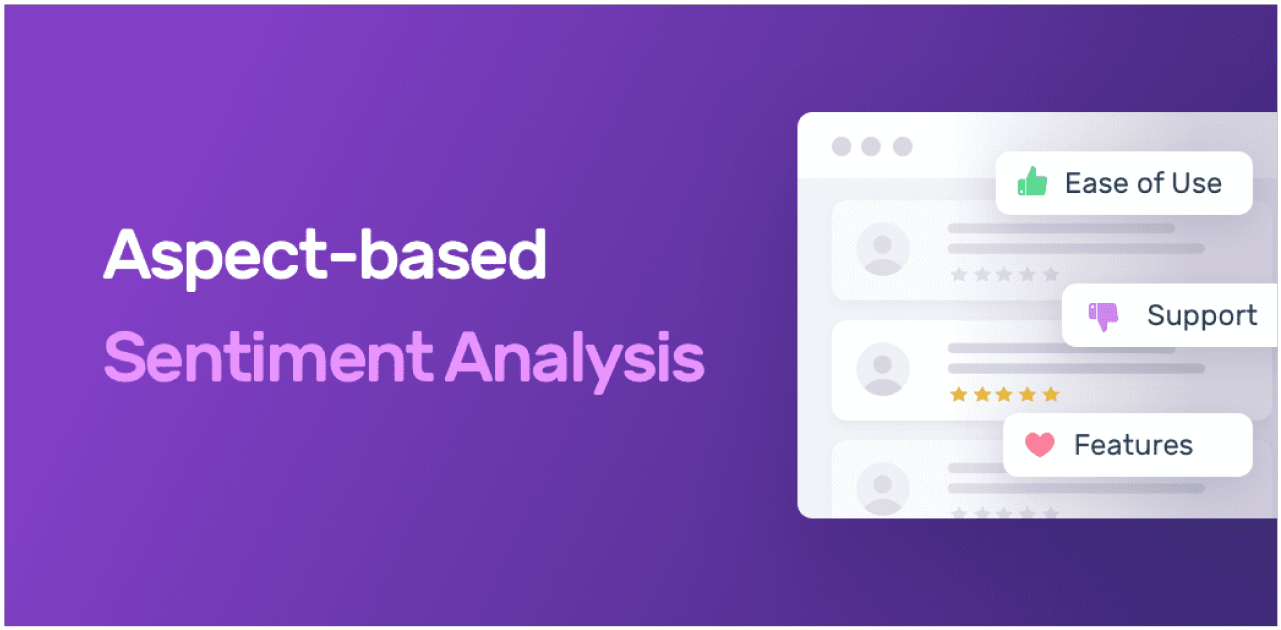


Aspect-Based Sentiment Analysis with Yelp Review Data

Final Report: 31.05.2024

—

Team: Sifael Sebastian Ndandala, Protogene Hahirwabayo, Piumi Abeyrathne
Topic: Natural Language Processing: Aspect-Based Sentiment Analysis
Github: https://github.com/DataMining-CU-Spring2024/Final-Project
WebApp: https://absa-analyzer-production.up.railway.app/

Section I: Project Overview, Goals and Learning Objectives

1.1. Introduction

Technological developments in Natural Language Processing (NLP) are becoming indispensable in shaping how we interact with information and engage in the digital world. The application of NLP technologies, such as ChatGPT and Gemini, is making significant inroads into the digital products we use daily, including how we search for information with search engines, communicate with online customer services, and even how we learn and conduct academic research. This transformative power, if used appropriately, is poised to enhance our productivity and improve our overall experience as we navigate both the physical and digital worlds.

For data engineers, the importance of developing expertise in Natural Language Processing cannot be overstated. NLP is a notoriously data-intensive and computationally demanding field. As the appetite for utilizing unstructured textual data continues to grow to aid the further development of large language models, the demand for data engineers with the technical skill set in NLP to develop the data infrastructure and machine learning models to support this rapidly evolving space will also increase.

Recognizing the potential of Natural Language Processing and the opportunity to develop core skills in the field, we settled on Aspect-Based Sentiment Analysis for our Data Mining project. As an advanced step from Sentiment Analysis, Aspect-Based Sentiment Analysis offers learning opportunities in fundamental techniques and software for Natural Language Processing. Furthermore, it also presents an opportunity to work on a real-world business application of NLP using product and services review dataset to gain tactical insights that can improve business operation.

This report provides an in-depth analysis of our implementation of Aspect-Based Sentiment Analysis using the Yelp dataset. We detail our implementation pipeline, covering techniques for data preprocessing, aspect extraction, feature generation techniques, and the sentiment analysis models developed. Furthermore, we present the results of our analysis and reflect on the challenges faced, the solutions we developed to address these challenges, and our opinions on future considerations.

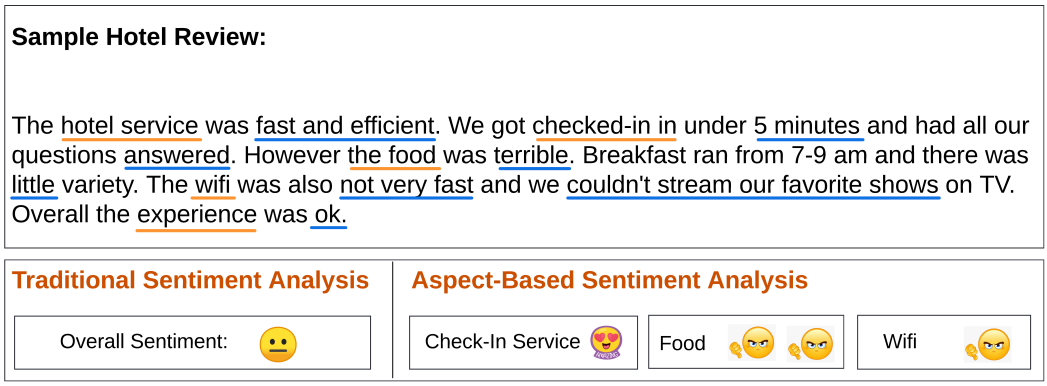
1.2. Project Overview - What is Aspect-Based Sentiment Analysis?

Aspect-Based Sentiment Analysis (ABSA) is a more granular form of sentiment analysis that focuses on identifying sentiments expressed about specific aspects or features of entities within a text (Lui 2015). Unlike traditional sentiment analysis, which assigns a general sentiment to an entire text, ABSA breaks down the text to extract sentiments related to particular components or attributes.

Aspect-Based Sentiment Analysis involves two main tasks:

1. **Aspect Extraction:** Identifying the aspects or features that are the subject of the text
2. **Sentiment Classification:** Determining the sentiment class associated with each identified review

The figure below demonstrates the conceptual idea of Aspect-Based Sentiment Analysis versus traditional sentiment analysis, using an example of a sample hotel review. Aspects of the review are highlighted with an orange underscore, while their respective sentiments are underscored with a blue line.



As demonstrated in the example, this project aims to take traditional sentiment analysis a step further, to dissect the customer reviews into specific sentiment details thereby providing actionable insights into the aspects customers liked and disliked about the product and service.

1.3. Project Goals and Learning Objectives

Our choice to focus on Aspect-Based Sentiment Analysis (ABSA) for several compelling reasons that align with our educational and professional goals. Firstly, the project offers an opportunity to apply and extend knowledge developed in data analytics course in the context of textual data. Using machine learning techniques including classification and deep learning and feature generating techniques, we will learn to apply numerical techniques on unstructured data.

Secondly, our choice is driven by an interest in delving into Natural Language Processing, a domain that holds significant promise in both industry applications and academic research. Through this project, we aspire to familiarize ourselves with core concepts underpinning some of the most sophisticated NLP solutions, such as ChatGPT and Gemini. Furthermore, we expect to navigate the complexities of NLP and understand its inherent challenges, preparing us for advanced applications such as text generation.

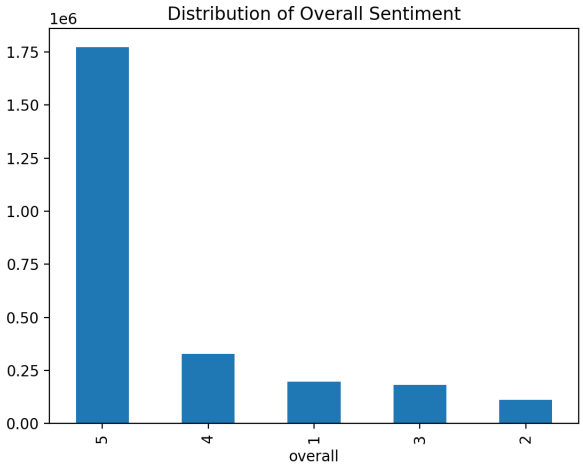
Finally, the ABSA project serves as a platform to explore and master various technologies and software tools used in NLP tasks, thereby enhancing our proficiency in programming, data processing, and machine learning. We intend to explore toolkits including **NLTK** and **Spacy** for NLP tasks, **scikit-learn** and **PyTorch** for machine learning and deep learning, and leverage pre-trained models such as **pre-trained word embeddings** and **BERT** models. Additionally, we will explore frameworks such as Django and R Shiny to package the final product into a user-friendly live application. By integrating these elements, our project not only consolidates our current understanding but also prepares us for more complex challenges ahead, ensuring we develop comprehensive competencies in these critical areas.

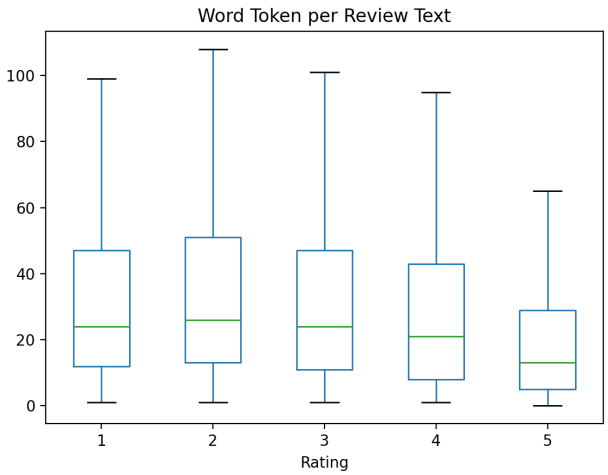
Section II: Dataset Acquisition and Description

2.1. Dataset Acquisition and Description

Our initial dataset of choice was the Amazon Product reviews dataset, a well known and accessible encompassing a wide range of product categories (McAuley 2018). However, in our exploration, we observed significant class imbalances across all product categories. Approximately ~85% of all reviews were positive (4 or 5 stars), about 6% were neutral (3 stars), and the remaining roughly 9% were negative (1 or 2 stars). While the high prevalence of positive reviews reflect the tendency of more satisfied customers to leave feedback, it would severely limit our ability to develop a robust model.

Amazon Product Review Dataset (McAuley, J. (2018)

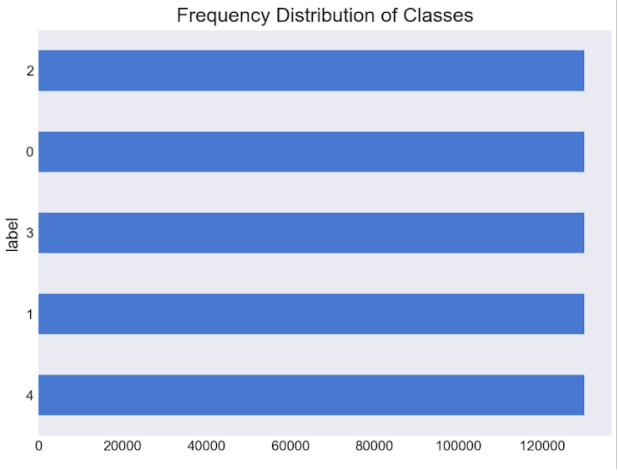




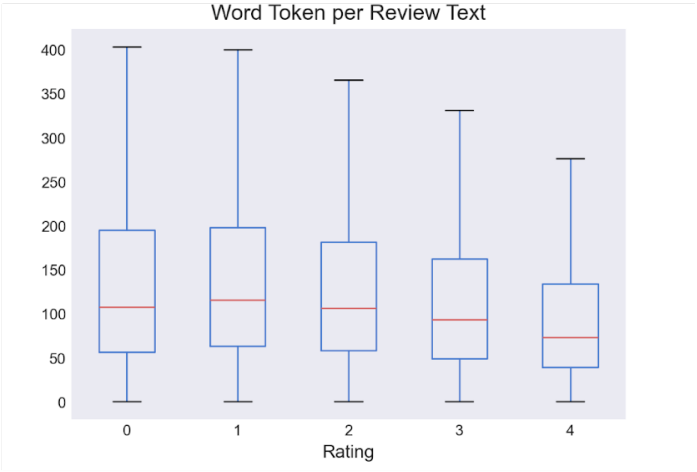
Recognizing the challenges posed by the significant majority class in the Amazon Review dataset, we sought out an alternative dataset that would afford us a more equitable distribution for our analysis and modeling efforts. In an attempt to avoid the potential biases that might arise from artificial rebalancing

techniques, our search led us to select the Yelp Reviews dataset (Yelp 2019). This dataset is known for its balanced composition and has undergone a more rigorous curation process, which we anticipate will enhance the quality and reliability of our exploratory and analytical endeavors.

Yelp Review Dataset



Uniform Distribution Against Classes



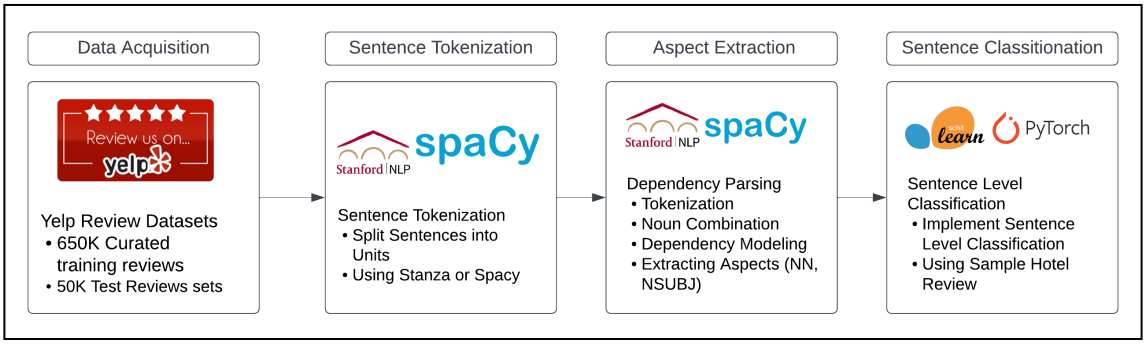
Comparable Token to Review across Classes

Section II: Aspect-Based Sentiment Analysis Implementation Pipelines

Aspect-Based Sentiment Analysis requires the combination of two tasks: *Aspect Extraction* and *Sentiment Analysis*. While these tasks interact with each other, they require separate implementation pipelines, which can be combined into a final output. This section discusses the implementation of these pipelines, providing detailed information on the processes and outcomes.

3.1. Aspect Extraction Pipeline

The Aspect Extraction pipeline is designed to dissect individual sentences from the user reviews, extract relevant subject of the text and analyse the sentiment at the sentence level with respect to the subject. The pipeline below demonstrates the overall implementation strategy.



3.2. Dependency Parsing for Aspect Extraction

Dependency parsing is an NLP technique that identifies the dependency relationships between words in a sentence, visualizing them as a directed graph where nodes represent words and edges represent dependencies (Liu, Bing 2012). For aspect extraction, it enables the precise identification of aspects (nouns or noun phrases) and the sentiment-bearing words (adjectives or verbs) that describe those aspects (Liu, Bing 2012).

To implement aspect extraction, we leveraged Stanford's dependency parser, which builds relationships between words in a sentence. It works by identifying the roots of the words and mapping the relationships of other words to the root. Our implementation follows a six-step process that begins with tokenization and POS tagging, combining consecutive nouns (e.g., "Hotel Service"), and removing stop words. The output is parsed to identify roots per sentence, where each word is mapped. From these, the dependency features are labeled, and nouns are finally extracted.

To demonstrate this in action, we implemented the parse on two sentences below. The parser successfully identified two aspects of the sentence: "food" and "variety."

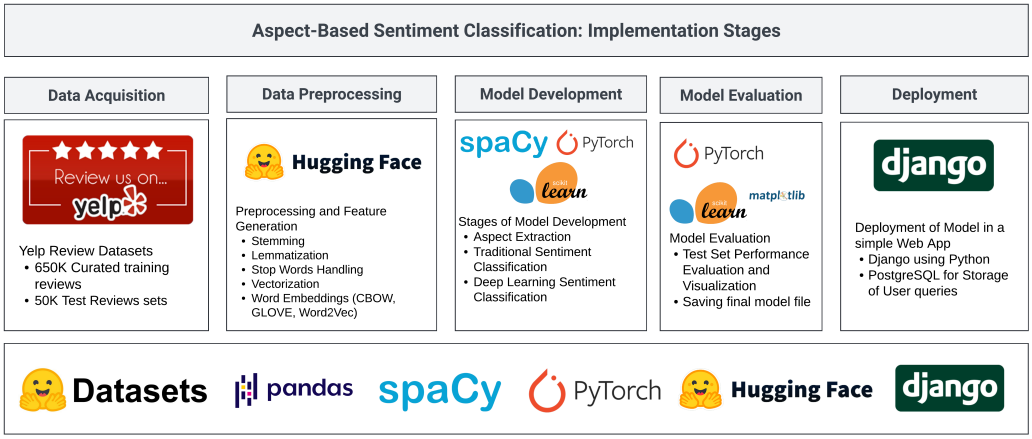
Example: *'However the food was terrible. Breakfast ran from 7-9 am and there was little variety.'*

```
Step 1: Tokenization and POS Tagging completed.
Tokenized and tagged sentence: [('However', 'RB'), ('the', 'DT'), ('food', 'NN'), ('was', 'VBD'), ('terrible', 'JJ'), ('.', '.'), ('Breakfast', 'NNP'),
('ran', 'VBD'), ('from', 'IN'), ('7-9', 'JJ'), ('am', 'VBP'), ('and', 'CC'), ('there', 'EX'), ('was', 'VBD'), ('little', 'JJ'), ('variety', 'NN'), ('.', '.')]

Step 2: Combining consecutive nouns completed.
New word list after combining nouns: ['However', 'the', 'food', 'was', 'terrible', '.', 'Breakfast', 'ran', 'from', '7-9', 'am', 'and', 'there',
'was', 'little', 'variety', '.']

Step 3: Filtering stop words completed.
Words after stop word filtering: ['However', 'food', 'terrible', '.', 'Breakfast', 'ran', '7-9', 'little', 'variety', '.']

Step 4: Dependency Parsing
Dependency parse results: [('However', 2, 'advmod'), ('food', 0, 'root'), ('terrible', 2, 'amod'), ('.', 2, 'punct')]

Step 5: Extracting Dependency Features
Feature: {'However': 'RB', 'food': 'NN', 'terrible': 'JJ', '7-9': 'JJ', 'little': 'JJ', 'variety': 'NN'}

Step 6: Extract Aspects
Final Aspects: ['food', 'variety']

['food', 'variety']
```

Once the Aspects are extracted, we can then perform sentence level sentiment analysis and map the sentiment value to the aspect extracted from the dependency parser.

3.3. Sentiment Analysis Pipeline

Our implementation of the Sentiment Analysis task follows five key stages outlined in the pipeline below, namely: Data Acquisition, Data Preprocessing, Model Development, Model Evaluation and Deployment.



The following sections will discuss the implementation of each stage, highlighting interesting findings, techniques employed and their outcomes, challenges faced and how we changed our approach to deal with these challenges.

3.3.1. Data Preprocessing

Our preprocessing steps included five key steps with the goal of cleaning out data and preparing it for feature extraction. The four stages included: *Punctuation Removal*, *Tokenization*, *Numerical Word Replacement*, *Spelling Correction* and *Lemmatization*. The code below demonstrates the abstracted implementation. In the appendix, we attach the complete implementation of the Preprocessing Class.

```
def preprocess(self, text):
    """
    Executes a preprocessing pipeline on the text,
    punctuation removal, numerical word replacement,
    (optional spelling correction), and lemmatization.
    """
    self.text = text
    self.punctuation_removal()
    self.replace_num_with_words()
    self.correct_spelling()
    self.lemmatize_text()
    return self.text.lower()
```

3.3.2 Punctuation Removal

The **punctuation_removal()** method in the `**Preprocessor**` class is responsible for cleaning the text by removing all punctuation marks. This method replaces the ampersand ('&') with the word "and" and then uses a regular expression to substitute any non-alphabetical character with a space. This step ensures that the text is devoid of punctuation, which simplifies further processing steps like tokenization and lemmatization.

```
def punctuation_removal(self):
    """Removes punctuation from the text, replacing '&' with 'and'."""
    self.text = self.text.replace('&', 'and')
    self.text = re.sub("[^a-zA-Z]", " ", self.text)
```

3.3.3 Tokenization

The `**tokenize**` method splits the input text into individual words or tokens using the `**nltk.word_tokenize**` function. Tokenization is a fundamental step in text preprocessing as it converts the continuous text into discrete components (tokens), which are easier to analyze and process in subsequent steps such as lemmatization or spelling correction.

```
def tokenize(self):
    """Tokenizes the text for further processing."""
    self.word_tokenize = nltk.word_tokenize
    return self.word_tokenize(self.text)
```

3.3.4. Numerical Word Replacement

In the `**replace_num_with_words**` method, numerical values within the text are converted into their corresponding word representations. This is achieved by iterating over each token, checking if it is a digit, and using the `inflect` library to convert the digit to words. This conversion is useful for natural language processing tasks where numbers need to be analyzed in their semantic context.

```
def replace_num_with_words(self):
     """Replaces all numeric values with their word representations"""
    self.text = ' '.join(self.p.number_to_words(word) if word.isdigit() else word for word in
self.tokenize())
```

3.3.5. Spelling Correction (Optional due to Computational Feasibility)

The optional `correct_spelling` method aims to correct any spelling mistakes in the tokens. It uses the `SymSpell` library, which is a fast spelling correction tool. For each token, the method tries to find the closest correct spelling. If no correction is possible (e.g., if the token is not found in the dictionary), the original token is retained. This method helps improve the quality of the text data, especially if it was sourced from informal communications like social media posts.

```
def correct_spelling(self):
    """Attempts to correct the spelling of words in the text. Optional in preprocess pipeline."""
    new_sentence = []
    for word in self.tokenize():
        try:
            correct_word = self.sym_spell.lookup(word, Verbosity.CLOSEST)[0].term
        except IndexError:  # Handles exceptions where word correction is not possible
            correct_word = word
        new_sentence.append(correct_word)
    self.text = ' '.join(new_sentence)
```

3.3.6. Lemmatization

The `lemmatize_text` method converts each token into its base or dictionary form (lemma) according to its part of speech. This is done using `nltk`'s `WordNetLemmatizer` and a part-of-speech tagger (`nltk.pos_tag`). Lemmatization helps reduce the inflectional forms of words, thus supporting the uniformity of the dataset. Additionally, this method filters out stopwords (commonly used words that are often removed in NLP tasks to focus on meaningful words), further refining the text for analysis.

```
def lemmatize_text(self):
        """
        Lemmatizes the text, converting words to their base form according to their parts of speech,
        and removes stopwords.
        """
        tokens = nltk.pos_tag(self.tokenize())
        lemmatized_tokens = [
            self.lemmatizer.lemmatize(token[0],
                                      pos=self.pos_dict.get(token[1][0].upper(), wordnet.NOUN))
            for token in tokens if token[0].lower() not in stopwords.words('english')
        ]
        self.text = ' '.join(lemmatized_tokens)
```

Preprocessing Analysis and Computational Complexity

The preprocessing steps above provide the best combination of text cleaning ahead of deeper NLP tasks, however, the computational complexity of iterating through each review made it near impossible to run on all our dataset. As a result we made two preprocessing steps optional: correct_spelling and

replace_num_with_words. Eliminating these two significantly improved our preprocessing and therefore we excluded them entirely in the final Preprocessing pipeline.

3.4. Feature Extraction

Following the preprocessing steps outlined above, we implemented traditional feature extraction techniques for Natural Language Processing: **Bag of Words (BoW)** and **Term Frequency-Inverse Document Frequency (TF-IDF)**. Both Bag of Words and TF-IDF are count-based feature extraction methods that often result in very sparse feature sets, as different reviews typically contain only a small subset of the vocabulary in the corpus. We were particularly keen on TF-IDF because it provides a weighting that emphasizes less frequent words in the overall reviews, which offer significant predictive value in specific reviews.

The implementation below demonstrates both the Count Vectorizer and TF-IDF for the case of 1-gram/ token features. In our final models, we also included a 2-gram model.

Bag of Words - CountVectorizer

```
CountVectorizer( analyzer = 'word',          # Word level vectorizer
                 lowercase = True,           # Lowercase the text
                 tokenizer = word_tokenize, # Use this tokenizer
                 token_pattern = None,
                 stop_words = 'english',     # remove english stopwords
                 ngram_range = (1, 1),       # 1 gram model
                 min_df = 5,)                # use words that appear > 5
```

TF-IDF Vectorizer

```
TfidfVectorizer( analyzer='word',            # Word level vectorizer
                 lowercase=True,             # Lowercase the text
                 stop_words = 'english',     # remove english stopwords
                 min_df = 5,                 # use words that appear > 5
                 tokenizer= word_tokenize,  # Use this tokenizer
                 token_pattern = None)
```

Finally, it is noteworthy to highlight that the most important parameter was the min_df = 5 which only included words that appeared 5 times of higher in the text. This resulted in a dataset with 14,102 features.

Section IV: Model Training and Results

4.1. Model Training and Results

On this project, we tested two different feature extraction methods combined with two machine learning models to evaluate their effectiveness in aspect-based sentiment analysis using the Yelp dataset. The feature extraction methods included Count Vectors and Term Frequency-Inverse Document Frequency (TF-IDF), applied to both 1-gram and 2-gram feature vectorization. The machine learning models tested were Naive Bayes and Decision Trees. This section outlines the models tested and discusses the results, highlighting the issue of overfitting observed in some cases.

Table of Accuracy (in %) of Models on Train and Test Data for Naive Bases and Decision Trees

Features Extractions Methods		Naive Bayes		Decision Trees	
		Train	Test	Train	Test
1-gram	Count Vectors	65	49	100	35
	TF-IDF	68	49	100	35
2-gram	Count Vectors	83	45	99	35
	TF-IDF	86	46	99	34

The results from the Naive Bayes model indicate a moderate level of overfitting, particularly with the 2-gram models. While the training accuracy for 2-gram Count Vectors and 2-gram TF-IDF is relatively high (83% and 86% respectively), the test accuracy drops significantly (45% and 46% respectively). This discrepancy suggests that the model is learning the training data well but fails to generalize to unseen data.

The Decision Trees model, however, shows a significant level of overfitting across all feature extraction methods. Both the 1-gram and 2-gram models exhibit perfect or near-perfect training accuracy (100% and 99% respectively), but their test accuracy is substantially lower (ranging from 34% to 35%). This pattern highlights that Decision Trees tend to overfit the training data, capturing noise and details specific to the training set rather than generalizable patterns.

Overall, while Naive Bayes shows some promise, especially with TF-IDF feature extraction, the models need further tuning and possibly regularization techniques to mitigate overfitting and improve their ability to generalize to new, unseen data. The Decision Trees model, on the other hand, requires significant adjustments or an alternative approach to address its overfitting issues.

4.2. Model Training Shortcoming

In addition to the Naive Bayes and Decision Trees models, we also attempted to implement more advanced models, including pre-trained BERT models and a Recurrent Neural Network (RNN). These models are well-known for superior performance in natural language processing tasks due to their ability to capture complex patterns in the data. However, we encountered significant challenges related to computational expensiveness. Furthermore, limited access to GPU resources meant a prohibitively length of training time spanning over days. The training process for these models is highly resource-intensive, and without adequate computational power, we were unable to effectively complete the training. As a result, we could not fully leverage the capabilities of these sophisticated models in our aspect-based sentiment analysis project. This limitation underscores the importance of access to robust computational infrastructure when working with advanced machine learning techniques.

4.3. Model Deployment through Django Application

One of our goals with this project was to create a Django application that integrated aspect extraction and sentiment analysis, enabling users to input reviews and obtain analyzed results. While we successfully developed the application in a local environment, we faced significant challenges during deployment. Specifically, we encountered difficulties configuring the production server with the necessary dependencies. The various Python packages required for our application often conflicted with each other, leading to compatibility issues that we were unable to resolve within our available timeframe. As a result, although the application functioned well in a development environment, these dependency conflicts prevented us from deploying it effectively in a production setting. This experience highlighted the complexities involved in deploying machine learning models in real-world applications and the importance of meticulous dependency management.

Section V: Project Reflections and Learning

As we reflect on the project implementation and the technical challenges, there are a few learning that have become apparent as necessary takeaways when working on similar Data Mining projects.

5.1. Parallel and Distributed Computing is an Exceptionally Important Toolkit

Throughout the Aspect-Based Sentiment Analysis (ABSA) project using the Yelp dataset, we have come to appreciate the immense value of parallel and distributed computing. Processing large datasets and running complex machine learning algorithms are computationally intensive tasks that can significantly benefit from parallelism. By leveraging tools like Dask and Dataset modules, we were able to parallelise the preprocessing steps allowing us to generate features for model training. Using core pandas and

numpy proved insufficient to deal with the Yelp dataset. This experience underscored the necessity of understanding and utilizing these computing paradigms that use GPU and parallelise CPU operations, as they are indispensable for handling large-scale data and achieving faster, more scalable solutions in both academic research and industry applications.

5.2. Stand on Top of Giants: Use Pre-trained Models Whenever Possible

Another critical learning point from our project was the strategic use of pre-trained models. The NLP field has made significant strides, producing powerful models such as BERT, GPT, and various pre-trained word embeddings. Using this models on a smaller subset of our data allowed us to leverage existing knowledge and advancements, thereby saving time and computational resources. While we could not scale these models to the large dataset such as the full Yelp dataset, we saw better performance using pre-trained models on a smaller subset of the data, enabling us to achieve higher accuracy and efficiency compared to building models from scratch. It is evident that future projects should begin by using pre-trained weights in order to quickly use learnings from others and customising these models for the individual task at hand.

5.3. Develop ML Training Pipelines and Processes Based on Checkpoints

The development of machine learning training pipelines and processes based on checkpoints proved to be an invaluable practice during our ABSA project. Checkpoints allowed us to save the state of our models at various stages of preprocessing and model training, providing a mechanism to resume training without starting from scratch in case of interruptions. This was particularly beneficial given the iterative nature of machine learning model development, where experiments often need to be paused, tweaked, and resumed. By implementing checkpoint-based pipelines, we were able to ensure continuity, reproducibility, and efficiency in our workflow. This approach not only safeguarded our progress but also facilitated systematic experimentation and fine-tuning of models, which are crucial for achieving optimal performance in machine learning tasks.

References and Sources:

1. Liu, B. ed., (2015). Aspect Sentiment Classification. [online] Cambridge University Press. Available at: https://www.cambridge.org/core/books/abs/sentiment-analysis/aspect-sentiment-classification/CD953CFBA4FD46CCD1B1CAADADACD061 [Accessed 30 May 2024].
2. McAuley, J. (2018). Amazon review data. [online] Ucsd.edu. Available at: https://jmcauley.ucsd.edu/data/amazon/.
3. Yelp (2019). Yelp Dataset. [online] Yelp.com. Available at: https://www.yelp.com/dataset
4. Liu, Bing (2012). Sentiment Analysis and Opinion Mining. Morgan & Claypool Publishers, 2012

5. De Marneffe, M.-C. and Manning, C. (2008). Stanford typed dependencies manual. [online] Available at: https://downloads.cs.stanford.edu/nlp/software/dependencies_manual.pdf.
6. lg845.github.io. (n.d.). Annotating your corpus with the Stanford Dependency Parser. [online] Available at: https://lg845.github.io/LAEL_CoreNLP/ [Accessed 31 May 2024].
7. Universal Dependencies (n.d.). Syntax. [online] Available at: https://universaldependencies.org/u/overview/syntax.html.