DS745: Project Four

John Woodward

2023-11-29

Oxford Dictionary Parsing and Sentiment Analysis

Are you ready to join me on cutting-edge natural language processing (NLP) AI? I'm delving into the intriguing world of pre-trained NLP transformers for the first time. Therefore, I selected Python to experiment and analyzed all 35,000 unique words of the Oxford Dictionary for sentiment. To analyze sentiment, I will borrow insights from an NLP transformer model trained by Amazon product reviews (Peirsman, 2023). Join me and explore the English language, tokenize words, unveil sentiments in English vocabulary, and learn about NLP transformers.

Oxford Dictionary Parsing

Pre-processing in text mining projects is crucial for helpful analysis; thus, it is indispensable. First, I loaded the data by downloading the text from a GitHub URL that contained the Oxford English Dictionary (Sujith, 2023). Then, I used the `re` library and regular expression pattern recognition to eliminate specific lines from the data (Python regex, 2023). These included abbreviations, symbols, suffixes, prefixes, and other non-words. I also used it to separate words from their definitions, which was challenging due to the various delimiters used in the data. The general format of the data was word-delimiter-definition. The delimiter consisted of a long list of different word types, such as "n.", "v.", "adj.", "adv.", "prep.", and so on. These delimiters varied, and there were differences in the em-dashes and other nuances that complicated tidying.

Oxford Dictionary Meets Sentiment Transformers

Next, to apply the NLP transformer, I encountered configuration and computation challenges:

1. I spent at least 8 hours downloading and configuring PyTorch correctly. I encountered several challenges and went down different paths, but I eventually found a solution that

worked. One challenge was solving package dependencies time-efficiently; I had to install an upgraded conda package solver called `libmamba`, which sped solving dependencies up at least 50% in the conda environments. Secondly, to avoid package clashes between PyTorch and NumPy, I had to use a custom environment (i.e., `Sentiment`) and downgrade from Python 3.12 to Python 3.11; then, I could use the transformers from `Huggingface.co` without any issues.

2. Lastly, after selecting the pre-trained model that was useful with Amazon product reviews and predicted sentiment from 1 to 5 stars based on a description (Peirsman, 2023), I discovered that it bottlenecked the project because of how long it took to compute each Word and Definition's sentiment. So, I focused solely on word sentiment scores. Furthermore, I multithreaded computing the sentiment scores for over 35,000 words from the Oxford Dictionary to complete the project on time. This process allowed for proper organization and efficient use of my processors, but more importantly, processing the text data needed for the project. My solution divided the data into alphabetically exported CSV files (e.g., Aa, Ab, Ac,..., Zx, Zy, Zz), allowing easy failure recovery and minimizing memory loss. Thus, I effectively distributed the workload between my laptop and desktop computers, enabling me to achieve my goals in 20 hours of total elapsed run time.

Text Mining Sentiment Analysis

Moving onto the visualizations of our analysis, Figure 1 shows the word sentiment distribution of the dictionary:

Figure 1: Kernel Density Plot of Word Sentiments

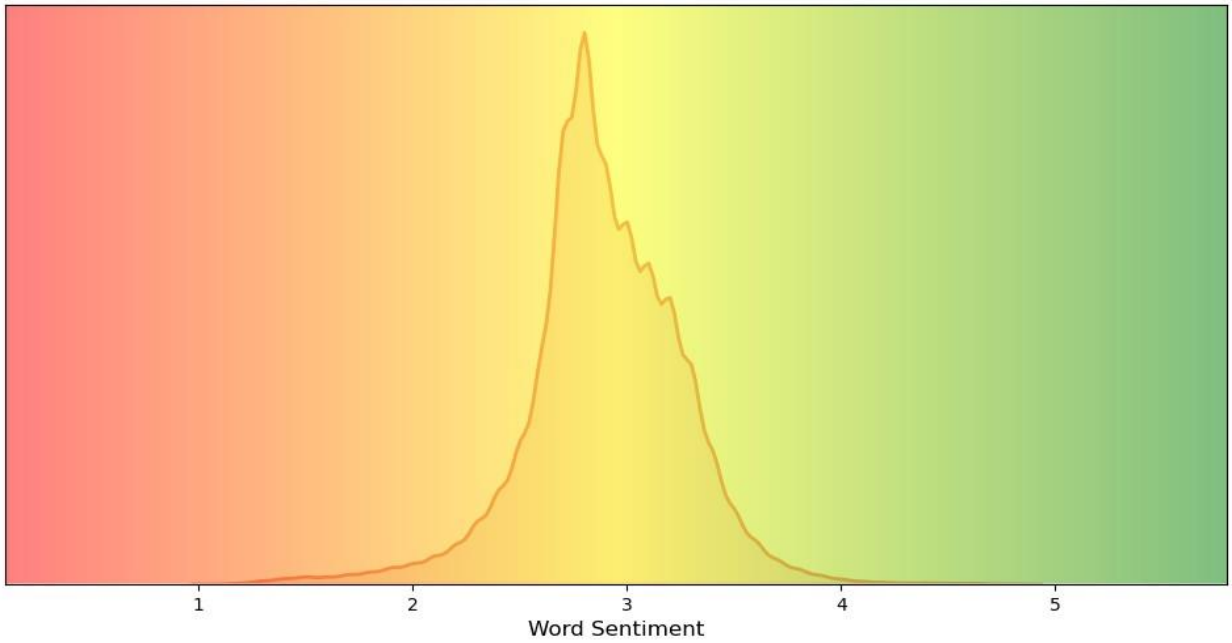

Figure 1 generally follows a bell curve, with a slight skew to the more negative word sentiment.

In contrast, Figure 2 shows how sentiment was applied in the dictionary by its starting letter:

Figure 2: Sentiment Scores Alphabetically

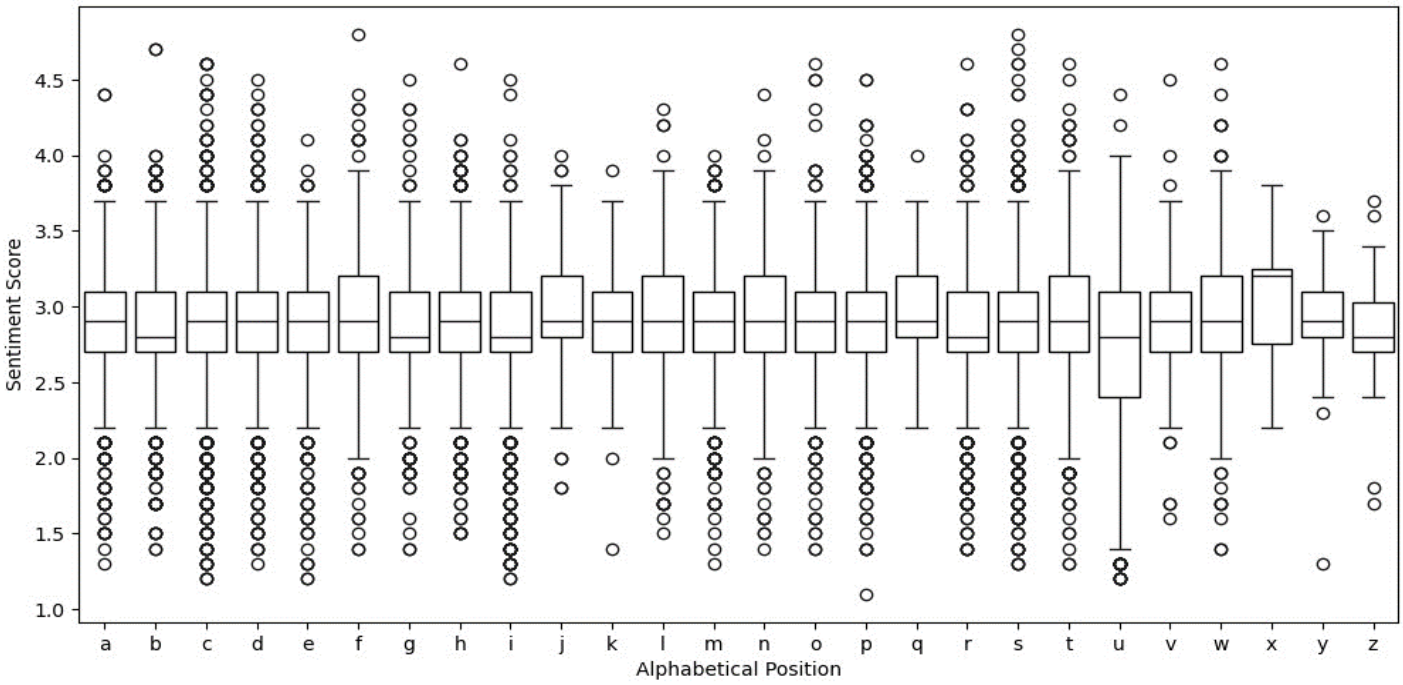

Figure 2 shows a generally similar pattern between the letters and a range of roughly 2 to 4 on word sentiment, with plenty of outlier words outside this. One exception was the letter "u," which I assume, based on Figure 3, is that the "un-" prefix additions to words cause their sentiment to be much lower than other letters:

Figure 3: Top and Bottom 10 Words by Sentiment

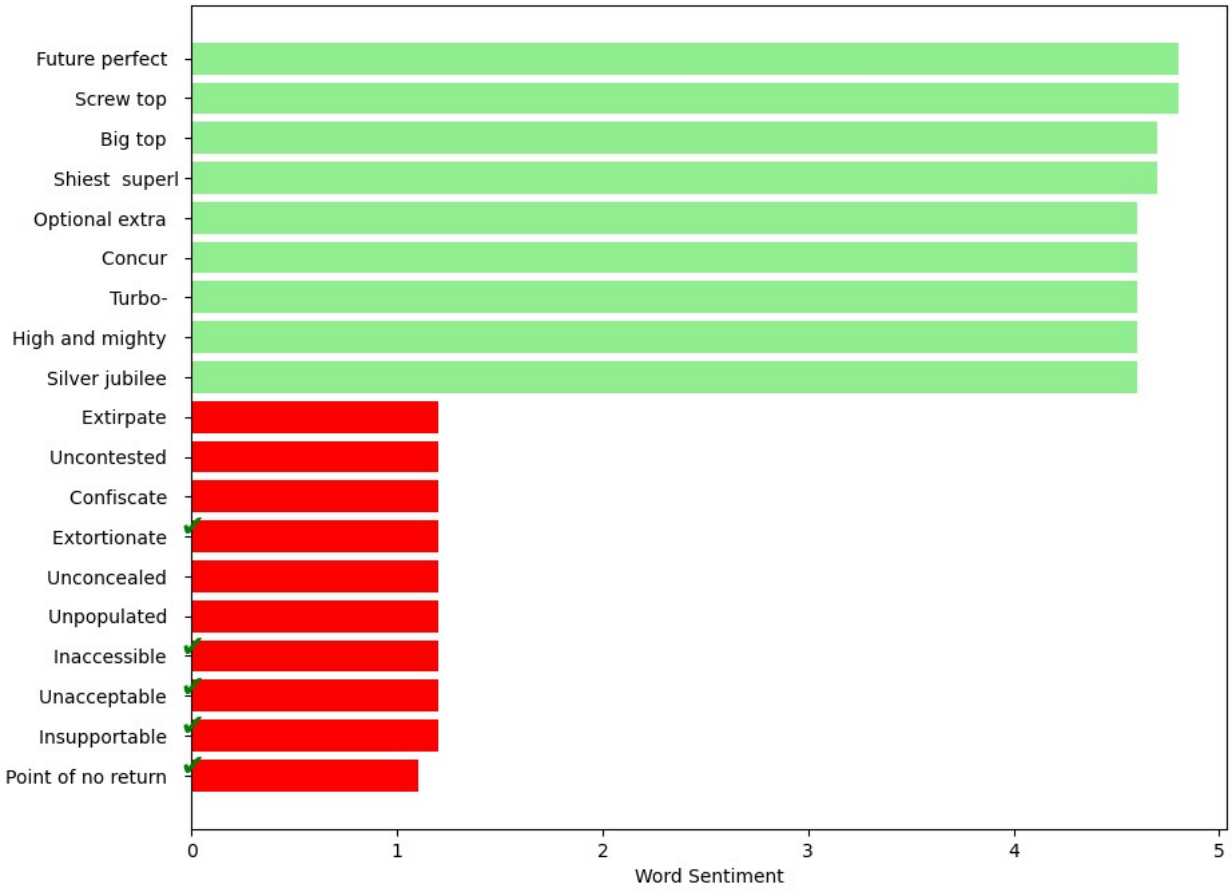

Figure 3 presents the top and bottom ten words based on sentiment. Interestingly, the bottom words appear more accurate than the top words. It's amusing to find words like "Screw top" and "Silver jubilee" ranking high, which one wouldn't expect. However, "jubilee" and "top" may have contributed to their higher ranking. This demonstrates the significance of using "Definition" for sentiment scoring rather than "Word", as each definition plays a significant context.

Figure 4: Word Clouds for Different Sentiment Groups

In Figure 4, the word cloud plot is divided into four groups based on the sentiment scores of words grouped from [1-2), [2-3), [3-4), and [4-5). Similar to Figure 3, negative word sentiments appear to be more accurate with words like "acid," "disease," "sickness," "disease," "cold," and others. "Return" is usually not negative, but it's considered negative in the context of the model's

sentiment analysis of Amazon product reviews. On the other hand, the top sentiments show more dubious 'high' sentiment words such as "Wheel," "Controller," "Command," and "Shooting."

Text Mining Discussion

This project has been incredibly informative, from learning how to pre-process text, using pre-trained transformers, and interpreting and visualizing text mining nuancedly. Choosing the Oxford Dictionary was a perfect project choice, too. Despite its somewhat plain appearance, it encapsulates the English language and benchmarks relatively reasonably how a model behaves.

For instance, I learned that calculating sentiment scores on the `Definitions` would have been more informative, albeit slightly more challenging to implement. Definitions provide a complete context that would have led to more accurate sentiment scores, particularly positive sentiment, especially as the model was trained on descriptive and multi-sentence product reviews. However, at the time, I opted to use `Word` in the sentiment computing process due to time constraints and the risk of exceeding token character limits. `Word` was a safer and more straightforward option as it did not risk causing a token limit error. While capping max tokens to a limit (e.g., 512 tokens) would have been a viable solution, I chose `Word` for simplicity since this was all new.

Analyzing sentiment with transformers was the time-consuming part of the project, requiring thorough planning and creativity to resolve. In the future, to apply NLP faster:

1. When I have access to multiple CPUs, it would have been helpful to have a combined network folder for multithreading even faster, as my code could recognize if a file existed already in a directory based on unique filenames (e.g., "Skipping sentiment_WORD_Ab as it already exists"). This could be considered a future "big data" text mining improvement, as I learned about document sharing with DS730.

2. Likewise, I faced a 20-hour delay during my NLP project due to using only my CPU. To speed up processing, I would like to explore using Nvidia's CUDA platform, which enables GPUs for faster training and inference, as I know PyTorch supports CUDA (PyTorch, 2023). While I could not configure CUDA for this project, I aim to use it for future projects in Python to speed up GPU-enabled multithreading.

In diving into text mining complexities with the Oxford Dictionary, this project provided valuable insights into pre-processing, pre-trained transformers, and a nuanced interpretation of linguistics. Focusing on sentiment analysis, particularly by `Word` rather than `Definitions`, revealed a pragmatic trade-off between simplicity and helpfulness, subject to constraints (e.g., time, hardware). Likewise, multithreading can improve efficiency for faster NLP processing with multiple CPUs and explore GPU acceleration through Nvidia's CUDA platform. In conclusion, the project highlights the active core of text mining, where design choices (e.g., NLP model, "cuda" vs. "cpuonly") significantly affect outcomes and how creative application of transformers may open doors to exciting possibilities.

References

(2023, 11 29). Retrieved from PyTorch: https://pytorch.org/get-started/locally/

Peirsman, Y. (2023, 11 29). *Nlptown/bert-base-multilingual-uncased-sentiment · hugging face*. Retrieved from Hugging Face: https://huggingface.co/nlptown/bert-base-multilingual-uncased-sentiment

Python regex. (2023, 11 28). Retrieved from https://www.w3schools.com/python/python_regex.asp

Sujith. (2023, 11 29). *Oxford English Dictionary*. Retrieved from GitHub: https://raw.githubusercontent.com/sujithps/Dictionary/master/Oxford%20English%20Dictionary.txt