ASPECT BASED SENTIMENT ANALYSIS

Melinda Dong | a1870910@adelaide.edu.au

1. OVERVIEW

The aim of this project is to build an aspect-based sentiment analysis (ABSA) system. based on syntactic parsing. The aspect-based sentiment analysis tries to analyse the sentiment, positive, negative, and neutral. Because if one aspect term is not positive, it will not necessary be negative or neutral, that's why we do these 3 parts separately.

There are 2 common syntactic parsing method: dependency parsing and constituency parsing. Dependency parsing tends to capture more fine-grained relationships between words. Constituency parsing, on the other hand, provides a higher-level view of the sentence structure, focusing on larger constituent phrases. Therefore, the project chooses dependency parsing, because it's more suitable for analysing specific aspects or entities within a sentence.

The project built 3 rules for positive sentiment detection. 4 rules for negative and 4 rules for neutral sentiment detection.

Since all the rules are manually build and there are various sentence expressions, it's difficult to cover all those expressions, the result didn't work as better as other machine learning method, the final F1 sore for positive sentiment detection is 0.716, for negative sentiment detection is 0.446 and 0.431 for neutral.

2. DATA PREPROCESSING

2.1 Data overview

The data we are used is `Restaurants.xml`, it contains 3041 reviews in total, for each review it contains the text, aspect term, and aspect category. For easy analysis, the original extensible markup language(.xml) format was converted into data frame.

Since the project only focuses on detect aspect term, so those reviews without aspect terms were removed and there are 2021 reviews left.

In those 2021 reviews, each review might contain more than one aspect term, there are 3693 aspect terms in total, 2164 positive, 805 negative and 724 neutral.

2.2 Data pre-processing

Before starting syntactic parsing, there are some words with "-", for example, "family-style", when doing parsing it would separate it as 2 tokens. So, I replace all "-" into "_" to keep the parsing neat.

Also, there are some aspect term contains more than one token, for example “orrechiete with sausage and chicken”, it supposed to be one item, so I check all the aspect terms, replace all “ ” into “_”, so when doing parsing, it will see the whole aspect term as one item.

2.3 Helper functions and lexicon

The main workflow is to check the dependency parsing table and graph of the reviews one by one and try to find some common rules to connect the aspect term with their corresponding key words (those words that contains sentiment information). Some helper functions were built to better extract rules, for example “extract_text_by_id”, “into_parsed_table” and “into_parsed_pic”.

Another important helper function is `is_neg_dep`, this function detects in aspect term’s children, head and sibling see if there’s some `neg` depency relationship, such as “no”, “not”, “never”, “neither”.

For the emotional judgment of a word, I tried 2 lexicons, one is Vader lexicon, for one word, Vader lexicon returns a dictionary contains the probability of that word being positive, negative, or neutral (compound were removed for convenience). Another is Opinion lexicon. It contains 2 words list for common positive and negative words. I tried both lexicons, and Opinion lexicon gives better result in these cases.

3. POSITIVE RULES

*In the early stages of exploration of the project, I literately check reviews and make new rules for new cases that couldn’t fit the existing rules, this way is too tedious, and somehow there would be overlapped between rules, after building 8 rules for positive sentiment detection, when combined all rules together, it still didn’t perform well. So, I changed my method, I roughly build a structure between rules by the where is the direction to searching aspect term’s corresponding key words, that are:

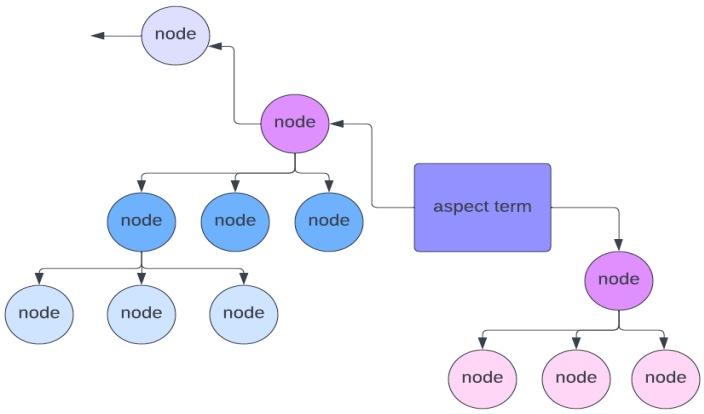
-- searching the first-level neighbors of the aspect term in the dependency tree, for example, direct head or direct children. Under some conditions, searching sub-children of the child.

-- searching the sibling nodes (those nodes have same direct head with aspect term) and sub children of siblings.

-- under some conditions, searching aspect term’s further ancestors.

There are some various in the rules for positive, negative, and neutral sentiment detections, but the same searching structures are applied for all of them.

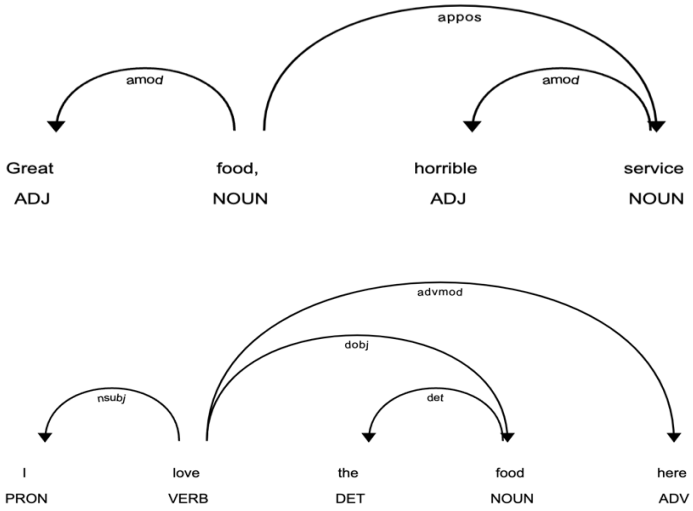
[see graph below, the pink nodes are searched in rule1, blue nodes are searched in rule2, and light purple nodes are searched in rule 3.]



3.1 Positive rule 1

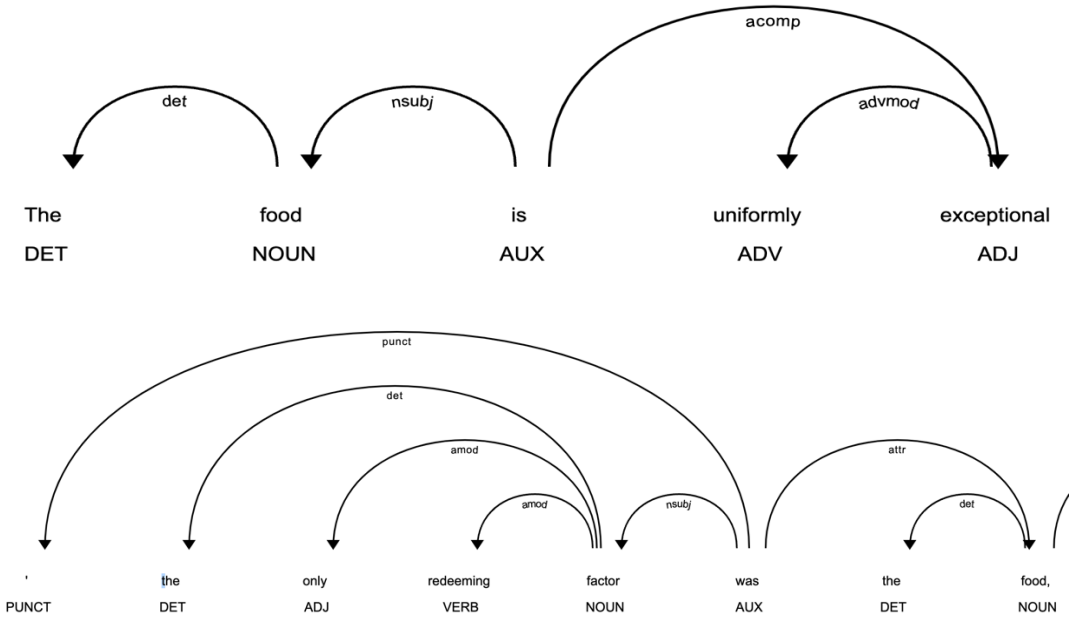
If there's no `neg` relationship around the aspect term, the first positive rule is to search the first-level neighbors(head/children) of the aspect term in the dependency tree see if can find positive words, if couldn't found, expand the searching scope to the sub children of the aspect term.

Because from observe, in most cases, the word describes the aspect term is normally adjacent in location to the aspect term. Here are some examples that can be covered by positive rule1:



3.2 Positive rule 2

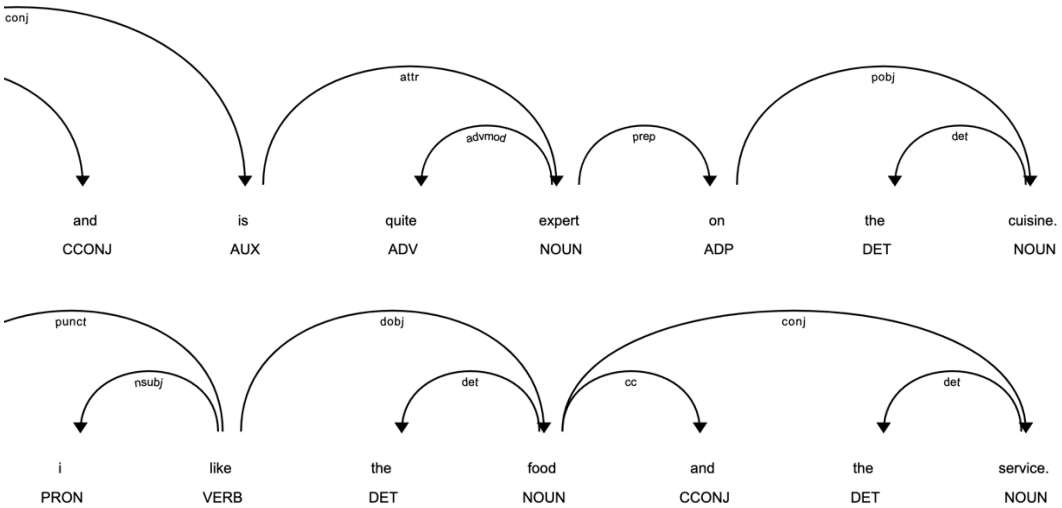
If there's no `neg` relationship around the aspect term, the second positive rule is to check aspect term's sibling nodes and the children of the sibling nodes, if it's the node's dependency relationship is ['amod','advmod','acomp','attr','nsubj`], then check if the node is in positive words list, if yes, return True. Here are some examples that can be covered by positive rule2:



3.3 Positive rule 3

If there's no `neg` relationship around the aspect term, the third positive rule is to check aspect term's ancestors, while the dependency relationship is in ['conj','compound','prep','pobj'], keep searching the ancestor of the node till the denpendency relationship is not in the list. Then check if the node is in positive words list and if the node's children in positive words list.

Here are some examples that can be covered by positive rule3:



3.4 Positive rules evaluation

There are 3 evaluation metrics: recall, precision and F1 score. Recall means in all the true positive terms, how many of them being successfully detected; precision means in all those positive predictions, how many of them are truly positive; F1 score is a combine of recall and precision. Here is the formula of those metrics:

Recall = TP/ (TP + FN)

Precision = TP/ (TP + FP)

F1 = 2*precision*recall / (precision + recall)

The final positive result is just simply stacking 3 rules together, the check order is their precision, in this case, check if rule2 or rule1 or rule3 is True, is one of them is True, return True, otherwise, return as others and here is the result.

	recall	precision	F1 score
Positive rule1	0.322	0.854	0.468
Positive rule2	0.541	0.860	0.664
Positive rule3	0.463	0.834	0.596
Final positive	0.686	0.812	0.743

4. NEGATIVE RULES

4.1 Negative rule 1,2,3

The first 3 of negative rules are the same with positive rules, just changed the checking words list from positive to negative. Briefly introduce here what are the first 3 rules:

Negative rule1: searching the first-level neighbors(head/children) of the aspect term in the dependency tree see if can find node in negative words list, if couldn't found, expand the searching scope to the sub children of the aspect term.

Negative rule2: checking aspect term's sibling nodes and the children of the sibling nodes, if it's the node's dependency relationship is in ['amod','advmod','acomp','attr','nsubj`], then check if the node is in negative words list, if yes, return True.

Negative rule 3: checking aspect term's ancestors, while the dependency relationship is in ['conj','compound','prep','pobj'], keep searching the ancestor of the node till the dependency relationship is not in the list. Then check if the node is in negative words list and if the node's children in negative words list, if yes, return True.

4.2 Negative rule 4

The 4th negative rule is negation rule, it just simply uses the helper function `has_neg_dep` to detect negative sentiment. For positive rules, negation checking was merged with every step, somehow, experiment shows extract negation rule separately gives slightly better performance for detecting negative sentiment.

Also, experiment shows adding extra conditions such as, "if there's a negation and there's positive words, then return True" perform less good then "if there's a negation then return True". Because it can't cover some sentences that contains negation but not positive words, such as "don't go there" or "I'm not recommend the beef". Of course, it will give misjudgement on expression as "I don't like the beef". It really depends on which kind of expression is more often. The experiment shows simply use `has_neg_dep` to detect negative sentiment gives better performance.

4.3 Negative rules evaluation

The final negative result is just simply stacking 4 rules together, the check order is their precision, in this case, check if rule2 or rule1 or rule3 or rule4 is True, is one of them is True, return True, otherwise, return as others and here is the result.

	recall	precision	F1 score
Negative rule1	0.112	0.577	0.187
Negative rule2	0.240	0.659	0.352
Negative rule3	0.207	0.553	0.302
Negative rule4	0.103	0.535	0.173
Final negative	0.405	0.553	0.468

5. NEUTRAL RULE

5.1 Neutral rule 0

Directly detect neutral sentiment from sentences is difficult, it doesn't have any specific pattern, for example "I asked for seltzer_with_lime, no ice." So neutral rule0 reuse all the positive rules and negative rules above, if it couldn't be predicted as positive or negative, it returns as neutral.

5.2 Neutral rule 1,2,3

similarly neutral rule 1, 2, 3 are like the first 3 positive rules, just changed the checking words list from positive to negative. There's no existing neutral words list, so I manually build the neutral words list, since it's manually built, it only contains 14 words, this would affect the performance of neutral 1,2,3. Vader lexicon was tried as well, but it has problems as well, it will define the words without sentiment information as neutral as well, such as "table", "was". Therefore, manually built neutral words list is adopted.

Briefly introduce here what are the 3 rules:

Neutral rule1: searching the first-level neighbors(head/children) of the aspect term in the dependency tree see if can find node in neutral words list, if couldn't found, expand the searching scope to the sub children of the aspect term.

Neutral rule2: checking aspect term's sibling nodes and the children of the sibling nodes, if it's the node's dependency relationship is in ['amod','advmod','acomp','attr','nsubj`], then check if the node is in neutral words list, if yes, return True.

Neutral rule 3: checking aspect term's ancestors, while the dependency relationship is in ['conj','compound','prep','pobj'], keep searching the ancestor of the node till the dependency relationship is not in the list. Then check if the node is in neutral words list and if the node's children in neutral words list, if yes, return True.

5.3 Neutral rules evaluation

The final neutral result is just simply stacking 4 rules together, the check order is their precision, in this case, check if rule2 or rule1 or rule3 or rule0 is True, is one of them is True, return True, otherwise, return as others and here is the result.

	recall	precision	F1 score
Neutral rule1	0.015	0.440	0.029
Neutral rule2	0.031	0.523	0.060
Neutral rule3	0.015	0.440	0.029
Neutral rule0	0.646	0.328	0.435
Final neutral	0.669	0.332	0.443

6. CONCLUSION

Here is the recap of the final performance:

	Positive rules	Negative rules	Neutral rules
recall	0.686	0.405	0.669
precision	0.812	0.553	0.332
F1 score	0.743	0.468	0.443

From the result, the overall result isn't very well. Positive rules are better than negative and neutral rules, it might because people tend to be more direct when they give good reviews and more tactful when they post bad reviews, for example "All the money went into the interior_decoration, none of it went to the chefs." And "Our server checked on us maybe twice during the entire meal." As for neutral sentiment, the directly detecting rules (neutral rule1, 2, 3) can only detect very few neutral reviews. Neutral rule0 can detect more reviews, but it was based on previous positive and negative rules, so it kind being dragged back of the poor negative rule performance.

The result shows doing aspect-based sentiment analysis only based on synthetic parsing is not enough. Here might be some of the reasons that lead the poor performance of dependency parsing in this case:

1. Limited understanding of the full meaning: Dependency parsing focuses on grammar and word relationships, but it can't capture the complete meaning and nuances needed for sentiment analysis. Some sentences have no obvious emotional vocabulary, it requires understanding the overall context and the sentiment associated with specific aspects to make a correct prediction.

2. Difficulty in understanding word meanings: Dependency parsing can struggle with understanding the different meanings of words. Some words are ambiguous, and some words will use positive words to express negative meaning, and vice versa, some expressions use slang and idiomatic expressions or metaphorical expression, all these cases would be difficult to handle.

3. Lack of domain-specific knowledge: Dependency parsing, in its basic form, may not incorporate domain-specific knowledge or specialized language used in specific fields. The lexicon I used is designed for general domains, even after I manually

update the words list, still it can cover all the possible sentiment lexicon for restaurant.

To improve the performance of aspect-based sentiment analysis, it is often necessary to combine dependency parsing with other techniques, such as machine learning-based approaches, domain-specific sentiment lexicons, or context-aware models, which can capture the semantic and sentiment-related aspects more effectively.

7. APPENDIX

Some of the dependency type that is mentioned in project, more about dependency type in this web: https://universaldependencies.org/u/dep/index.html

1. `amod` (Adjectival Modifier): This relation represents an adjectival modifier of a noun. It connects an adjective to the noun it modifies. For example, in the phrase "big house," the word "big" is an adjectival modifier of the noun "house."

2. `advmod` (Adverbial Modifier): This relation represents an adverbial modifier of a verb, adjective, or other adverb. It connects an adverb to the word it modifies. For example, in the sentence "She sings beautifully," the word "beautifully" is an adverbial modifier of the verb "sings."

3. `acomp` (Adjectival Complement): This relation represents an adjectival complement of a verb. It connects an adjective to the verb it complements. For example, in the sentence "He is happy," the word "happy" is an adjectival complement of the verb "is."

4. `attr` (Attribute): This relation represents the attribute of a copular verb. It connects a noun or adjective to the copular verb, indicating the property or identity of the subject. For example, in the sentence "She is a doctor," the noun "doctor" is the attribute of the copular verb "is."

5. `nsubj` (Nominal Subject): This relation represents the nominal subject of a clause. It connects a noun or pronoun to the main verb of the clause, indicating the entity performing the action. For example, in the sentence "John eats an apple," the word "John" is the nominal subject of the verb "eats."

6. `conj` (Conjunct): This relation represents a conjunct, which is a word or phrase that is connected to another word or phrase in a coordinating fashion. It connects multiple elements that have the same syntactic function. For example, in the sentence "She is smart and talented," the words "smart" and "talented" are conjuncts connected by the conjunction "and."

7. `compound` (Compound): This relation represents a compound word, which is a combination of multiple words that function as a single unit. It connects the constituent words of the compound. For example, in the word "laptop computer," the words "laptop" and "computer" are compounds connected by the relation "compound."