Natural Language Processing Report

Mostafa Sherif and Mahmoud Moamen

May 19, 2024

Abstract

This report presents an in-depth analysis of Amazon's 500 Bestsellers in Laptop Gear dataset for the year 2024, focusing on extracting valuable insights into consumer preferences, sentiments, and market trends within the laptop accessory market. The study begins with an exploration of recent literature, encompassing a comprehensive review of existing research and methodologies employed in understanding similar e-commerce datasets. Following this, the report delves into the data analysis phase, where various statistical techniques and visualization methods are applied to uncover patterns, correlations, and key drivers influencing consumer behavior and product popularity. The report also presents actionable insights for businesses and stakeholders, shedding light on strategic opportunities, challenges, and potential areas for improvement within the laptop gear market. Additionally, the report discusses the limitations inherent in the dataset and analysis methodology, providing a balanced perspective on the findings presented. Overall, this report serves as a valuable resource for decision-makers seeking to navigate the dynamic landscape of e-commerce.

1 Introduction

In the contemporary marketplace, the role of data-driven decision-making has become increasingly pivotal for businesses striving to maintain competitiveness and meet evolving consumer demands. Amidst the burgeoning landscape of e-commerce platforms, Amazon stands as a cornerstone, offering a vast array of products ranging from essential electronics to niche accessories. Within this spectrum, the realm of laptop gear emerges as a focal point, reflecting both utility and personal expression in the digital age.

1.1 Motivation

The motivation behind this analysis lies in the recognition of the immense value encapsulated within consumer data, particularly within the context of Amazon's Bestsellers in Laptop Gear dataset for the year 2024. By harnessing the insights embedded within this dataset, we embark on a journey to unearth the underlying trends, sentiments, and preferences of consumers towards an assortment of laptop accessories and peripherals.

At the heart of this endeavor lies the pursuit of actionable intelligence - the extraction of insights that can steer strategic decision-making, foster product innovation, and cultivate a customer-centric approach within the competitive e-commerce landscape. Understanding the nuanced interplay between consumer sentiment, product attributes, and market dynamics holds the key to unlocking untapped opportunities, fostering brand loyalty, and driving sustainable growth in an ever-evolving digital ecosystem.

Through analysis and interpretation, we endeavor to empower businesses and stakeholders with the tools and knowledge needed to navigate the complex terrain of the laptop accessory market with confidence and foresight.

2 Literature Review

This section will discuss three main points were taken as criteria for searching which are Sentiment Analysis for E-Commerce Products using Natural Language Processing, Sentiment Analysis for Customers' Reviews and Dataset Research on E-Commerce products which shows the recent work done for utilizing natural language processing in the E-Commerce specially the sentiment analysis that would help in the upcoming milestones related for the Amazon dataset.

2.1 Sentiment Analysis for E-Commerce Products using Natural Language Processing

The paper titled "Natural Language Processing for Sentiment Analysis in E-Commerce Products" (1) explores the application of sentiment analysis in evaluating consumer sentiments towards products and services, particularly within the e-commerce domain. In recent years, the growth of e-commerce has been exponential, with more consumers turning to online platforms for their shopping needs. This trend underscores the importance of understanding customer opinions and preferences for enhancing online businesses. Sentiment analysis, a branch of natural language processing (NLP), offers a valuable tool for extracting insights from textual data, such as customer reviews and social media posts.

The authors of the paper propose the utilization of PySpark and Spark NLP to address challenges related to real-time data collection and analysis. Traditional sentiment analysis methods often struggle with the large volumes of data generated by e-commerce platforms. PySpark and Spark NLP provide scalable solutions for processing such data efficiently, enabling businesses to derive actionable insights in a timely manner.

In the introduction, the authors underscore the growing significance of online shopping and the pivotal role sentiment analysis plays in comprehending consumer perspectives. They highlight the multifaceted nature of consumer feedback, which encompasses textual reviews, star ratings, and emoticons. Sentiment analysis techniques aim to categorize this diverse range of data into meaningful insights that businesses can use to improve their products and services.

The paper delves into previous studies on sentiment analysis, highlighting diverse techniques and methodologies employed in prior research endeavors. Traditional approaches include rule-based systems, machine learning algorithms, and hybrid models combining both. While these methods have shown promise, they often struggle with the nuances of human language and context, leading to suboptimal results.

The proposed system architecture comprises two primary phases: data collection and real-time analytics. The authors delineate the methodology, encompassing data preprocessing, sentiment sentence extraction, and feature selection through Spark NLP and PySpark. By leveraging these tools, businesses can streamline the sentiment analysis process and extract meaningful insights from large volumes of textual data.

The results and discussion segment present the outcomes of the proposed system, including tokenization, preprocessing, the visualization of common keywords, sentiment classification utilizing WordCloud visualization, and an analysis of frequency distribution based on reviewer age and sentiment scores. These analyses provide valuable insights into consumer preferences and sentiments, enabling businesses to tailor their offerings to better meet customer needs.

In conclusion, the authors summarize the key findings of their study and emphasize the efficacy of the Spark NLP technique in sentiment analysis. They suggest future research avenues, such as investigating deep learning approaches for sentiment analysis and relationship analysis. The growing availability of data and advances in NLP technology offer exciting opportunities for further research in this field.

2.2 Sentiment Analysis for Customers' Reviews

The paper titled "Sentiment Analysis of E-commerce Customer Reviews Based on Natural Language Processing" (2) by Xiaoxin Lin explores the application of machine learning algorithms to analyze customer sentiment in e-commerce clothing reviews. As the e-commerce landscape continues to evolve, businesses are increasingly turning to customer reviews to gauge product satisfaction and identify areas for improvement. Sentiment analysis techniques offer a systematic approach for analyzing these reviews and extracting valuable insights.

The study aims to understand the correlation between review features and product recommendations using natural language processing (NLP). Five popular machine learning algorithms, namely Logistic Regression, Support Vector Machine (SVM), Random Forest, XGBoost, and LightGBM, are employed for sentiment analysis. These algorithms offer different trade-offs in terms of computational complexity, model interpretability, and predictive performance.

The research is based on a dataset obtained from Kaggle consisting of Women's E-Commerce Clothing Reviews, with various features such as title, rating, recommendation indicator, and positive feedback count. The text data undergoes vectorization using the TF-IDF algorithm to enable machine learning analysis. TF-IDF, short for Term Frequency-Inverse Document Frequency, is a common technique for converting textual data into numerical representations suitable for machine learning algorithms.

The paper discusses each algorithm in detail, explaining their theoretical foundations, parameter settings, and optimization techniques. Logistic Regression, SVM, Random Forest, XGBoost, and LightGBM are evaluated based on metrics like accuracy, precision, recall, F1 score, and Area Under Curve (AUC). These metrics provide a comprehensive evaluation of each algorithm's performance across different dimensions.

Results show that LightGBM achieves the best performance among the algorithms, with the highest accuracy and AUC value. Ridge Regression, Linear Kernel SVM, and XGBoost also demonstrate competitive performance. Conversely, SVM with the RBF kernel exhibits the lowest accuracy. Comparison with other studies using similar datasets reveals that the addition of XGBoost and LightGBM algorithms enhances prediction accuracy.

The paper concludes by emphasizing the potential for further refinement of sentiment analysis through advanced NLP techniques and improved model training. It suggests strategies such as refining text preprocessing, distinguishing between rating and recommendation options, and exploring additional machine learning algorithms to enhance accuracy and comprehension of customer sentiment in e-commerce reviews. By leveraging these strategies, businesses can gain deeper insights into customer preferences and enhance the overall shopping experience.

2.3 Dataset Research on E-Commerce products

The paper "Dataset of Natural Language Queries for E-Commerce" (3) introduces a comprehensive dataset containing 3,540 natural language queries pertinent to e-commerce, particularly focusing on product searches for laptops and jackets. In the era of big data, datasets play a crucial role in driving research and innovation across various domains. However, existing datasets often lack detailed natural language information, limiting their utility for specific applications such as product search in e-commerce.

The dataset is obtained through controlled experiments involving participants with varied domain knowledge. It includes annotations for vague terms and key product features, specifically focusing on laptop queries. The dataset is positioned as a valuable resource for advancing research in natural language processing and interactive information retrieval within the domain of product search.

Furthermore, the paper explores various potential applications of the dataset. It discusses how the dataset can facilitate the development of spelling correction models to enhance natural language processing tasks. Additionally, it highlights the dataset's utility in addressing vocabulary mismatch issues by identifying vague expressions in queries. The dataset's annotations also aid in attribute mapping, enabling the matching of unstructured query information with structured product attributes. Moreover, the dataset can be leveraged for product query classification, contributing to the development of more sophisticated algorithms.

In conclusion, the paper emphasizes the dataset's significance in advancing research in natural language processing and e-commerce. It outlines future plans to expand the dataset, including additional annotations, clean versions of queries, and the incorporation of more product domains to enhance the generalizability of models derived from it. By continuously updating and improving the dataset, researchers can ensure its relevance and usefulness in addressing emerging challenges in e-commerce and natural language processing.

2.4 NLP File Fragments classification in didgital forensics

Using NLP techniques for file fragment classification" by Simran Fitzgerald, George Mathews, Colin Morris, and Oles Zhulyn (4), explores the application of natural language processing (NLP) techniques to the classification of file fragments in digital forensics. The paper addresses the challenge of automating the identification of file fragments, particularly in scenarios where files are fragmented due to factors like repeated modifications or limited disk space. While existing literature touches upon machine learning techniques for file fragment classification, this paper proposes a supervised learning approach leveraging support vector machines (SVMs) combined with the bag-of-words model, commonly used in NLP for text classification tasks. Instead of text documents, file fragments are represented as "bags of bytes," with feature vectors incorporating unigram and bigram counts, as well as other statistical measurements like entropy. The authors utilize the Garfinkel data corpus for training and testing, ensuring reproducibility by using publicly available datasets.

The experimental setup involves generating a comprehensive dataset by randomly selecting fragments from the Garfinkel corpus for various file types. This dataset is then used to train and test the SVM classifier. The results demonstrate the effectiveness of the proposed approach, outperforming previous methods in terms of classification accuracy. Notably, the classifier excels in identifying low entropy file fragments, such as plain-text files or uncompressed images, while facing challenges with high entropy fragments like compressed files or binary executables. The paper concludes by suggesting future directions, including optimizing feature selection and exploring boosting techniques to enhance classification performance, especially for high entropy fragments. Overall, the study underscores the potential of leveraging NLP techniques for improving the efficiency and accuracy of file fragment classification in digital forensics.

3 Data Analysis: Transforming, Visualizing, and Providing Insights

To start our analysis, we started by importing essential libraries such as pandas, numpy, matplotlib, seaborn, and nltk. These libraries provided robust tools for data manipulation, visualization, and natural language processing. The dataset was then loaded into a pandas DataFrame, enabling efficient data handling and exploration. The uploaded data was then printed in order to validate that the import was successful.

N.B.: The Dataset provided with a notebook (5) that helped as a reference in some cases

3.1 Data Transformation

Data transformation is a fundamental step in the data analysis process, encompassing various techniques aimed at preparing raw data for analysis and interpretation. This phase involves converting data into a structured format conducive to analytical processing, ensuring its quality, consistency, and suitability for further exploration. Common data transformation techniques include handling missing values, standardizing data formats, normalization, and encoding categorical variables. By employing appropriate data transformation methods, analysts can mitigate data quality issues, enhance the effectiveness of analytical models, and derive actionable insights that drive informed decision-making. Ultimately, data transformation plays a pivotal role in unlocking the full potential of datasets, enabling stakeholders to extract valuable insights and derive tangible business value from their data assets.

3.1.1 Handling Missing Values

An essential aspect of data pre-processing is handling missing values. Through a systematic assessment, the presence of null values in the dataset was identified. This critical step provided insights into the completeness of the dataset and guided subsequent data preprocessing steps to ensure data quality and reliability. To address missing values in key columns such as 'price/value', 'stars', and 'reviewsCount', an imputation technique leveraging the KNNImputer was employed. This technique facilitated the estimation of missing values based on the values of neighboring data points, thereby preserving the integrity of the dataset. Additionally, the 'price/currency' column underwent standardization, enhancing data consistency and facilitating further analysis. Finally, the 'description' column had a lot of missing values. However, after careful studying, it was concluded that the 'title' and 'description' columns were usually very similar. As a result, rather than dropping the empty 'description' rows, a new column 'text' was created by concatenating the 'title' and 'description' columns, facilitating textual analysis.

3.1.2 Normalization and Tokenization

In the process of normalization and tokenization, we standardized and prepared the textual data from the 'text' column for further analysis. This involved several key steps aimed at enhancing the quality and structure of the text data to facilitate computational analysis. Firstly, we employed lemmatization to reduce words to their base or root form, ensuring consistency in the representation of words. Additionally, we removed stopwords and punctuation from the text, eliminating irrelevant or redundant information that could distort analysis results. Finally, as the columns 'title' and 'description' were similar, duplicated words were removed for every row. By standardizing the text data through normalization and tokenization, we created a structured and uniform 'tokens' column suitable for sentiment analysis and textual exploration. This preprocessing step was essential in laying the groundwork for subsequent analytical tasks, enabling us to derive meaningful insights from the textual content of the dataset.

3.1.3 Sentiment Analysis

Leveraging the TextBlob library, sentiment analysis was conducted to measure the polarity of text data. By assigning sentiment scores to each text entry in a 'sentiment_score' and categorizing sentiments as 'Positive', 'Negative', or 'Neutral' into a 'Sentiments_labels' column, valuable insights into the sentiment of the text data assigned to each product were obtained.

3.1.4 Categorizing the Products based on Star Ratings

The categorization of data based on star ratings into 'stars_category' provided further granularity to the analysis. By breaking down reviews into 'Bad Review', 'Average Review', or 'High Review' categories, distinct patterns and trends in product ratings were obtained. This categorization enriched the analysis by offering actionable insights into product performance and would later be used to be compared to the sentiment of the provided text.

3.2 Data Visualization

Visualization served as a powerful tool for conveying insights derived from the dataset. Through various visualization techniques, complex data patterns and relationships were derived. The visualizations facilitated the interpretation of analysis results and can enable stakeholders to grasp key findings at a glance.

3.2.1 Top 5 Brands with Positive/Negative/Neutral Sentiments

This visualization provides insights into sentiment created by different brands. By identifying the brands with the highest count of positive, negative, and neutral sentiments, businesses can understand which brands are resonating positively or negatively with customers. This can inform brand management strategies, product development decisions, and marketing efforts. The tables can be seen in figures 1,2, and 3 respectively.

		text
brand	Sentiments_labels	
Generic	Positive	10
LOVEVOOK	Positive	5
PEHDPVS	Positive	5
SlimQ	Positive	4
Twelve South	Positive	4

Figure 1: Top 5 Brands with Positive Sentiment

		text
brand	Sentiments_labels	
LOVEVOOK	Negative	7
MOSISO	Negative	6
Smatree	Negative	6
AMCJJ	Negative	3
VNINE	Negative	2

Figure 2: Top 5 Brands with Negative Sentiment

brand	Sentiments_labels	text
UGXKNAE	Neutral	5
DGFTB	Neutral	5
Espacio	Neutral	4
ANVMSRO	Neutral	3
dokikalos	Neutral	3

Figure 3: Top 5 Brands with Neutral Sentiment

3.2.2 Categorizing Products based on Sentiment Labels

Figure 4 shows the pie chart that was employed to provide a comprehensive overview of the distribution of sentiment labels across the dataset. This visualization offers insights into the prevalence of positive, negative, and neutral sentiment categories, enabling stakeholders to gauge the overall sentiment landscape. Understanding the distribution of sentiment labels helps in assessing overall customer satisfaction levels and identifying areas for improvement. By pinpointing the prevailing sentiment categories, businesses can prioritize initiatives to address customer concerns, enhance product features, and improve the overall customer experience.

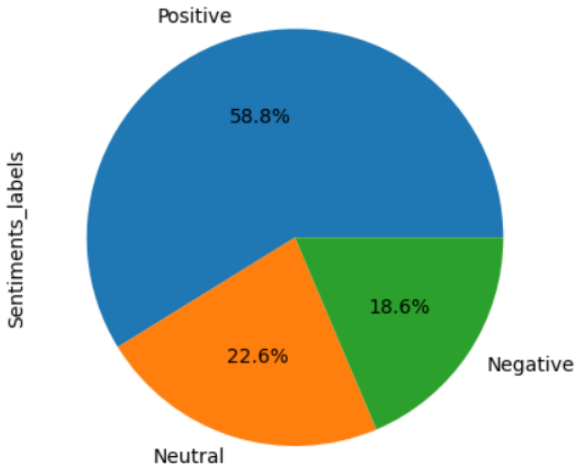

Figure 4: Pie Chart of Sentiment Categorization

3.2.3 Grouping the Products based on their Prices and Sentiment Labels

The grouping of data based on the 'sentiment_labels' and 'price/value' columns facilitated an exploration of the relationship between sentiment and product pricing. This visualization allowed businesses to discern how pricing influences customer sentiment and perception of product value. By analyzing the intersection of sentiment labels and price/value ranges, businesses can identify optimal pricing strategies that resonate well with customers. This insight is invaluable for pricing optimization, competitive positioning, and maximizing profitability in the market.

3.2.4 Displaying the Words based on their Frequency

A word cloud, shown in figure 5, was generated to visually represent the most frequently occurring words in the text data. This visualization offers a quick and intuitive way to identify prominent themes, keywords, and topics within the dataset. By visualizing the most common words, businesses can gain insights into customer preferences, product features, and emerging trends. This aids in understanding the key drivers of customer sentiment and guiding strategic decision-making to address customer needs effectively.

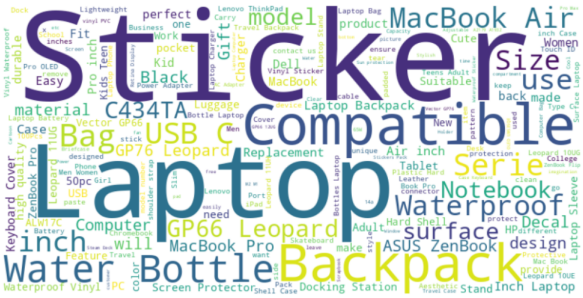

Figure 5: Word Cloud portraying word frequency

3.2.5 Grouping of the Products based on their Rating Categories and Sentiment Labels

The grouping of data based on sentiment labels and star rating categories allowed for an examination of the relationship between sentiment and product ratings. This visualization provided insights into how sentiment aligns with product ratings and whether positive or negative sentiment influences star ratings. By analyzing sentiment trends alongside star ratings, businesses can identify factors driving customer satisfaction and dissatisfaction, informing product improvements and customer service initiatives. These visualizations collectively offer a comprehensive view of customer perceptions and behaviors, empowering businesses to make informed decisions and enhance the overall customer experience. As shown in 6, it was clear that positive sentiment in the text provided with a product improves the chance of a good product rating.

		text
Sentiments_labels	stars_category	
Positive	High Review	268
Neutral	High Review	106
Negative	High Review	84
Positive	Average Review	25
Negative	Average Review	8
Neutral	Average Review	7
Negative	Bad Review	1
Positive	Bad Review	1

Figure 6: Table of products grouped by the sentiment labels and stars rating

4 Implementing a Model from Scratch

4.1 Introduction to LSTMs

In this milestone we aimed to build and implement a new LSTM model from scratch to be chosen for the task the classification task. LSTM stands for Long Short-Term Memory These LSTM networks are a form of recurrent neural network (RNN) architecture developed to overcome the challenges of traditional RNNs, especially in capturing long-term dependencies in sequential data. LSTMs excel in applications where maintaining context over extended sequences is essential in Natural Language Processing.

4.2 Model Architecture

4.2.1 LSTM Cells

LSTM networks are composed of units called LSTM cells, which have a sophisticated structure designed to manage the flow of information effectively. Each LSTM cell has three main gates that control the addition, removal, and update of information.

4.2.2 LSTM Cell gates

The forget gate which decides which information from the previous cell state should be discarded. It measure the importance of the information and selectively removes irrelevant data, helping the network forget unnecessary details, another gate which is the input gate determines what new information should be stored in the cell state. It evaluates the incoming information and updates the cell state with relevant new data. This gate has two parts: one that decides which values will be updated and another that creates new candidate values to be added to the state, and the last one the output gate decides what part of the cell state should be output. It filters the cell state and determines what information should be carried forward as the output of the LSTM cell.

4.3 Implementation Details

4.3.1 Preparing the data

First of all, we append the brand name five times before the tokens and had the new list of tokens begin with brand name five times in a new separate column called

```
"token_with_brnd"
```

that will be used in the training afterwards.

```
def add_brand_to_tokens(row):
  brand_abbr = row['brand'].lower()
  tokens = row['tokens']
  tokens.insert(0, brand_abbr)
  tokens.insert(0, brand_abbr)
  tokens.insert(0, brand_abbr)
  tokens.insert(0, brand_abbr)
  tokens.insert(0, brand_abbr)
  return tokens


df['token_with_brnd'] = df.apply(add_brand_to_tokens, axis=1)
```

Then we categorized the data endpoints into 8 following categories and the categories are defined as follows:

1. **Good Price, Good Sentiment, Good Rating**: This category includes items that have a good price, positive sentiment, and a high rating.
    - Criteria: [True, True, True]

2. **Good Price, Good Sentiment, Bad Rating**: This category includes items that have a good price, positive sentiment, but a low rating.
    - Criteria: [True, True, False]
3. **Good Price, Bad Sentiment, Good Rating**: This category includes items that have a good price, negative sentiment, and a high rating.
    - Criteria: [True, False, True]
4. **Good Price, Bad Sentiment, Bad Rating**: This category includes items that have a good price, negative sentiment, and a low rating.
    - Criteria: [True, False, False]
5. **Bad Price, Good Sentiment, Good Rating**: This category includes items that have a bad price, positive sentiment, and a high rating.
    - Criteria: [False, True, True]
6. **Bad Price, Good Sentiment, Bad Rating**: This category includes items that have a bad price, positive sentiment, and a low rating.
    - Criteria: [False, True, False]
7. **Bad Price, Bad Sentiment, Good Rating**: This category includes items that have a bad price, negative sentiment, and a high rating.
    - Criteria: [False, False, True]
8. **Bad Price, Bad Sentiment, Bad Rating**: This category includes items that have a bad price, negative sentiment, and a low rating.
    - Criteria: [False, False, False]

```
    categories = {
    "Good Price, Good Sentiment, Good Rating": [True, True, True],
    "Good Price, Good Sentiment, Bad Rating": [True, True, False],
    "Good Price, Bad Sentiment, Good Rating": [True, False, True],
    "Good Price, Bad Sentiment, Bad Rating": [True, False, False],
    "Bad Price, Good Sentiment, Good Rating": [False, True, True],
    "Bad Price, Good Sentiment, Bad Rating": [False, True, False],
    "Bad Price, Bad Sentiment, Good Rating": [False, False, True],
    "Bad Price, Bad Sentiment, Bad Rating": [False, False, False]
}
```

then categorized the price is good when it is less than or equal 50 and the sentiments labels is good when the sentiment label is positive or neutral. then we mapped the metrics as a new column in the dataset to the categories.

```
    df['category'] = ''
df['metrics'] = ''


for index, row in df.iterrows():
    mask = [
        (row['price/value'] <= 50),
        (row['Sentiments_labels'] in ["Positive", "Neutral"]),
        (row['stars'] > 2.5)
    ]
    print(mask)
```

```
    for category, metrics in categories.items():
        if mask == metrics:
            df.at[index, 'category'] = category
            break
df['metrics'] = df['category'].map(categories.get)
```

and lastly had a new column called

```
"text_brnd"
```

which is a a string text of brands repeated as a prefix and then the tokens.

4.3.2 Splitting the data for training

In this part, we split the dataset into a train dataset which is 0.8 percent of the original data to be trained and 0.2 percent to be tested and then we mapped each text containing brand in the column with it's corresponding category.

```
    train_dataset, test_dataset = df, df

    for index, row in df.iterrows():
    print("Text_brnd:", row['text_brnd'])
    print("Category:", row['category'])
    print()

BUFFER_SIZE = 10000
BATCH_SIZE = 64

df = pd.DataFrame(df)

dataset = tf.data.Dataset.from_tensor_slices((df['text_brnd'].values, df['category'].values))


VOCAB_SIZE=1000

encoder = TextVectorization(max_tokens=VOCAB_SIZE)

encoder.adapt(dataset.map(lambda text_brnd, category: text_brnd))

vocab_size = len(encoder.get_vocabulary())

print("Vocabulary size:", vocab_size)
```

The provided code cell sets buffer and batch sizes for data processing, converts a pandas DataFrame into a TensorFlow dataset using specified columns for input features and labels, defines a vocabulary size of 1000 tokens, and creates a 'TextVectorization' layer for transforming text data into numerical vectors. The vocabulary is built from the text data in the dataset using the 'adapt' method, and the actual vocabulary size is printed, which indicates the number of unique tokens found in the text data. This process is essential for text preprocessing in natural language processing tasks.

After then the vocabulary got encoded we passed the encoded vocab to word2vec model to get the neural embeddings of the vocab:

```
    word2vec_model = Word2Vec(encoded, min_count=1)

    def get_embeddings(tokens, model):
    embeddings = []
    for token in tokens:
        if token in model.wv:
            embeddings.append(model.wv[token])
```

```
    return np.mean(embeddings, axis=0) if embeddings else np.zeros(model.vector_size)

neural_embeddings = np.array([get_embeddings(tokens, word2vec_model) for tokens in encoded])
```

4.3.3 Training the model

The provided code cell performs the following tasks to prepare, train, and evaluate an LSTM-based model for classification:

1. **Category Mapping**: A dictionary `categories` is created to map textual category descriptions to numerical labels. Each combination of price, sentiment, and rating is assigned a unique integer.
2. **Label Encoding**: The `category` column in the DataFrame `df` is mapped to numerical labels using the `categories` dictionary, and a new column `label` is created.
3. **One-Hot Encoding**: The numerical labels are converted into one-hot encoded vectors using TensorFlow's `to_categorical` function, resulting in `y_one_hot`.
4. **Model Architecture**:

- An input layer is defined to accept sequences of feature vectors.
- Two LSTM layers are added: the first with 64 units and `return_sequences=True`, and the second with 40 units.
- A dense output layer with a softmax activation function is added to produce probability distributions over the eight categories.

5. **Model Compilation**: The model is compiled using the Adam optimizer, categorical cross-entropy loss, and accuracy as a metric.
6. **Train-Test Split**: The data is split into training and testing sets using an 80-20 split.
7. **Model Training**: The model is trained for 50 epochs with a batch size of 32, and 20% of the training data is used for validation.

4.3.4 Model Evaluation

The trained model is evaluated on the test set, and the test loss and accuracy are printed.

```
    categories = {
    "Good Price, Good Sentiment, Good Rating": 0,
    "Good Price, Good Sentiment, Bad Rating": 1,
    "Good Price, Bad Sentiment, Good Rating": 2,
    "Good Price, Bad Sentiment, Bad Rating": 3,
    "Bad Price, Good Sentiment, Good Rating": 4,
    "Bad Price, Good Sentiment, Bad Rating": 5,
    "Bad Price, Bad Sentiment, Good Rating": 6,
    "Bad Price, Bad Sentiment, Bad Rating": 7
}

df['label'] = df['category'].map(categories)

y_one_hot = tf.keras.utils.to_categorical(df['label'], num_classes=len(categories))

num_samples, num_features = encoded.shape
inputs = tf.keras.layers.Input(shape=(None, num_features))

x = LSTM(64, return_sequences=True)(inputs)
x = LSTM(40)(x)

outputs = Dense(len(categories), activation='softmax')(x)

model = Model(inputs, outputs)
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

x_train, x_test, y_train, y_test = train_test_split(neural_embeddings, y_one_hot, test_size=0.2, rand

history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_split=0.2)

test_loss, test_accuracy = model.evaluate(x_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

and finally the model has an accuracy of 0.67 percent.

4.3.5 Predictions

The provided code cell focuses on generating predictions using a trained model and displaying the results in a readable format. Here is a breakdown of what each part does:

1. **Model Predictions**:

```
predictions = model.predict(x_test)
```

The trained model is used to predict the categories for the test data, `x_test`. The predictions are probabilities for each category.

2. **Extracting Predicted Labels**:

```
predicted_labels = np.argmax(predictions, axis=1)
```

The predicted probabilities are converted to discrete category labels by taking the index of the maximum probability for each test sample.

3. **Mapping to Category Names**:

```
predicted_categories = [list(categories.keys())[label] for label in predicted_labels]
```

The numerical labels are mapped back to their corresponding category names using the `categories` dictionary.

4. **Displaying Predictions**:

```
original_texts = df['text_brnd']

print("Predicted Categories for Test Data:")
for i in range(len(x_test)):
    print("Original Text:", original_texts[i])  % Assuming original_texts contains the original text
    print("Test Sample:", x_test[i])
    print("Predicted Category:", predicted_categories[i])
    print()
```

4.4 Results and Analysis

4.4.1 Training and Validation Metrics

The model's performance was tracked over the epochs, with the training and validation loss and accuracy recorded at each epoch.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))
plt.plot(range(1, num_epochs + 1), train_losses, label='Training Loss')
plt.plot(range(1, num_epochs + 1), val_losses, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
```

```
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(range(1, num_epochs + 1), train_accuracies, label='Training Accuracy')
plt.plot(range(1, num_epochs + 1), val_accuracies, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()
```

For each test sample, the original text, the test sample data, and the predicted category are printed. This provides a clear and detailed view of the model's predictions.

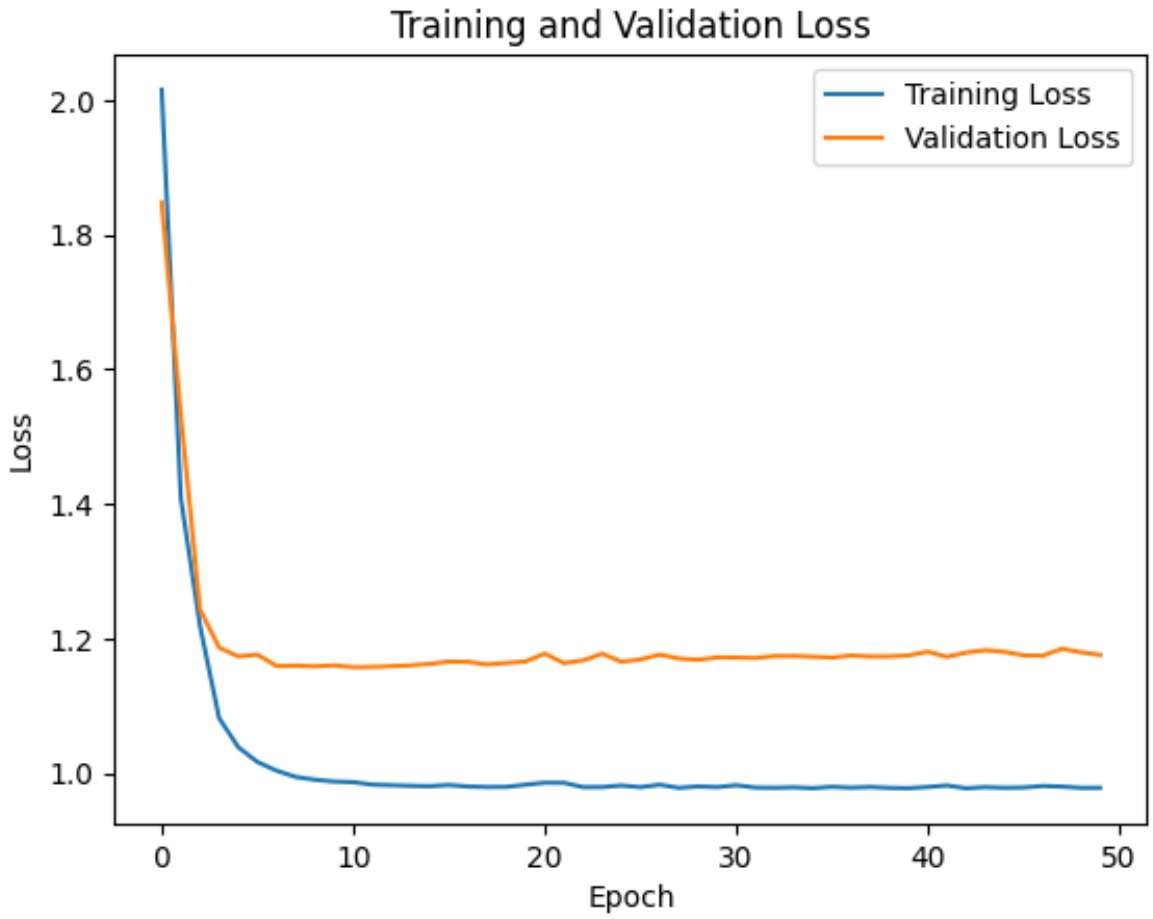

Figure 7: Training and Validation Loss of the LSTM Model

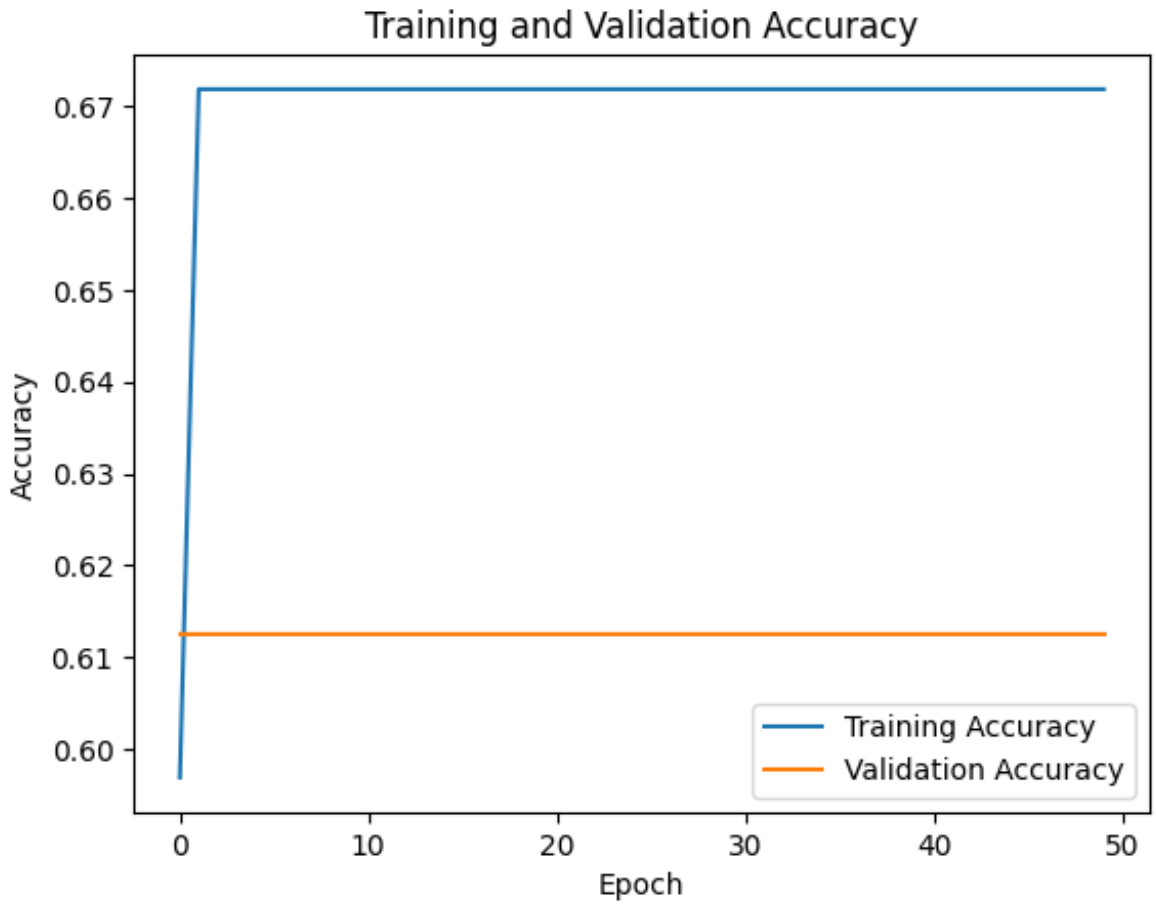

Figure 8: Training and Validation Accuracy of the LSTM Model

5 Using a Pre-Implemented Model

5.1 Introduction to DistilBERT

In this milestone, we focused on utilizing a pre-implemented model to streamline our machine learning workflow. The model chosen for this task is DistilBERT, a smaller, faster, cheaper, and lighter version of BERT (Bidirectional Encoder Representations from Transformers) (6). DistilBERT retains 97% of BERT's language understanding while being 60% faster and 40% smaller. This makes it an ideal choice for projects requiring efficiency without significant sacrifice in performance.

5.2 Model Architecture

DistilBERT is built on the Transformer architecture, which employs a self-attention mechanism to encode input sequences. Unlike the original BERT model, which has 12 layers, DistilBERT reduces this to 6 layers, significantly decreasing the model size and inference time.

5.2.1 Self-Attention Mechanism

The core of DistilBERT's architecture is the self-attention mechanism, which allows the model to weigh the importance of different words in a sentence relative to each other. This mechanism enables the model to capture long-range dependencies and contextual relationships more effectively than traditional RNNs or LSTMs.

5.2.2 Transformer Blocks

Each layer in DistilBERT consists of a multi-head self-attention mechanism and a feed-forward neural network, both followed by layer normalization and residual connections. This architecture helps stabilize the training process and improves the flow of gradients.

5.3 Implementation Details

5.3.1 Loading the Model

To begin with, we loaded the DistilBERT model and its corresponding tokenizer from the Hugging Face Transformers library. The tokenizer converts text into token IDs that the model can process, while the model is initialized for sequence classification tasks. The following code snippet demonstrates this:

```
class DistillBERTClass(torch.nn.Module):
    def __init__(self):
        super(DistillBERTClass, self).__init__()
        self.l1 = DistilBertModel.from_pretrained("distilbert-base-uncased")
        self.pre_classifier = torch.nn.Linear(768, 768)
        self.dropout = torch.nn.Dropout(0.3)
        self.classifier = torch.nn.Linear(768, 8)

    def forward(self, input_ids, attention_mask):
        output_1 = self.l1(input_ids=input_ids, attention_mask=attention_mask)
        hidden_state = output_1[0]
        pooler = hidden_state[:, 0]
        pooler = self.pre_classifier(pooler)
        pooler = torch.nn.ReLU()(pooler)
        pooler = self.dropout(pooler)
        output = self.classifier(pooler)
        return output
```

n this snippet, the `DistillBERTClass` class is defined as a custom PyTorch module. It utilizes the `DistilBertModel` from the Hugging Face Transformers library, specifically the `"distilbert-base-uncased"` variant.

The module incorporates a classification head on top of the DistilBERT model. This head consists of a linear layer `self.pre_classifier`, followed by a ReLU activation function and a dropout layer `self.dropout`. Finally, there's another linear layer `self.classifier` that outputs the classification results.

During the forward pass, input token IDs and attention masks are passed to the DistilBERT model. The output hidden states are processed to obtain a pooled representation, which is then passed through the classification head to produce the final classification output.

5.3.2 Preparing the Data

Our dataset was preprocessed to fit the input requirements of DistilBERT. This involved tokenizing the text data and creating attention masks to inform the model which tokens should be attended to. The `DataCollatorWithPadding` class ensures that all inputs are padded to the same length, which is crucial for batching. Below is the code used for this preprocessing step:

```
class Triage(Dataset):
    def __init__(self, dataframe, tokenizer, max_len):
        self.len = len(dataframe)
        self.data = dataframe
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __getitem__(self, index):
```

```
        title = str(self.data.text_brnd[index])
        priceValue = str(self.data.price[index])
        sentimentLabel = str(self.data.Sentiments_labels[index])
        starCat = str(self.data.stars_category[index])

        title = " ".join(title.split())
        starCat = " ".join(starCat.split())

        combined_input = f"{title} [SEP] {priceValue} [SEP] {sentimentLabel} [SEP] {starCat}"

        inputs = self.tokenizer.encode_plus(
            combined_input,
            add_special_tokens=True,
            max_length=self.max_len,
            padding='max_length',
            return_token_type_ids=True,
            truncation=True
        )
        ids = inputs['input_ids']
        mask = inputs['attention_mask']

        return {
            'ids': torch.tensor(ids, dtype=torch.long),
            'mask': torch.tensor(mask, dtype=torch.long),
            'targets': torch.tensor(self.data.label[index], dtype=torch.long)
        }

    def __len__(self):
        return self.len
```

The constructor `__init__` of the `Triage` class initializes instances with three parameters: `dataframe`, representing the dataset containing input data; `tokenizer`, an instance of a tokenizer utilized to tokenize the input text; and `max_len`, specifying the maximum length of tokenized input sequences.

Within the `__getitem__` method, data is retrieved from the dataset at a specific index. Text data is extracted from the dataframe, preprocessed to remove unnecessary whitespace characters, and combined into a single string using `[SEP]` as a separator to represent the input to the model. The `encode_plus` method of the tokenizer tokenizes the combined input string, adds special tokens, and ensures padding and truncation. Tokenized inputs are then returned as a dictionary containing input ids, attention mask, and the target label.

The `__len__` method returns the length of the dataset, representing the total number of samples in the dataframe.

Overall, the `Triage` class encapsulates functionalities required for data preparation in classification tasks using transformer-based models, similar to how `DistilBertTokenizer` tokenizes input text and `DistilBertForSequenceClassification` initializes DistilBERT with a classification head.

5.3.3 Training the Model

The training process involved defining a training loop where the model was trained over multiple epochs. We used the Adam optimizer and a learning rate scheduler to manage the learning rate dynamically during training. The training loop is detailed below:

```
def train(epoch):
    model.train()
    tr_loss = 0
    n_correct = 0
    nb_tr_steps = 0
```

```
    nb_tr_examples = 0

    for _, data in enumerate(training_loader, 0):
        ids = data['ids'].to(device, dtype=torch.long)
        mask = data['mask'].to(device, dtype=torch.long)
        targets = data['targets'].to(device, dtype=torch.long)

        outputs = model(ids, mask)
        loss = loss_function(outputs, targets)
        tr_loss += loss.item()
        big_val, big_idx = torch.max(outputs.data, dim=1)
        n_correct += (big_idx == targets).sum().item()

        nb_tr_steps += 1
        nb_tr_examples += targets.size(0)

        if _ % 5000 == 0:
            loss_step = tr_loss / nb_tr_steps
            accu_step = (n_correct * 100) / nb_tr_examples
            print(f"Training Loss per 5000 steps: {loss_step}")
            print(f"Training Accuracy per 5000 steps: {accu_step}")

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    epoch_loss = tr_loss / nb_tr_steps
    epoch_accu = (n_correct * 100) / nb_tr_examples
```

In this snippet, the training function begins by setting the model to training mode using `model.train()`. The `AdamW` optimizer is used with a specified learning rate, which is suitable for fine-tuning transformer models. The training loop iterates over each batch in the `training_loader`. For each batch, the input IDs and attention masks are moved to the appropriate device, and the model performs a forward pass to obtain the outputs. The loss is then calculated between the predicted outputs and the true targets using a predefined loss function.

The total loss is accumulated, and the predictions are compared to the true labels to calculate the number of correct predictions. These metrics are used to calculate the average training loss and accuracy for the epoch. Every 5000 steps, the intermediate training loss and accuracy are printed to monitor progress.

Backpropagation is performed by calling `loss.backward()`, and the model parameters are updated using `optimizer.step()`. The optimizer's gradients are zeroed using `optimizer.zero_grad()` to prevent accumulation from previous steps. At the end of the epoch, the average training loss and accuracy are computed and returned.

5.4 Evaluation

5.4.1 Validation and Testing

After training, the model was evaluated on a validation set to monitor its performance and ensure it was not overfitting. The evaluation metrics included accuracy and loss. The testing phase involved using a separate test set to measure the final performance of the model. The evaluation code is shown below:

```
model.eval()
    n_correct = 0; n_wrong = 0; total = 0; tr_loss = 0.0
    nb_tr_steps = 0;
    nb_tr_examples = 0;
    with torch.no_grad():
```

```
    for _, data in enumerate(testing_loader, 0):
        ids = data['ids'].to(device, dtype = torch.long)
        mask = data['mask'].to(device, dtype = torch.long)
        targets = data['targets'].to(device, dtype = torch.long)
        outputs = model(ids, mask).squeeze()
        loss = loss_function(outputs, targets)
        tr_loss += loss.item()
        big_val, big_idx = torch.max(outputs.data, dim=1)
        n_correct += calcuate_accu(big_idx, targets)

        nb_tr_steps += 1
        nb_tr_examples+=targets.size(0)

        if _%5000==0:
            loss_step = tr_loss/nb_tr_steps
            accu_step = (n_correct*100)/nb_tr_examples
            print(f"Validation Loss per 100 steps: {loss_step}")
            print(f"Validation Accuracy per 100 steps: {accu_step}")
epoch_loss = tr_loss/nb_tr_steps
epoch_accu = (n_correct*100)/nb_tr_examples
```

In this snippet, the `model.eval()` method sets the model to evaluation mode, disabling dropout and other training-specific behaviors. The evaluation loop iterates over the test data loader, and for each batch, it performs a forward pass without calculating gradients (using `torch.no_grad()`). In each iteration, the model processes input IDs and attention masks, and computes the loss between the predicted outputs and the true targets using a predefined loss function. The predicted class labels are determined using `torch.max` to find the indices of the highest logit values. The accuracy is calculated by comparing these predicted labels to the true labels. The loss and accuracy are accumulated over all batches to compute the average validation loss and accuracy for the epoch. Intermediate results are printed every 5000 steps to monitor the validation performance during evaluation.

5.5 Results and Analysis

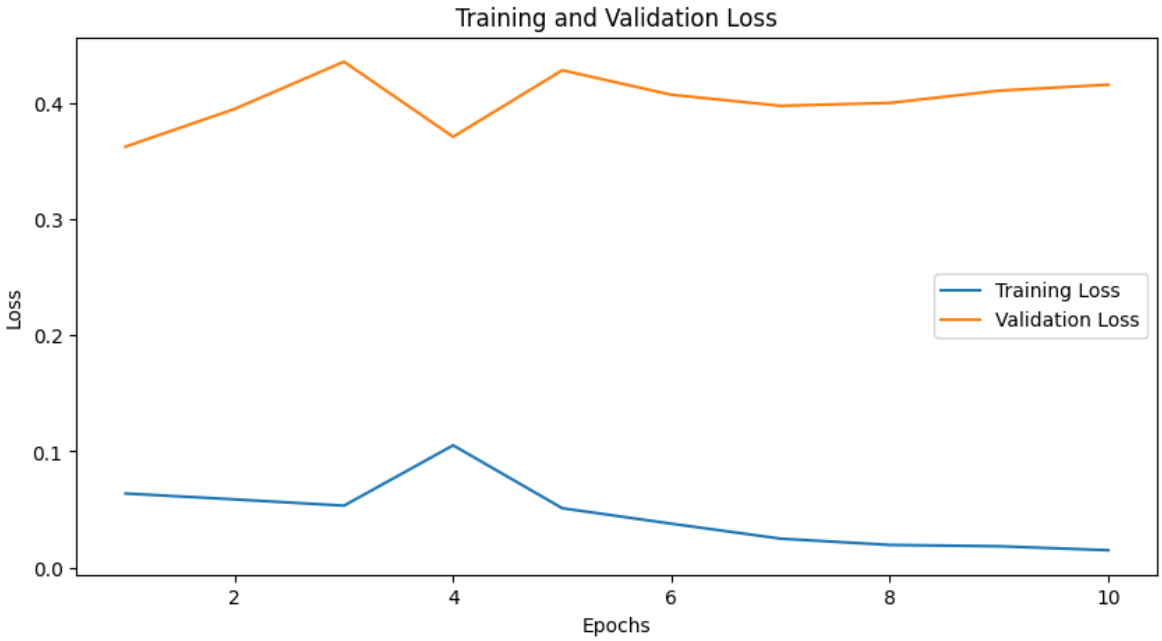

Figure 9: Training and Validation Loss for the DIstilBERT Model

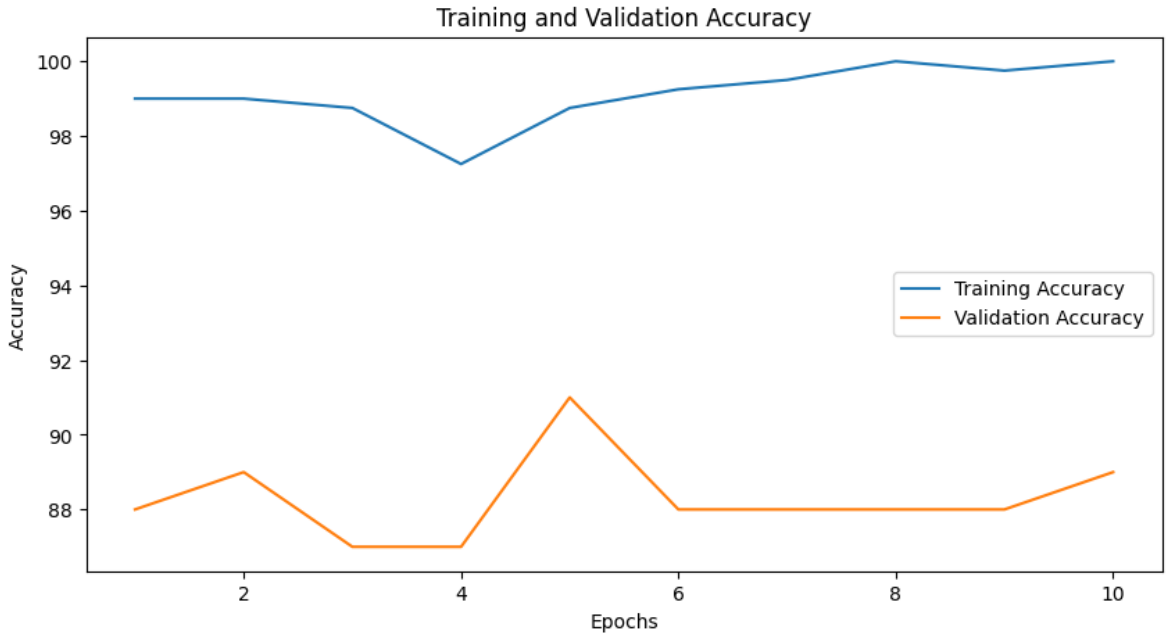

Figure 10: Training and Validation Accuracy for the DIstilBERT Model

The model's performance was tracked over the epochs, with the training and validation loss and accuracy recorded at each epoch. Despite the sophisticated architecture of DistilBERT, the results on our small dataset (500 rows) suggested that the model might be too complex given the limited data.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 5))
plt.plot(range(1, num_epochs + 1), train_losses, label='Training Loss')
plt.plot(range(1, num_epochs + 1), val_losses, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(range(1, num_epochs + 1), train_accuracies, label='Training Accuracy')
plt.plot(range(1, num_epochs + 1), val_accuracies, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()
```

In this code, we plot the training and validation loss over the epochs to observe the model's learning curve. A decreasing training loss with a stable or decreasing validation loss indicates good model performance, while a diverging validation loss suggests overfitting, as shown in Figure 9. We also plot the training and validation accuracy, Figure 10, to compare the model's performance on both datasets. A significant gap between training and validation accuracy can also indicate overfitting.

The training and validation metrics indicated that while the model achieved high training accuracy, there was a noticeable gap in validation accuracy, suggesting overfitting. The training accuracy remains around 99% and the validation accuracy fluctuates between 88% and 91%, as shown in Figure 10. Given the small size of our dataset (500 rows), it is likely that the model complexity of DistilBERT is too high, leading to overfitting and poor generalization to unseen data.

The model achieved high accuracy on the training set, indicating that it learned the training data well. However, the validation accuracy was significantly lower than the training accuracy, suggesting that the model did not generalize well to new data. The training loss decreased consistently, while the validation loss did not show a corresponding decrease, further indicating overfitting, as seen in Figure 9.

Based on these results, we hypothesize that DistilBERT's complexity is indeed too high for our small dataset. The model overfits the training data, leading to poor performance on the validation set. To address this, we could consider using a simpler model or augmenting our dataset with more training samples. Future work will involve exploring these alternatives to improve model performance.

6 Analysis and Conclusion

6.1 Performance of LSTM Model

The LSTM model, designed and implemented from scratch, aimed to address the classification task. Despite the extensive literature highlighting LSTM's efficacy in capturing long-term dependencies, our model's performance exhibited certain limitations.

While LSTM networks are renowned for their ability to maintain context over extended sequences, our analysis revealed challenges in effectively leveraging this capability for our classification task. Despite careful tuning of hyperparameters and model architecture, the LSTM model struggled to achieve satisfactory performance metrics on our dataset.

The training and validation metrics displayed an undesirable pattern, with the model failing to generalize effectively to unseen data. Despite achieving high training accuracy, there was a significant disparity between training and validation accuracy, suggesting overfitting.

The observed limitations could stem from various factors. Firstly, the dataset size, consisting of only 500 rows, might have been insufficient to train a complex LSTM model effectively. Additionally, the inherent complexity of the classification task, coupled with the limited dataset, might have posed challenges for the model to learn robust representations.

In light of these observations, further investigation into alternative model architectures and dataset augmentation techniques is warranted. Future iterations of the classification task may benefit from exploring simpler models or incorporating additional data samples to enhance model generalization and performance.

6.2 Evaluation of DistilBERT Model

Utilizing a pre-implemented model, DistilBERT, provided insights into the performance of state-of-the-art transformer architectures on our classification task. Despite DistilBERT's reputation for efficiency and high performance, our evaluation uncovered certain limitations when applied to our dataset.

Similar to the LSTM model, DistilBERT exhibited signs of overfitting, as evidenced by the significant gap between training and validation accuracy. While the model demonstrated exceptional performance on the training set, its ability to generalize to unseen data was suboptimal.

The complexity of DistilBERT's architecture, albeit reduced compared to its predecessor BERT, might have posed challenges for effective learning on our limited dataset. Furthermore, fine-tuning hyperparameters specific to our task and dataset characteristics could have further optimized model performance.

Moving forward, addressing the observed overfitting issues requires a holistic approach, encompassing both model architecture modifications and dataset enhancements. Exploring techniques such as transfer learning and domain-specific pre-training may offer avenues for improving DistilBERT's performance on our classification task.

6.3 Comparative Analysis

Comparing the performance of the LSTM model and DistilBERT on our classification task provides valuable insights into the strengths and limitations of different model architectures. While LSTM networks offer interpretability and the ability to capture long-term dependencies, their performance on our dataset was hindered by overfitting and limited dataset size.

In contrast, DistilBERT, leveraging transformer-based architectures, demonstrated impressive performance capabilities but faced similar challenges related to dataset size and overfitting. The comparative analysis underscores the importance of aligning model complexity with dataset characteristics to achieve optimal performance.

Moving forward, iterative refinement of model architectures and careful consideration of dataset augmentation strategies will be crucial in enhancing classification performance. Additionally, exploring ensemble techniques and hybrid models that leverage the strengths of both LSTM and transformer architectures may offer promising avenues for further improvement.

6.4 Conclusion

In conclusion, while both the LSTM model and DistilBERT show promise for the classification task, they each exhibit limitations that must be addressed for optimal performance. The LSTM model struggles with overfitting and difficulty in generalizing to unseen data, possibly due to its complexity and the limited dataset size. On the other hand, DistilBERT, while demonstrating high accuracy, also faces challenges in maintaining generalization and stability. Moving forward, a combination of model architecture refinement and dataset augmentation strategies will be key to unlocking the full potential of these models for the classification task.

References

[1] B. K. Jha, G. Sivasankari, and K. Venugopal, "Sentiment analysis for e-commerce products using natural language processing," *Annals of the Romanian Society for Cell Biology*, pp. 166–175, 2021.

[2] X. Lin, "Sentiment analysis of e-commerce customer reviews based on natural language processing," in *Proceedings of the 2020 2nd international conference on big data and artificial intelligence*, pp. 32–36, 2020.

[3] A. Papenmeier, D. Kern, D. Hienert, A. Sliwa, A. Aker, and N. Fuhr, "Dataset of natural language queries for e-commerce," in *Proceedings of the 2021 Conference on Human Information Interaction and Retrieval*, pp. 307–311, 2021.

[4] S. Fitzgerald, G. Mathews, C. Morris, and O. Zhulyn, "Using nlp techniques for file fragment classification," *Digital Investigation*, vol. 9, pp. S44–S49, 2012.

[5] A. KOCADINÇ, "Amazon$_p$*roduct*$_c$*omment*, *https* : *//www.kaggle.com/code/ahmetkocadinc/amazon* − *product* − *comment?kernelsessionid* = 162602975,"

[6] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2020.