Data & AI 6

# Aspect Based Sentiment Analysis Project Report

Team 29

Member 1: Yiannis Ftiti
Member 2: Yiannis Ftiti

Karel de Grote Hogeschool

28/10/2025

# Introduction

Aspect Based Sentiment Analysis is a form of sentiment analysis that identifies specific *aspects* (features, topics, or attributes) in a text and determines the sentiment expressed toward each aspect.

For example:

> “The pizza was delicious but the service was terrible.”
>
> - *pizza* → **positive**
> - *service* → **negative**

This project implements and compares 3 different implementations:

1. A **rule based lexicon model** using spaCy and VADER.
2. A **transformer based model** using a pretrained model(DeBERTa v3).
3. An **LLM based model** using local LLMs via Ollama (Phi-3 and Mistral 7B).

All models implement a unified API, defined in base.py, as per the project's requirements.

# Methodology

This project implements 3 ABSA methods:

## Lexicon Based

This is a rule based, linguistic approach built using ***spaCy*** for syntactic parsing and ***VADER Sentiment*** for sentiment scoring.

Process Steps:

1. We extract the aspects: The text is parsed with spaCy, and each token that is a noun or proper noun (NOUN, PROPN) is treated as a potential aspect (like “*battery*”, “*service*”).
2. Find the opinion word tied to each aspect: For each aspect word, the algorithm looks at how it connects to other words in the sentence to find opinions about it. It checks for adjectives that describe the noun (*amod*), adjectives linked through helper verbs like “is” or “was” (*acomp*), verbs that act on the aspect (*dobj*), and adverbs or adjectives used with verbs in passive

phrases (*nsubjpass*, *advmod*).

3. Score the opinion words: The detected opinion words are evaluated using VADER's sentiment lexicon, which assigns a compound polarity score from -1 to +1.

4. Negation Handling: Here we detect any negating words such as *"not"* or *"never"* which will invert the polarity of the opinion word.

5. Aggregation: For each aspect, we save the sentiment label (positive, negative, neutral), confidence score and the text index(from and to) in which the word is located in the text.

This implementation is really simple, executes fast, and performs well for clearly expressed sentiments. However, it struggles with sarcasm, implicit opinions, and context dependent sentiment shifts.

## Transformer Based

The ML ABSA method uses a pre-trained transformer model(***yangheng/deberta-v3-base-absa-v1.1***), which is specifically fine tuned for ABSA tasks. This method relies on contextual embeddings rather than predefined lexicons, which allows it to capture subtle relationships between aspects and opinion words.

Process Steps:

1. Aspect Detection**:** spaCy's noun chunking identifies candidate aspect terms within the sentence.

2. Model Input Formatting**:** For each aspect, the sentence and aspect term are paired and tokenized as the model's input.

3. Contextual Sentiment Prediction**:** The pre trained model(DeBERTa) model evaluates the input pair, to understand how sentiment words relate to the aspect in context.

4. Output Interpretation**:** The model produces a logit vector, which is converted into probabilities and mapped to a label (positive, negative, or neutral) with a confidence score.

This implementation performs really well on most examples, as it understands context and word dependencies beyond basic lexical rules/patterns. But, it still struggles with sarcasm and subtle pragmatic language not seen during pre training.

### LLM Based ABSA

The LLMABSA approach uses Large Language Models running locally via ***Ollama***, specifically **Phi-3** and **Mistral 7B**. Instead of relying on pre-trained sentiment classifiers, this method uses prompt engineering to instruct the model to perform aspect extraction and sentiment classification directly from text.

Process Steps:

1. Prompt Design**:** A structured prompt explaining what it's supposed to do, provide it with a few examples of the expected input & output, and request it to output in a JSON format, containing all info we need(the aspect, sentiment, confidence & text position).
2. Generation**:** The input text is passed to the LLM using the ollama client. The model generates the aspect-sentiment pairs.

3. Post Processing: The model's expected json response is cleaned and validated, then we append the results.

4. Error Handling: In case of a failed generation, or malformed JSON response, this implementation will retry a certain amount of times(3 times by default) automatically, else it will return an empty array.

This approach demonstrates the reasoning and generalization abilities of LLMs. Although slower, it can interpret sarcasm, implicit sentiment, and complex phrasing better than the transformer model.

# Design Decisions

## Lexicon ABSA

In this implementation, I'd like to explain my design choice regarding the dependency pattern selection. The model uses a small set of syntactic rules (*amod, acomp, dobj, nsubjpass*, and *advmod*) to identify relationships between aspects and their connected opinion words. These rules were selected because they cover the most frequent and semantically meaningful cases in English sentences, for example:

- *amod*: "amazing phone"

- *acomp*: "the food was great"

- *dobj*: "I hated the service"

- *nsubjpass* + *advmod*: "was beautifully designed"

Although this rule set does not account for every possible grammatical variation, it serves a good balance between accuracy and simplicity, so I believe this implementation is good enough. In other words, the goal was to design a model that performs well on common review style sentences without over engineering rare cases.

## Transformer ABSA

In this implementation, I'd like to explain why I  used noun chunk, as opposed to the token level noun detection that was used in the Lexicon ABSA model.

In the LexiconABSA implementation, aspects were identified at token level because the system relied on syntactic dependency relations between individual words. This word by word method makes it easier to directly link opinion words to the specific aspects they describe(for example, connecting "good" to "pizza" in the sentence "The pizza was good.")

However, In the Transformer ABSA model, sentiment prediction is handled by a pre trained contextual deep learning model that already captures linguistic dependencies. Therefore, it is more effective to pass noun chunks as inputs as opposed to single tokens. Also, using noun chunks allows the transformer to consider complete semantic units like "battery life" or "customer service" as single aspects.

### LLM ABSA

For this implementation, I used two different local LLMs(Phi-3 and Mistral 7B) to test how models of different sizes impact performance and interpretation. I chose Phi-3 because it's an extremely lightweight model (~2.2B parameters) and Mistral 7B, which represents a heavier, more advanced model.

Regarding prompts, since large language models can sometimes produce outputs with extra explanations, markdown formatting, or slightly malformed JSON, I implemented a small post processing helper function in utils.py called load_clean_json to make the parsing process more reliable.

This function cleans and validates the model's response before it is converted into an AspectSentiment object. It removes unwanted characters such as backticks, extracts only the valid JSON array from the text, and corrects small syntax issues like trailing commas.

This design choice makes sure that even if the LLM response isn't perfectly formatted, the implementation can still process and interpret it correctly.

## Experimental Setup

To evaluate and compare the 3 ABSA implementations, a small custom dataset was created using the **data_creation.py** script. This script combines random samples from 2 different datasets(Laptop_Train_v2 and Restaurants_Train_v2), both from the SemEval 2014, along with a set of 9 custom examples generated by ChatGPT, in order to ensure that the dataset covers all 3 edge cases:

- Simple cases(single aspect sentences with clear polarity): "*The pizza was delicious.*".
- Complex cases(multiple aspects or mixed sentiments): "*The phone screen is bright, but the battery drains quickly.*".
- Edge cases(sarcasm, implicit sentiment, ambiguous expressions): "*The movie was a masterpiece — I almost fell asleep.*".

### Evaluation Procedure

The evaluation process was done through the comparison.ipynb notebook. Each ABSA implementation(LexiconABSA, ML_ABSA, and LLMABSA (using both Phi-3 and Mistral 7B models)) were tested on the dataset that was created by the data_creation script.

For each sentence, the model's predicted sentiment for each aspect was compared against the ground truth label using these metrics:

- Accuracy, which represents the proportion of correctly predicted aspect–sentiment pairs.
- F1-score (macro): measures the balance between precision and recall across all sentiment classes, which ensures that each class (positive, negative, neutral) contributes equally to the overall score.
- Runtime:  total time taken to process the entire dataset(measured in seconds).

## Results & Analysis

### Qualitative Comparison

The qualitative evaluation was performed using exploration.ipynb, which compared the three ABSA implementations on several representative sentences, including simple, complex, and sarcastic cases.

- Lexicon ABSA:
  The lexicon based system correctly identified explicit opinions such as "The battery life is terrible" but failed to interpret sarcasm. In "The movie was a masterpiece — I almost fell asleep," it returned no negative sentiment since the sarcasm is implicit and not lexically expressed. On top of that, because this implementation identifies aspects at the token level, it sometimes misses full multi-word aspect phrases. In "The camera quality is amazing but the battery life is awful," it extracted "quality" (positive) and "life" (negative) instead of the complete phrases "camera quality" and "battery life."
- ML ABSA (Transformer):
  The transformer based model was more consistent and context aware compared to the lexicon based implementation. In "The camera quality is amazing but the battery life is awful," it handled multiple aspects within the same sentence effectively, and assigned the correct sentiment to each. However, as a supervised model trained on literal sentiment expressions, it still struggled with sarcasm and indirect tone. In "The movie was a masterpiece — I almost fell asleep," it misclassified the overall sentiment as positive because it focused on the positive word "masterpiece" without recognizing the sarcasm. And while noun chunking helped it capture multi-word aspects like "camera quality" and "battery life," the model can still miss subtle or implied opinions depending on the sentence structure.
- LLM ABSA (Phi-3 and Mistral 7B):

The LLM based models showed the strongest contextual understanding and flexibility compared to the other 2 implementations. They were better at interpreting sarcasm and complex phrasing, though they did behave differently, depending on the model used. In the sarcastic example “The movie was a masterpiece — I almost fell asleep,” Phi-3 demonstrated some understanding of irony by recognizing the overall negative sentiment toward the movie, though it misclassified “falling asleep” as positive(when taken literally).

Mistral 7B, on the other hand, produced the reverse interpretation, treating the movie as positive and the act of sleeping as negative, which shows that both models grasped parts of the sentiment but differed in contextual emphasis. In the straightforward sentences like “The camera quality is amazing but the battery life is awful,” both LLMs performed accurately, identifying “camera quality” as positive and “battery life” as negative with high confidence. Similarly, in “The restaurant has a modern interior and the food is fine,” both models extracted the correct aspects and assigned reasonable sentiments, though Phi-3 rated the sentiment around “food” slightly more neutral than Mistral.

The qualitative results reveal how each implementation gives us a different balance between processing speed, linguistic depth, and ability to adapt to complex language. Lexicon ABSA operates the fastest but provides only a basic, surface level understanding of sentiment. ML ABSA achieves a strong middle ground, which combines reasonable speed with a better understanding of context. On the other hand, LLM ABSA delivers the most comprehensive and context sensitive analysis, effectively handling subtle or complex expressions, though this comes at the cost of much longer runtimes, and higher computational demands.

## Quantitative Comparison & Performance Metrics

Each model was tested using the same data, measuring 3 metrics: Accuracy, Macro F1-score, and Runtime (seconds). Memory consumption was also monitored to provide an estimate of computational efficiency(the reported memory usage values were calculated based on the difference between process memory before and after model execution, regarding the negative values (as seen for the LLM models): (Quoting ChatGPT’s response, as I don’t have the expertise to definitively explain why the memory value is negative): “***Indicates that the Python process released cached memory or temporary data during or after inference. This does not represent actual negative usage but reflects fluctuations in the process’s memory footprint during execution.***”).

| ABSA Model | Accuracy | F1 | Time (seconds) | Memory (MB) |
|---|---|---|---|---|
| Lexicon | 0.167 | 0.125 | 0.22 | 1 |
| Transformer | 0.666 | 0.437 | 5.09 | 328 |
| Phi-3 | 0.633 | 0.324 | 72.79 | -6.4 |
| Mistral(7B) | 0.7 | 0.452 | 202.54 | -105.4 |

The Lexicon ABSA model performed the fastest and used minimal memory but achieved the lowest accuracy (at 16.7%) and F1-score (0.125), which highlights its limited ability to interpret sentiment beyond basic lexical patterns. The ML ABSA transformer model showed a huge improvement, achieving 66.7% accuracy and a balanced F1 of 0.437, performing reliably on explicit sentiment expressions. Regarding the LLM based models, Mistral 7B achieved the highest accuracy (70%) and F1 (0.453), surpassing all other implementations in interpretive quality. However, it took substantially more time to execute(~200s). As for Phi-3, while faster (73s) and more lightweight, it did slightly underperform in both accuracy and consistency (F1 0.324).

Regarding the memory usage difference between the Transformer ABSA and the LLM ABSA, in the Transformer implementation, the Hugging Face model is loaded inside the python process, as opposed to the LLM implementation, where it runs via Ollama, which is a separate background process.

# Discussion

Each ABSA method has its own strengths and weaknesses.

The Lexicon ABSA model is simple, executes fast, and is easy to understand, which makes it good for quick analysis or smaller applications. However, because it only follows basic word rules, it struggles with context, sarcasm, and hidden opinions. The ML ABSA transformer model provides a nice balance between accuracy and speed. It can understand the relationship between words more accurately than the lexicon model and works well for most normal sentences. But, just like the Lexicon implementation, it also has trouble with sarcasm and indirect meanings because it interprets text literally. The LLM ABSA models performed the best when it came to understanding complex sentences and subtle opinions. They can detect tone and meaning more deeply, but they also take much longer to run and need more computing power..

## Use case recommendations

The Lexicon ABSA model is best suited for quick & low resource analysis on simple reviews or short pieces of text where speed matters more than depth. The ML ABSA transformer model is the best choice for general purpose sentiment analysis, as it offers a good balance between accuracy and efficiency. As for the LLM ABSA models, they are ideal when deeper understanding is required, such as detecting sarcasm, irony, or mixed opinions, as long as execution time is not of a concern.

## Challenges Faced

During the development of this project, 3 main challenges were encountered.

The first issue was a runtime error related to *PyTorch* DLLs when trying to run the Transformer and LLM models in PyCharm(also known as PyTorch DLL Hell). The error message:

*"OSError: [WinError 1114] A dynamic link library (DLL) initialization routine failed. Error loading 'c10.dll' or one of its dependencies."*

This obstacle prevented me from working on the other 2 implementations for a while. Eventually I discovered that the issue was caused by missing Windows dependencies required by PyTorch. The problem was resolved by installing the latest Microsoft Visual C++ Redistributable.

The second challenge is related to the prompt reliability in the LLM ABSA implementation. While the models generally followed the prompt format, they occasionally went off road and produced a messy or overly descriptive responses, for example, combining multiple aspects into one ("*battery health/life*") or generating unnecessary explanations such as *"neutral to positive due to sarcasm detection as implied negative tone"* instead of clean sentiment labels. This was addressed by improving the prompt instructions and adding a lightweight post processing step (load_clean_json located in utils.py) to ensure the json could still be parsed and evaluated correctly. Unfortunately this problem is still somewhat present, even in the comparison notebook(check the output of the Phi-3 model).

Lastly, one of the most significant challenges I've faced during this project was time management. Balancing the workload of 3 different courses(Data 5, Data 6, and Programming 6) made it difficult to dedicate consistent focus to this project. The overlapping deadlines and shifting priorities resulted in a reduced amount of focused time available for a more detailed implementation and evaluation of the models. And while external workload was a major factor, I do acknowledge that better personal time management and more structured task planning could have helped reduce the impact of these difficulties.

## Conclusion

Through this project, I gained a much better understanding of how rule based, transformer based, and LLM based approaches handle aspect based sentiment analysis. Each method has its own strengths: the lexicon model excels in simplicity and speed, the transformer model offers strong contextual understanding, and the LLMs demonstrate advanced reasoning abilities.

Regarding potential improvements, I would expand the dataset to provide better coverage of all cases, especially edge cases. As for the models, in the Lexicon ABSA implementation, I would improve it by adding more syntactic rules and handling additional dependency cases to capture a wider range of aspect–opinion relationships. For the LLM-based approach, I'd focus on refining the prompt and performing more extensive tests, to ensure the LLM doesn't produce unnecessary explanations or overly verbose outputs.

This concludes the Data 6 ABSA Project.

Thank you for reading this, and looking forward to the oral exam!

## References

*yangheng/deberta-v3-base-absa-v1.1 · Hugging Face*. (n.d.).

https://huggingface.co/yangheng/deberta-v3-base-absa-v1.1

*SPACY · Industrial-strength Natural language processing in Python*. (n.d.). https://spacy.io/

Cjhutto. (n.d.). *GitHub - cjhutto/vaderSentiment: VADER Sentiment Analysis. VADER (Valence Aware Dictionary and Sentiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media, and works well on texts from other domains*. https://github.com/cjhutto/vaderSentiment

Pontiki, M., Galanis, D., Pavlopoulos, J., Papageorgiou, H., Androutsopoulos, I., & Manandhar, S. (2014). SEMEVAL-2014 Task 4: Aspect based Sentiment analysis. *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*. https://doi.org/10.3115/v1/s14-2004

*Ollama*. (n.d.). Ollama. https://ollama.com/