# 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 a Vader or Senti 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.

I also undertook some further research to see if I could build on my Opinion Miner pipeline, this can be found under the \*\*Further Work\*\* section. Here I built A ML classifier, this pipeline consists of three functions: `parse\_and\_normalise\_tags`, `build\_ml\_classifier` and `evaluate\_ml\_classifier`.  
  
  
  
  
# 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.

#### References

- https://spacy.io/models/en/  
  
  
  
# Product Feature Extraction

The `feature\_extraction` function is the controller function. It is passed three parameters:

- The `noun\_comparison\_flag` switches between the two feature extraction models during evaluation.

- The `noun\_string` selects which pre-processed review to use

- The `similarity\_threshold` defines the threshold to filter out dissimilar words

`POS\_Noun\_Tagging` - To extract features from the reviews, I have incorporated Part-of-Speech (POS) tagging, a NLP technique where each word in a text is tagged with its grammatical role using the `pos\_tag` function from the NLTK package. This tagging considers the word itself and also its context within the sentence. It factors the word's positioning relative to others and the overall sentence structure, which enhances the precision of the tagging process. The text then 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. The 15 filter was manually chosen through analysing various compund noun outputs.

`dependency\_parsing\_noun\_extraction` - I also 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 provides a comprehensive map of word interactions and strongly 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. Dependency parsing also 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 strengthens its value.

`similarity\_filter` - This function is designed to refine a list of nouns, 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. The models averaging adds a level of diversification. The 15 most common nouns are selected also.

#### References

- https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00561-y

- https://towardsdatascience.com/natural-language-processing-dependency-parsing-cf094bbbe3f7

- https://web.stanford.edu/class/cs224n/readings/cs224n-2019-notes04-dependencyparsing.pdf

- https://wiki.pathmind.com/word2vec

- https://www.geeksforgeeks.org/pre-trained-word-embedding-using-glove-in-nlp-models/

- <https://towardsdatascience.com/a-comprehensive-python-implementation-of-glove-c94257c2813d>

"""

These functions are utilised within the opinion\_miner\_controller module.

The primary function, create\_feature\_table, takes a DataFrame and a list of features as input.

It processes the DataFrame, extracting features and sentiments, then updates a dictionary with the feature counts.

Each review's features and sentiments are converted into tags, following a specific format for labeling data.

These tags are added to the main DataFrame under the column 'My\_Sentiment\_Tags' for later evaluation.

"""  
  
  
# Sentiment Analysis

The `sentiment\_controller` function inputs the data dataframe, the classifier and the string type to in which to formulate the sentiment.

Each feature that has been labelled will be passed individual for sentiment classification, for each review in the file.

I have selected two models for sentiment analysis, `SentiWordNet` and `VADER`, each with distinct strengths and applications.

#### SentiWordNet

SentiWordNet is an advanced lexical resource that augments the traditional WordNet database by integrating three types of sentiment scores: positivity, negativity, and objectivity, into each synset. These sentiment scores, ranging from 0 to 1, describe how objective, positive, and negative the terms within the synset are, determined through semi-supervised learning techniques and manual adjustments to ensure an accurate reflection of word sentiments. SentiWordNet enhances text representations by adding sentiment related properties of terms. The development of SentiWordNet involves the quantitative analysis of glosses related to synsets and vectorial term representations. This process uses a committee of eight classifiers with similar accuracy levels but differing behaviours to derive the sentiment scores, thereby providing detailed word sense representation and extensive coverage of over 115,000 WordNet synsets. However, SentiWordNet has limitations, including static sentiment scores that may not reflect changing contextual meanings, limited coverage of modern slang or newly coined terms, and potential inaccuracies in capturing the sentiments of polysemous words due to its reliance on generalised word usage.

#### VADER

To address some of the limitations seen in SentiWordNet, particularly in handling dynamic and informal text, VADER (Valence Aware Dictionary and sEntiment Reasoner) presents it self as a strong alternative. Recognised for its ability to efficiently process and analyse large volumes of text data, VADER is particularly valuable for evaluating customer feedback on social media, which may well translate to online produt reviews. Its strength lies in managing texts that feature unconventional forms, such as emojis, varied punctuation, and internet specific expressions. The core of VADER's functionality is its sentiment lexicon, developed through crowdsourced input via Amazon’s Mechanical Turk. This approach ensures the lexicon is not only comprehensive but also reflective of modern language and expressions typical in social media communications. Unlike SentiWordNet, VADER is stronger when understanding context, assessing how sentiments are influenced by how something is said through capitalisation, punctuation, or other stylistic nuances rather than solely by what is said. This sophisticated context understanding makes VADER especially effective in sentiment analysis, providing a deep insight into both explicit and implicit sentiments embedded within digital communications.

#### References

- https://ontotext.fbk.eu/sentiwn.html

- https://srish6.medium.com/sentiment-analysis-using-the-sentiwordnet-lexicon-1a3d8d856a10

- https://www.analyticsvidhya.com/blog/2021/06/vader-for-sentiment-analysis/

- https://towardsdatascience.com/an-short-introduction-to-vader-3f3860208d53

- https://vadersentiment.readthedocs.io/en/latest/

- <https://www.analyticsvidhya.com/blog/2021/06/vader-for-sentiment-analysis/>

# Further Work

### A OneVsRestClassifier Logistic Regression Classifier

\* The `parse\_and\_normalise\_tags` function parses annotated tags into a machine-readable format and normalises features to create a vector matrix.

\* The \*\*MultiLabelBinarizer\*\* from \*\*sklearn.preprocessing\*\* transforms the list of tag-sentiment combinations into a binary format. Each unique tag-sentiment combination across all entries becomes a feature column, with each row in the output matrix Y indicating the presence (1) or absence (0) of that tag-sentiment in a specific review.

\* The \*\*MultilabelStratifiedShuffleSplit\*\* from the \*\*iterstrat.ml\_stratifiers\*\* package splits the data into training and test sets while maintaining the same proportion of each label in both sets, ensuring all labels are adequately represented.

\* A logistic regression model, wrapped in a \*\*OneVsRestClassifier\*\*, is employed to train a separate binary classifier for each label. The model predicts the probability of a review being associated with each tag-sentiment based on its TF-IDF features.

\* Class weights are calculated using the \*\*compute\_class\_weight\*\* function, which assigns weights inversely related to class frequencies. These weights are incorporated into the logistic regression model to adjust the model’s focus, ensuring that errors involving minority classes are more heavily penalised during training.

\* After prediction, the \*\*mlb.inverse\_transform\*\* method converts the predicted binary labels back into the tag-sentiment format, making the output more interpretable by translating the binary predictions back into the original tags and sentiments.

#### References

- https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html

- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MultiLabelBinarizer.html

- https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/

- https://blockgeni.com/using-one-vs-rest-and-one-vs-one-for-multi-class-classification/

- <https://machinelearningmastery.com/cost-sensitive-logistic-regression/>

"""

Processing functions to:

- Filter for the models sentiment tags and the data's sentiment tags

- Extracts the data's sentiment tags as a dictionary

- Creates a mapping dictionary that maps the models features with the datas features, with similarity variability

- Create a dataframe for the models labeled reviews and the datas labeled reveiws for evaluation

"""

"""

Processing functions to create the confusion matrix, calculate the metrics from the matrix and output a matrix comparison chart

"""

"""

Main controller function of the Opinion Miner

Arguments:

- File: The file name to be parsed

- sentiment\_classifier: A string to define which sentiment classifer should be selected

- noun\_string: Parameter to select which pre-processed review type to use for feature extraction

- sent\_string: Parameter to select which pre-processed review type to use for sentiment analysis

- similarity\_threshold: Parameter used during feature extraction to define how close the feautre list should be to the predicted name of the product

Returns:

- conf\_matrix\_df: A confusion matrix comparing my models feature classifications with the provided data set

- metrics\_df: A table displaying the precision, recall, accuracy and F1 Score

"""

"""

The method I used to see which similarity filter to use in my noun feature processing

"""

"""

The method I used to see which pre-processing review string combination to use when extracting features and classifying

"""