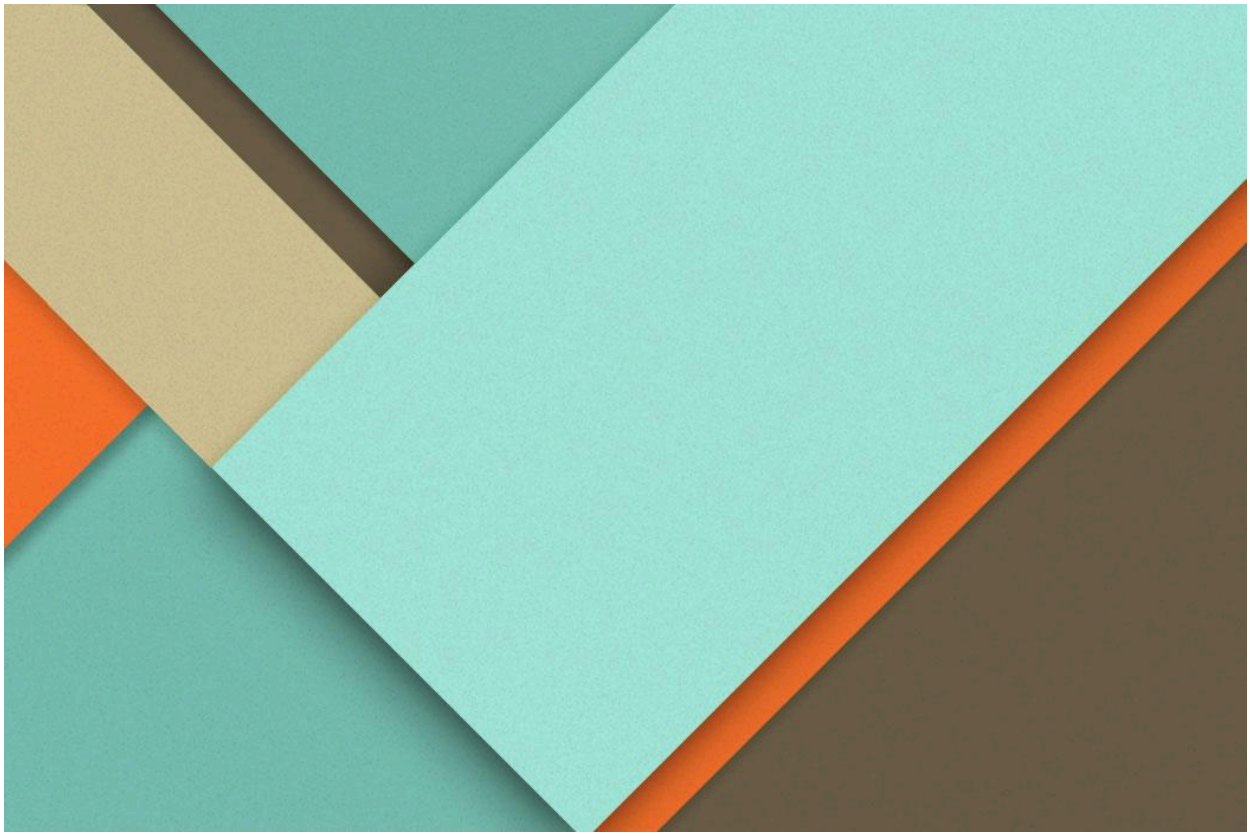


Aspect-Based Sentiment Analysis

Project Update: 09.04.2024

—

Team: Sifael Sebastian Ndandala, Piumi Abeyrathne, Protogen Hahirwabayo

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/

Project Overview

Natural Language Processing (NLP) is a critical and fast-evolving field that plays a pivotal role in bridging human language and computational understanding, consequently it is revolutionizing how we interact with technology and each other. Advancements in NLP computational techniques are paving a way for innovations that range from simple applications such as text analysis and language translation to complex systems capable of understanding nuances of human language, reasoning, and emotions and generating coherent and appropriate text in conversation.

Recognizing the potential of understanding and developing technical skills in NLP, we settled on Aspect-Based Sentiment Analysis as the topic for our Data Mining Project. **Aspect-Based Sentiment Analysis** is an advanced technique in natural language processing that goes beyond determining the overall sentiment of a text. Instead, it identifies sentiments related to specific aspects of products and/or services thereby providing insights that can be translated into business strategy. The example below demonstrates the difference between **Traditional Sentiment Analysis** and **Aspect-Based Sentiment Analysis** using a sample Hotel Review.

Sample Hotel Review:

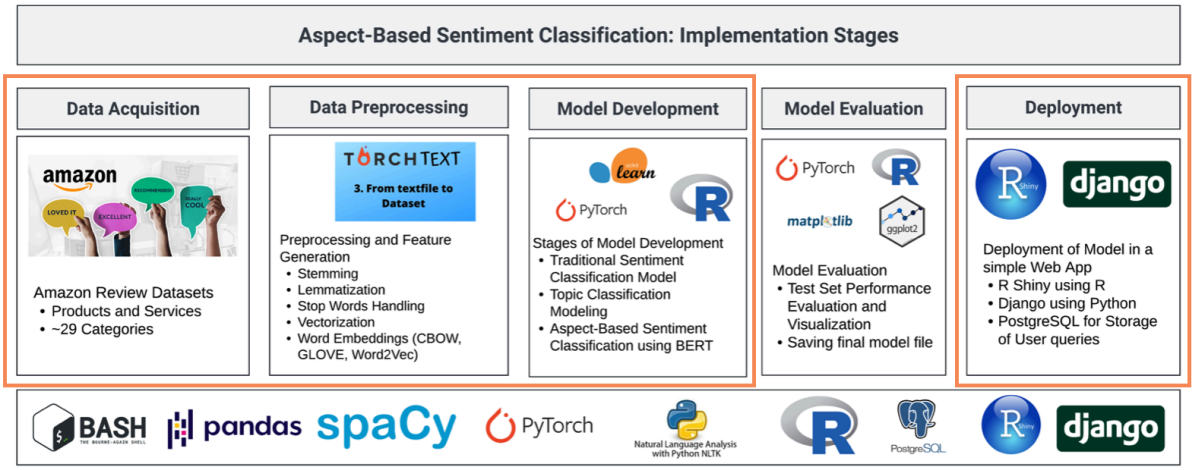
> The hotel service was fast and efficient. We got checked in in under 5 minutes and had all our questions answered. However the food was terrible. Breakfast ran from 7-9 am and there was little variety. The wifi was also not very fast and we couldn't stream our favorite shows on TV. Overall the experience was ok.

Traditional Sentiment Analysis	Aspect-Based Sentiment Analysis		
Overall Sentiment: 😐	Check In Service 😍	Food 😠😠	Wifi 😠

This project aims to take traditional sentiment analysis a step further, to dissect customer reviews into specific sentiment details thereby providing granular and actionable insights into what aspects customers liked or disliked about the product and service.

Implementation Updates

The implementation of the Aspect-Based Sentiment Classification project follows five key stages, as outlined in the implementation guidance below: Data Acquisition, Data Preprocessing, Model Development, Model Assessment & Improvement, and Deployment. Currently, we have made progress in stages 1-3 and have successfully deployed a preliminary version of our web application, demonstrating its early capabilities as part of stage 5.



In the following sections, we will discuss the implementation of each stage, highlighting interesting findings, the techniques employed, their outcomes, and the challenges faced that led to informed changes in our approach.

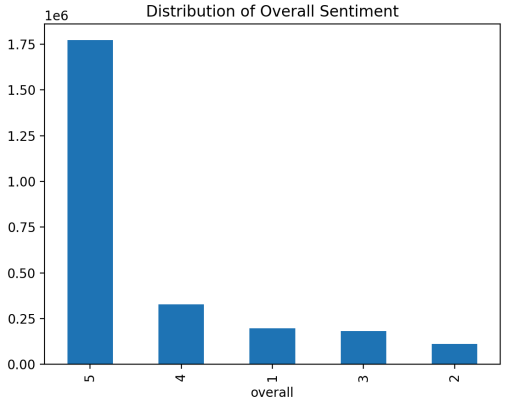
Stage 1: Data Collection

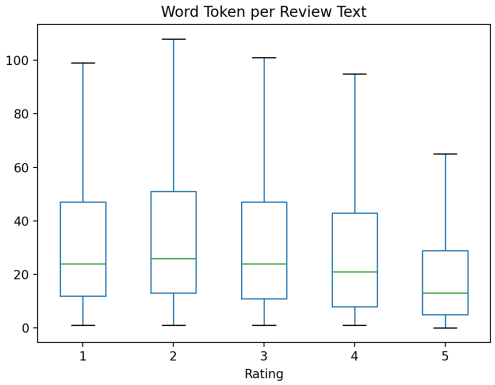
Our initial dataset of choice was the Amazon product reviews, a well-known and accessible dataset encompassing a wide range of product categories. The variety in categories offered an opportunity to develop robust models that could target aspects of the review across many domains thereby making our model widely usable.

In our exploration of the Amazon Reviews dataset, we observed a significant class imbalance issue 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). The prevalence of positive reviews could reflect a tendency for more engaged or satisfied customers to leave feedback. However, this imbalance severely limits our ability to develop a

robust model. On the other hand, it was informative to find that the word token distribution, normalized against outliers, was relatively uniform, with a median of about 20-25 words per review. Positive reviews displayed a slightly more compact distribution.

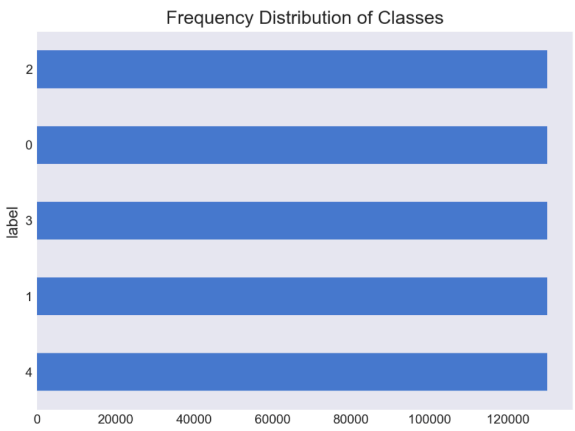
Amazon Review Datasets:



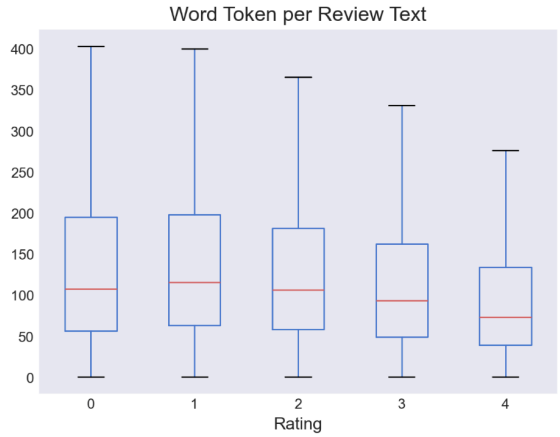


Recognizing the challenges posed by the significant majority class in the Amazon Review dataset, we sought out an alternative dataset that would afford us a more equitable distribution for our analysis and modeling efforts. In an attempt to avoid the potential biases that might arise from artificial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



Uniform Distribution Against Classes



Comparable Token to Review across Classes

Stage 2: Data Preprocessing and Feature Extraction

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()
```

2.1. 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)
```

2.2. 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)
```

2.3. 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())
```

2.4. 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)
```

2.5. 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

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 but offers 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.

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

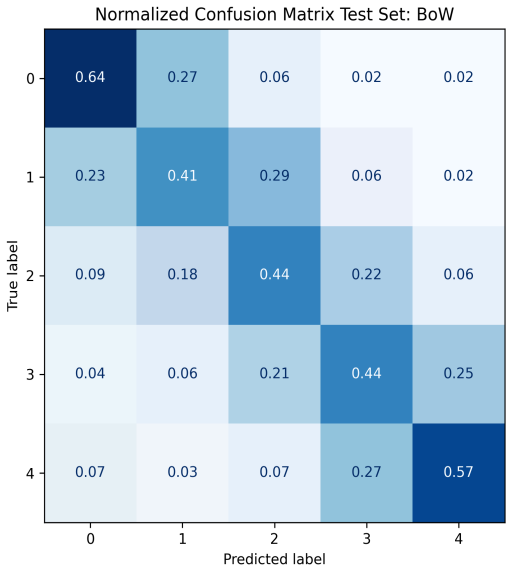
```
TfidfVectorizer( analyzer='word',            # Word level vectorizer
                 lowercase=True,             # Lowercase the text
                 stop_words = 'english',     # remove english stopwords
                 min_df = 5,                 # use words that appear > 5
                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

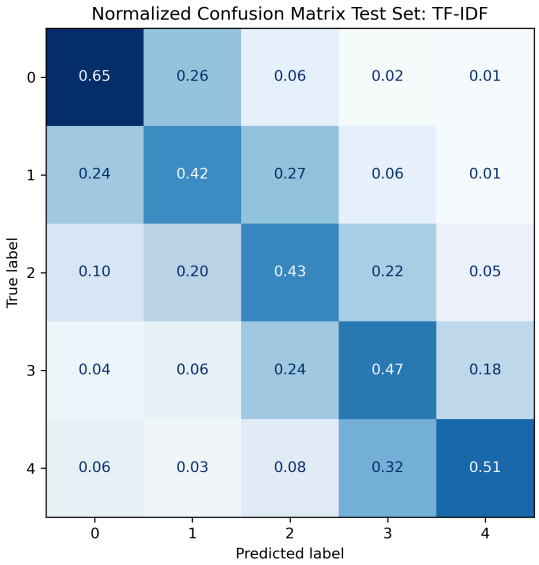
Stage 3: Model Development

Our first iteration of Model Development relied on BoW and TF-IDF features. Using these features, we intended to implement three specific models: Naive Bayes, Support Vector Machines (SVM), and Logistic Regression. These models were designed to classify the text into their respective ratings from 0-4 (Very Bad to Very Good). The size of the features/predictors for these models is a 1-gram vector of 14,102 words that appeared a minimum of 5 times in the corpus.

Due to computational limitations, training the Logistic Regression and SVM models was significantly effortful, while the Naive Bayes model yielded results with relatively low cost in training time. In particular, the SVM model would deplete all the computational resources such that our computers needed restarting. On the other hand, Logistic Regression trained continuously without end, forcing us to cancel the training and explore other techniques.

The NaiveBayes model achieved 64% and 65% accuracy for the Bag of Words and TF-IDF feature sets, respectively. The confusion matrices below demonstrate the breakdown of accuracy by each review rank. {0: Very Bad, 1: Bad, 2: Neutral, 3: Good, 4: Very Good}

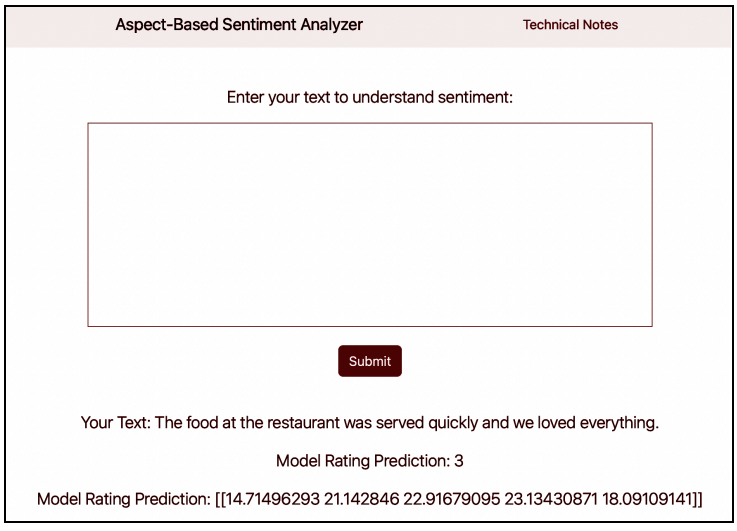




Stage 5: Model Deployment

Completing our first iteration of model development allowed us to deploy the model as a product through a web application, enabling users to directly enter reviews and interact with the model. To achieve this, we used Python's Django web development framework to build a User Interface that accepts text and returns the classified sentiment. The application is available here:

https://absa-analyzer-production.up.railway.app/



Challenges and Solutions

During the implementation stages, we encountered several challenges that necessitated changes in our approach, techniques, and the adoption of new software to meet our goals. Some of these challenges have been highlighted in the implementation sections above. Below is a summary of the challenge and the changes in approach we made

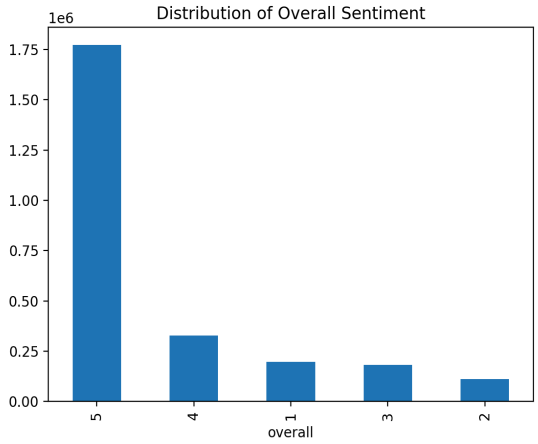
Challenge 1: Class Imbalance in Amazon Reviews

Our first major challenge was the class imbalance present in the original dataset we planned to use. The Amazon review dataset contained approximately 86% positive reviews, and 6% each of neutral and negative reviews. As a result, we could not train a robust model that could reliably predict positive and neutral reviews with high accuracy.

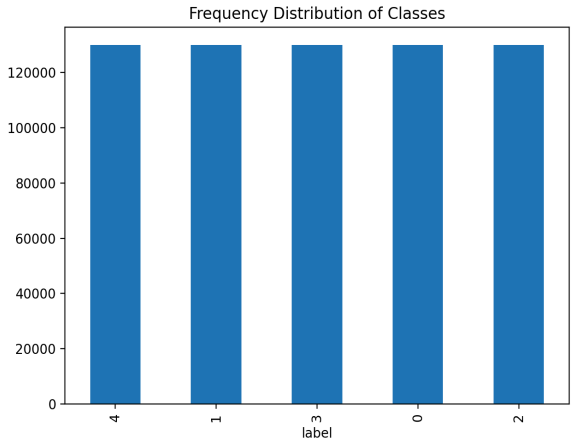
Solution 1: Switching the Dataset from Amazon to Curated Yelp Reviews

To address the class imbalance problem, we searched for a more balanced dataset that offered the same robustness in categorical reviews but did not exhibit class imbalances. We settled on the Yelp Review datasets, which did not have class imbalances and offered the categorical features that Amazon Reviews did (i.e., Yelp reviews are about businesses and products much like Amazon Reviews).

Amazon Reviews Dataset



Yelp Review Dataset

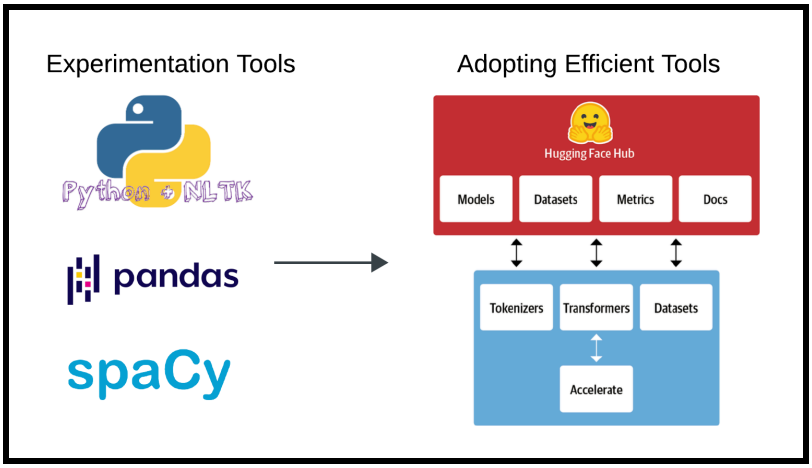


Challenge 2: Computational Limitations

Computational limitations posed another significant challenge in the implementation of our project. Natural Language Processing is notoriously computationally heavy, and although we anticipated this, it became evident that we would need to optimize our code and be creative in our implementation. Preprocessing steps such as tokenization and the use of regular expressions on pandas dataframes proved to be incredibly inefficient and, at times, prohibitively so.

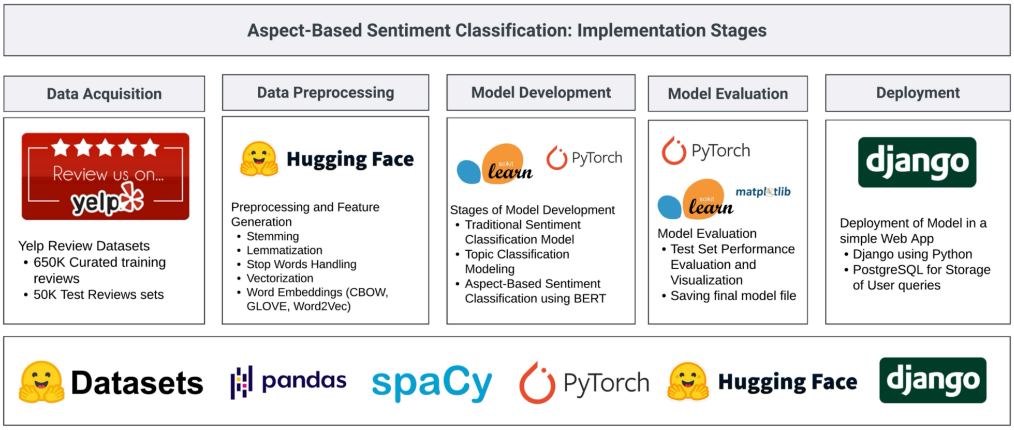
Solution II: Transition to Computationally Efficient Tools like the HuggingFace Ecosystem

To overcome the computational limitations, we explored a number of modern tools in NLP and discovered HuggingFace, an ecosystem that contains optimized preprocessors, datasets, and even pre-trained models that interface seamlessly with packages like pandas and PyTorch. This move significantly enhanced our processing efficiency and allowed us to handle large datasets more effectively.



Revised Implementation Steps and Next Steps

The initial iterations of our implementation have provided valuable insights into the complexity of Natural Language Processing and illuminated various approaches, tools, and techniques that we can employ to overcome these challenges. Consequently, we have revised the set of tools and techniques that we will use in the latter part of our execution to achieve our target results. Below is the revised implementation plan, detailing the appropriate tools and techniques that we will use going forward.



Appendix: Sources and References

Yelp Review Dataset: https://www.yelp.com/datasets

HuggingFace Datasets: https://huggingface.co/docs/datasets/en/index

Spacy Package: https://spacy.io/

Spacy NLP Tutorial: http://course.spacy.io/