
df=pd.read_csv("new_data.csv")
df['encoded'].value_counts()
```

```
2    60379
0    29680
1     9903
Name: encoded, dtype: int64
```

```
final['encoded'].value_counts()
```

```
2    60379
0    29680
1     9903
Name: encoded, dtype: int64
```

```
final['true_sentiment'].value_counts()
```

```
positive    60379
negative    29680
neutral      9903
Name: true_sentiment, dtype: int64
```

Setting up tokenizers and dictionaries

```
features = final['clean_review']
sentiment = final['encoded']

#Set up tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(features)
word_index = tokenizer.word_index
```

Saving the tokenizers for future use

```
import pickle

# saving
with open('tokenizer.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

Split the reviews into training sets and test sets

```
#train test split
X_train, x_test, Y_train, y_test = train_test_split(features, sentiment, test_size=0.3, random_state = 45)

assert X_train.shape[0] == Y_train.shape[0]
assert x_test.shape[0] == y_test.shape[0]

print(f"shape of training sentences {(X_train.shape)}\n")
print(f"shapes training labels {(Y_train.shape)}\n")
print(f"shape of esting sentences {(x_test.shape)}\n")
print(f"shapes of testing labels {(y_test.shape)}")
print(" ")

shape of training sentences (69973,)

shapes training labels (69973,)

shape of esting sentences (29989,)

shapes of testing labels (29989,)
```

Tokenizing, padding and converting reviews into numerical vectors

```
#Function to tokenizing, converting and padding the reviews into numerical vectors
def sequence_padding(sentences, tokenizer, padding, max_sentence_length, truncating):
    sequences = tokenizer.texts_to_sequences(sentences)
    pad_trunc_sequences = sequence.pad_sequences(sequences, maxlen=max_sentence_length, padding=padding, truncating=truncating)
    return pad_trunc_sequences

padding="post"
truncating= "post"
train_set = sequence_padding(X_train, tokenizer, padding, max_sentence_length, truncating=truncating)
test_set = sequence_padding(x_test, tokenizer, padding, max_sentence_length, truncating=truncating)

print(f"Padded training sequences have shape: {train_set.shape}\n")
print(f"Padded testing sequences have shape: {test_set.shape}")

Padded training sequences have shape: (69973, 250)

Padded testing sequences have shape: (29989, 250)
```

Converting the training and test labels to One-hot Encoded Vectors

```
#process the Label data
train_labels = np.array(Y_train)
test_labels = np.array(y_test)

#Convert the Labels to One hot encoding vector for softmax for neural network

num_labels = len(np.unique(test_labels))
Y_oh_train = np_utils.to_categorical(train_labels, num_labels)
y_oh_test = np_utils.to_categorical(test_labels, num_labels)

print(f"Shape for Train Labels: {Y_oh_train.shape}\n")
print(f"Shape for test Labels: {y_oh_test.shape}")

# 0: ['1', '0', '0']
# 1: [0, 1, 0]
# 2: [0, 0, 1]

Shape for Train Labels: (69973, 3)

Shape for test Labels: (29989, 3)
```

Read the Glove Vector file

```
#read glove vector
def read_glove_vector(glove_vec):
    with open(glove_vec, 'r', encoding='UTF-8') as f:
        words = set()
        word_to_vec_map = {}
        for line in f:
            w_line = line.split()
            curr_word = w_line[0]
            word_to_vec_map[curr_word] = np.array(w_line[1:], dtype=np.float64)
        return word_to_vec_map

word_to_vec_map = read_glove_vector('glove.6B.100d.txt')
```

Creating the Embedding Matrix

```
# Get the Length of vectors
vocab_size = len(word_index) + 1
embed_vector_len = 100

#create embedding matrix and embedding layer
emb_matrix = np.zeros((vocab_size, embed_vector_len))

for word, index in word_index.items():
    embedding_vector = word_to_vec_map.get(word)
    if embedding_vector is not None:
        emb_matrix[index, :] = embedding_vector
```

Creating the Model

```
def create_model(vocab_size, embed_vector_len, max_sentence_length, emb_matrix):
    model = Sequential()
    model.add(Embedding(input_dim=vocab_size, output_dim=embed_vector_len, weights = [emb_matrix],
                        input_length=max_sentence_length, trainable=False))
    model.add(LSTM(128, return_sequences=False))
    model.add(Dense(32, input_shape=Y Oh_train.shape, activation='sigmoid'))
    model.add(Dense(32, activation='sigmoid'))
    model.add(Dense(3, activation='sigmoid'))
    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    return model
```

```
model = create_model(vocab_size, embed_vector_len, max_sentence_length, emb_matrix)
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 250, 100)	4482800
lstm (LSTM)	(None, 128)	117248
dense (Dense)	(None, 32)	4128
dense_1 (Dense)	(None, 32)	1056
dense_2 (Dense)	(None, 3)	99

=====
Total params: 4,605,331
Trainable params: 122,531
Non-trainable params: 4,482,800
=====

Fitting the Model

```
# Create callbacks
callbacks = [EarlyStopping(monitor='val_loss', patience=5),
             ModelCheckpoint('model.h5', save_best_only=True,
                             save_weights_only=False)]
```

```
history = model.fit(train_set, Y_oh_train, epochs=30, batch_size=128, verbose=1, validation_split=0.2, callbacks=callbacks)
```

```
Epoch 1/30
438/438 [=====] - 12s 23ms/step - loss: 0.9051 - accuracy: 0.5951 - val_loss: 0.8949 - val_accuracy: 0.6066
Epoch 2/30
438/438 [=====] - 10s 22ms/step - loss: 0.8951 - accuracy: 0.6023 - val_loss: 0.8956 - val_accuracy: 0.6066
Epoch 3/30
438/438 [=====] - 10s 22ms/step - loss: 0.8952 - accuracy: 0.6022 - val_loss: 0.8963 - val_accuracy: 0.6066
Epoch 4/30
438/438 [=====] - 10s 22ms/step - loss: 0.8950 - accuracy: 0.6023 - val_loss: 0.8955 - val_accuracy: 0.6066
Epoch 5/30
438/438 [=====] - 10s 23ms/step - loss: 0.8943 - accuracy: 0.6029 - val_loss: 0.8952 - val_accuracy: 0.6069
Epoch 6/30
438/438 [=====] - 10s 23ms/step - loss: 0.8935 - accuracy: 0.6040 - val_loss: 0.8933 - val_accuracy: 0.6081
Epoch 7/30
438/438 [=====] - 10s 24ms/step - loss: 0.8931 - accuracy: 0.6042 - val_loss: 0.8910 - val_accuracy: 0.6066
Epoch 8/30
438/438 [=====] - 11s 24ms/step - loss: 0.8905 - accuracy: 0.6082 - val_loss: 0.8936 - val_accuracy: 0.6089
Epoch 9/30
438/438 [=====] - 11s 24ms/step - loss: 0.8944 - accuracy: 0.6033 - val_loss: 0.8936 - val_accuracy: 0.6067
Epoch 10/30
438/438 [=====] - 11s 24ms/step - loss: 0.8929 - accuracy: 0.6035 - val_loss: 0.8924 - val_accuracy: 0.6085
Epoch 11/30
438/438 [=====] - 11s 25ms/step - loss: 0.8898 - accuracy: 0.6044 - val_loss: 0.9190 - val_accuracy: 0.6088
Epoch 12/30
438/438 [=====] - 11s 24ms/step - loss: 0.6989 - accuracy: 0.7294 - val_loss: 0.5954 - val_accuracy: 0.7800
Epoch 13/30
438/438 [=====] - 11s 25ms/step - loss: 0.5373 - accuracy: 0.8090 - val_loss: 0.5424 - val_accuracy: 0.7936
Epoch 14/30
438/438 [=====] - 11s 25ms/step - loss: 0.4458 - accuracy: 0.8418 - val_loss: 0.4280 - val_accuracy: 0.8462
Epoch 15/30
438/438 [=====] - 11s 25ms/step - loss: 0.3656 - accuracy: 0.8713 - val_loss: 0.3745 - val_accuracy: 0.8712
Epoch 16/30
438/438 [=====] - 11s 25ms/step - loss: 0.2987 - accuracy: 0.9023 - val_loss: 0.3237 - val_accuracy: 0.8975
Epoch 17/30
438/438 [=====] - 11s 26ms/step - loss: 0.2458 - accuracy: 0.9278 - val_loss: 0.3099 - val_accuracy: 0.9052
Epoch 18/30
438/438 [=====] - 11s 25ms/step - loss: 0.2113 - accuracy: 0.9420 - val_loss: 0.2903 - val_accuracy: 0.9170
Epoch 19/30
438/438 [=====] - 11s 25ms/step - loss: 0.1855 - accuracy: 0.9532 - val_loss: 0.2781 - val_accuracy: 0.9221
Epoch 20/30
```

Evaluating the model with the Test Set, and saving the Model

```
model.evaluate(test_set, y_oh_test)
model.save('final_model.h5')
```

```
938/938 [=====] - 10s 10ms/step - loss: 0.2051 - accuracy: 0.9547
```

Plotting the Confusion Matrix of the Neural Network Predictions and True Training Labels

```
y_pred = model.predict(train_set)

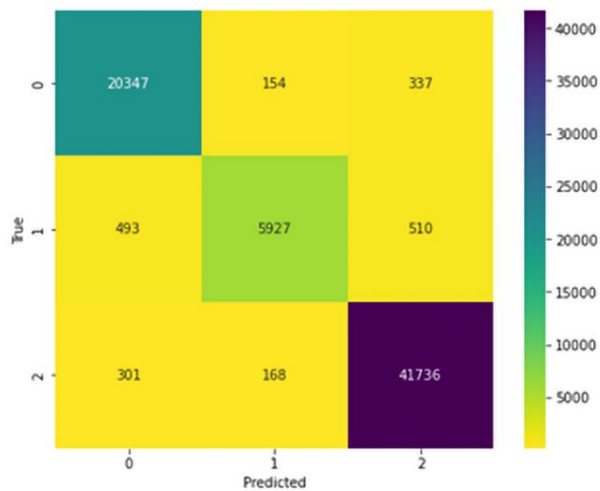
2187/2187 [=====] - 18s 8ms/step

prediction_labels = np.argmax(y_pred, axis=1)

y_train_labels = np.argmax(Y_train, axis=1) if Y_train.ndim > 1 else Y_train

neural_cm = confusion_matrix(y_train_labels, prediction_labels)

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(8,6))
sns.heatmap(neural_cm, cmap='viridis_r', annot=True, fmt='d', square=True, ax=ax)
ax.set_xlabel('Predicted')
ax.set_ylabel('True');
```



Evaluating the Performance metrics of Neural Network Predictions

```
target_names = ['class 0', 'class 1', 'class 2']
accuracy = accuracy_score(y_train_labels, prediction_labels)
precision = precision_score(y_train_labels, prediction_labels, average='weighted')
recall = recall_score(y_train_labels, prediction_labels, average='weighted')
f1 = f1_score(y_train_labels, prediction_labels, average='weighted')
cr = classification_report(y_train_labels, prediction_labels, target_names=target_names)
```

```
print('Accuracy: ', accuracy)
print('Precision: ', precision)
print('Recall: ', recall)
print('F-1 score: ', f1)
print('Classification Report: \n', cr)
```

```
Accuracy: 0.9646720878052963
Precision: 0.9641261382649658
Recall: 0.9646720878052963
F-1 score: 0.9643046814275554
Classification Report:
```

	precision	recall	f1-score	support
class 0	0.96	0.96	0.96	20838
class 1	0.88	0.83	0.86	6930
class 2	0.98	0.99	0.98	42205
accuracy			0.96	69973
macro avg	0.94	0.93	0.93	69973
weighted avg	0.96	0.96	0.96	69973

REFERENCES

- [1] McMahan, B., & Rao, D. (2019). "Natural Language Processing with PyTorch: Build Intelligent Language Applications Using Deep Learning."
- [2] Pak, A., & Paroubek, P. (2010). "Twitter as a Corpus for Sentiment Analysis and Opinion Mining." *Proceedings of LREC*, 10.
- [3] Mohammad, S.M., Salameh, M., & Kiritchenko, S. (2016). "Sentiment Lexicons for Arabic Social Media." *LREC*.
- [4] S. Park and Y. Kim. (2016). "Building Thesaurus Lexicon using Dictionary Based Approach for Sentiment Classification." *IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)*, 39-44
- [5] Bautin, M., Vijayarenu, L., & Skiena, S. (2021). "International Sentiment Analysis for News and Blogs." *Proceedings of the International AAAI Conference on Web and Social Media*, 2(1), 19-26.
<https://doi.org/10.1609/icwsm.v2i1.18606>
- [6] Muhammad, S. H., Adelani, D. I., Ruder, S., Ahmad, I. S., Abdulmumin, I., Bello, B. S., Choudhury, M., Emezue, C. C., Abdullahi, S. S., Aremu, A., Jeorge, A., & Brazdil, P. (2022). "NaijaSenti: A Nigerian Twitter Sentiment Corpus for Multilingual Sentiment Analysis."
<https://doi.org/10.48550/arXiv.2201.08277>
- [7] Yan H., Dai, J., Ji, T., Qiu, X., & Zheng, Z. (2021). "A Unified Generative Framework for Aspect-based Sentiment Analysis." *In Proceedings of the*

- 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Association for Computational Linguistics, 2416–2429, Online. <https://doi.org/10.18653/v1/2021.acl-long.188>
- [8] Singh, M., Jakhar, A.K. & Pandey, S. (2021). “Sentiment analysis on the impact of coronavirus in social life using the BERT model.” *Soc. Netw. Anal. Min*, 11, 33. <https://doi.org/10.1007/s13278-021-00737-z>.
- [9] Lyu, J. C., Han, E. L., & Luli, G. K. (2021). “COVID-19 Vaccine-Related Discussion on Twitter: Topic Modeling and Sentiment Analysis.” *Journal of medical Internet research*, 23(6), e24435, <https://doi.org/10.2196/24435>
- [10] Rustam, F., Khalid, M., Aslam, W., Rupapara, V., Mehmood, A., & Choi, G.S. (2021). “A performance comparison of supervised machine learning models for Covid-19 tweets sentiment analysis.” *PLoS ONE* 16(2): e0245909, <https://doi.org/10.1371/journal.pone.0245909>
- [11] Nandwani, P., & Verma, R. (2021). “A review on sentiment analysis and emotion detection from text.” *Soc. Netw. Anal. Min*, 11, 81. <https://doi.org/10.1007/s13278-021-00776-6>
- [12] Sherstinsky, A. (2020). “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network.” <https://doi.org/10.48550/arXiv.1808.03314>
- [13] Hu, M., & Liu, M. (2004). “Mining and Summarizing Customer Reviews.” *In Proceedings of the tenth ACM SIGKDD international conference on*

Knowledge discovery and data mining (KDD '04). Association for Computing Machinery, New York, NY, USA, 168–177.

<https://doi.org/10.1145/1014052.1014073>

[14] TextBlob: Simplified Text Processing: <https://textblob.readthedocs.io/en/dev/>

[15] TensorFlow API Guide: https://www.tensorflow.org/api_docs

[16] Keras API Guide: <https://keras.io/about/>

[17] Getting Started – scikit-learn : https://scikit-learn.org/stable/getting_started.html

[18] Glove – Getting Started: <https://nlp.stanford.edu/projects/glove>

[19] Pennington, J., Socher, R., & Manning, C.D. (2014). "Glove: Global vectors for word representation." *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, Association for Computational Linguistics, Doha, Qatar, 1532–1543.
<https://doi.org/10.3115/v1/D14-1162>

[20] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). "Distributed Representations of Words and Phrases and their Compositionality." <https://doi.org/10.48550/arXiv.1310.4546>

[21] Word2Vec – Wikipedia: <https://en.wikipedia.org/wiki/Word2vec>

[22] Mathworks: Long Short-term Memory (LSTM) – What is LSTM?
<https://www.mathworks.com/discovery/lstm.html>

[23] Ajagekar A. (2014). Adam - Cornell University Computational Optimization

Open Textbook - Optimization Wiki,

<https://optimization.cbe.cornell.edu/index.php?title=Adam>

[24] Kingma, D.P., & Ba, J. (2014). "Adam: A method for stochastic optimization."

<https://doi.org/10.48550/arXiv.1412.6980>

[25] Flask Frontend Guide: <https://flask.palletsprojects.com/en/>