**# Task and Data Analysis**

In the task assigned to me, I faced various challenges due to the limitations of the data provided. The issues primarily stemmed from inconsistencies in tagging conventions and poorly labeled data. This hindered the evaluation process as accurately assessing and interpreting the data, leading to some inaccuracies and inefficiencies in the results derived from the analysis.

Key limitations of the data included:

- \*\*Inconsistent Naming Conventions\*\*: The tags varied significantly, making it difficult to evaluate consistently. For example, tags like “Zoom” and “Zoom mode,” “dvd player” and “player,” and “Universal remote control” and “remote” indicate a lack of standardisation.

- \*\*Poor Labeling\*\*: Some items were incorrectly labeled, such as labeling a “camera” to have a “door” feature. These inaccuracies questioned the reliability of the labels as a source of truth.

- \*\*Raw Text Format\*\*: The data required manual parsing, which is more error-prone compared to structured formats that could be easily read by built-in parsers, like CSV.

- \*\*Complexity Added by Titles\*\*: The inclusion of titles along with the body text in some reviews added an extra layer of complexity to the data parsing process.

- \*\*Varying File Lengths\*\*: Files varied in length. I tended to prefer larger files for their more substantial data volume, which I believed would provide a more robust basis for prediction.

To handle these issues, I developed a pipeline controlled by my `opinion\_miner\_controller` function. This pipeline is executed in the final two cells of my notebook, where the opinion miner runs and outputs various samples. The process includes several critical functions:

1. `read\_file`: My data parser manages data nuances, such as titles, annotations, and special string characters. It separates text data from tags and sentiments, organising it into a structured Pandas dataframe.

2. `pre\_processing\_controller`: This function prepares the data for analysis by cleaning text, processing stop words, tokenising, lemmatising, and chunking nouns. Feature normalisation for machine learning is a part of pre-processing, but run outside of the main pipeline.

3. `feature\_extraction`: Extracts features from the parsed review based on the pre-processing string type, a similarity threshold between product and features, and a choice between two feature extraction models. gloVe and Word2Vec similarity models are used to assst in extracting nouns with a relation to the product.

5. `sentiment\_controller`: Uses parameters like the sentiment classifier and the pre-processing review string type to apply either a Vader / TextBlob classification or a SentiWordNet classification.

6. \*\*Further Processing\*\*: Includes creating feature table dictionaries and mapping dictionaries to align similar words (e.g., 'picture' and 'pic') to their correct tags for evaluation.

7. \*\*Output\*\*: Outputs include a confusion matrix, a metrics table showing precision, accuracy, recall, and the F1 score, and a feature table from the miner.

Additionally, outside the `opinion\_miner\_controller` function, I developed functions to optimise the miner’s performance:

- `average\_metrics`: Averages evaluation metrics across three sample files.

- `sentiment\_model\_average\_comparison`: Compares the performance of two sentiment models.

- `noun\_model\_comparison`: Shows differences between two noun extraction methods.

- `show\_optimum\_string\_variables`: Identifies the best pre-processed strings for the miner.

- `sim\_filter`: Finds the optimal similarity parameter for feature extraction.

- `build\_ml\_classifier` and `evaluate\_ml\_classifier`: An alternative approach using a OneVsRestClassifier to both extract and classify features.

**# Data Pre-Processing**

The data parsing process initiates with the `read\_file` function, designed to effectively handle and extract pertinent content from data files. This function begins by reading the file, identifying, and excluding metadata indicated by a line of asterisks. It then addresses complexities such as titles marked by `[t]`, which are handled by the `handle\_titles` function by appending the title to the corresponding review to preserve context and maintain data integrity.

Reviews are subsequently split using '##' as a delimiter to separate tags, which may contain embedded metadata and sentiment scores, from the main review content. The extracted tags and text data are then organised into a Pandas dataframe.

This dataframe is processed by the `pre\_processing\_controller`, which includes several key functions:

- \*\*Tokenised\_Review\*\*: Tokenises reviews and preserves the integrity of compound phrases and adjectives directly linked to nouns using the `preserve\_compound\_phrases` function. This function constructs compound phrases by concatenating related words with an underscore, thus preserving semantic relationships within the text.

- \*\*Soft\_Filtered\_Review\*\*: Cleans up the Tokenised\_Review by removing numerical characters, punctuation, and normalising capitalisation.

- \*\*Soft\_Filtered\_Review\_String\*\*: Converts Soft\_Filtered\_Review to a string and processes it through `chunking\_post\_process` to further preserve compound phrases.

- \*\*Filtered\_Review\*\*: Applies more aggressive filtering than Soft\_Filtered\_Review by removing stopwords, using the `nltk.corpus` library.

- \*\*Lemmatised\_Review\_String\*\*: Lemmatises reviews using the `Spacy` NLP library to reduce word dilution and enhance uniformity across the text data. Lemmatisation is a valuable pre-processing step that reduces words to their base forms, simplifying text complexity. Lemmatisation improves data consistency, by standardising words.

Additionally, the `parse\_and\_normalise\_tags` function parses annotated tags into a machine-readable format and normalises features to create a vector matrix, facilitating further machine learning applications.

In advancing natural language processing (NLP) methodologies, I have incorporated both Part-of-Speech (POS) tagging and dependency parsing into my research framework to enhance text analysis capabilities. Initially, the process begins with POS tagging, a fundamental NLP technique where each word in a text is meticulously tagged with its grammatical role using the `pos\_tag` function from the NLTK package. This tagging not only considers the word itself but also its context within the sentence—factors like the word's positioning relative to others and the overall sentence structure—which significantly enhances the precision of the tagging process. Following this, the text undergoes noun extraction, where words tagged with specific noun labels such as 'NN', 'NNS', 'NNP', 'NNPS' are retained. These nouns are then subjected to frequency analysis, identifying the 15 most common nouns for further analysis or processing.

Building upon this foundation, I developed a dependency parser as a more advanced feature extraction model, which surpasses traditional POS tagging by constructing a detailed dependency tree. This tree visually maps the relationships between words in a sentence, using arrows to denote dependencies and illustrate which words depend on others. This approach not only provides a comprehensive map of word interactions but also significantly enhances semantic understanding. According to the Stanford CS224n course on dependency parsing, this technique excels in accurately identifying grammatical roles and relationships, making it particularly effective in clarifying complex constructions like passive voice and nested phrases. This clarity is crucial for accurately interpreting sentences and extracting meaningful information, which aids in tasks like sentiment analysis and information extraction. Additionally, dependency parsing plays a critical role in Semantic Role Labeling by assigning semantic roles to phrases based on their function in the main action of the sentence. Its linguistic consistency across different languages also bolsters its value, enabling the development of robust multilingual NLP applications. By enhancing both the depth and accuracy of semantic analysis, dependency parsing serves as a foundational tool for advanced NLP applications, offering a nuanced and dynamic understanding of text structure and connections.

This function is designed to refine a list of nouns, which are presumed to be features of a particular product, ensuring that only the most relevant and distinct nouns are retained. Initially, the product is identified as the first noun in the list, given that the list is sorted by noun relevance. The function then extracts words from a list of tuples, discarding any secondary values. It calculates the semantic similarity of each word to the product using both Word2Vec and GloVe models, and an average of these similarities determines the relevance of each noun. Words that meet a predefined similarity threshold are retained for further consideration. To enhance the uniqueness of the selected features, the function applies a secondary filtering process. This process removes synonyms or near-synonymous words—such as 'pic' and 'picture'—by examining the similarity scores from GloVe's top similar words and excluding those that are highly similar. By averaging results from two distinct NLP models, the approach not only diversifies the natural language processing techniques but also ensures that the final list of nouns is both concise and accurately reflective of the product's attributes, minimising redundancy and enhancing feature relevance.

References needed:

* glove
* word2vec
* dependency parsing
* pos tagging
* vader classification (remove one)
* text blob classification (remove one)
* senti classification
* ml model