2024/25

ABSA COP Based

CORSO DI SOCIAL NETWORK ANALYSIS

EDOARDO MISURELLI - MATRICOLA 224-438

Summary

Abstract

ABSA (Aspect-Based Sentiment Analysis) is a task used to identify specific aspects within a text and assign them a polarity (e.g., negative, neutral, or positive). My project focuses on this task within a particular context: COP, the annual conference on climate change.

Every year, people around the world express support or discontent regarding COP through social media platforms like X (formerly Twitter). The main goal of this project is to collect and analyse these data to fine-tune a pre-trained model for ABSA, specifically within the Climate Change and COP domain.

The project consists of two main steps:

- **Step 1:** Use Mistral to generate samples, which will serve as a training dataset for fine-tuning.
- **Step 2:** Fine-tune a pre-trained model on the ABSA task within the COP domain.

Finally, the model was evaluated using a test set, also generated by Mistral.

Data Cleaning and Filtering

Data cleaning is a crucial phase in any analysis process, especially when working with social media data, where content is often noisy and filled with symbols, non-standard characters, and other irrelevant information. This chapter focuses on the process of cleaning and filtering tweets, with particular attention to removing unwanted elements such as URLs, mentions, and duplicates. Additionally, it explains how the code filters data based on text length to ensure a more consistent and usable dataset.

Main Steps of Data Cleaning

- **Removal of Apostrophes**
    - Apostrophes (') are removed to standardize the text and reduce unnecessary variations.
- **Removal of URLs**
    - URLs, identified by the pattern http\S+, are removed. This step is essential to prevent web links, which often lack semantic value, from influencing the analysis results.
- **Removal of Non-ASCII Characters**
    - Any character not belonging to the ASCII standard is removed, reducing issues related to encoding.
- **Removal of Symbols**
    - Symbols like parentheses (), exclamation marks !, and question marks ? are removed and replaced with spaces.
- **Removal of Text Inside Square Brackets**
    - Content enclosed in square brackets is eliminated to avoid including irrelevant information.
- **Removal of Mentions and Hashtags (Optional)**
    - If the remove_mentions parameter is set to True, mentions (@username) are removed.
    - If the remove_hashtags parameter is set to True, hashtags (#word) are also removed.

Dataset Filtering

After cleaning the individual tweets, the dataset is further processed to ensure data quality and consistency. This is done through two main operations:

- **Removal of Duplicates**
    - Duplicate tweets in the "Clean" column are removed, ensuring that each cleaned tweet is unique.
- **Length Filtering**
    - Tweets exceeding a length of 256 characters are discarded. This step helps avoid excessively long content, which could be out of context or considered spam.
    - Give a context of max 256 chars to the Tokenizer.

Using Mistral to generate ABSA sample

The next step was to use Mistral for generating sample data derived from tweets. To achieve this, I utilized **Mistral Instruct v3**, a model capable of providing responses based on the given prompt.

I focused on defining a **valid prompt** to submit to Mistral in order to obtain samples in the following **JSON format**:

- **Text**
  A field containing the tweet text.
- **Labels**
  A field that includes three subfields:
  - **Span**: Identifies the substring that contains a specific aspect found in the text.
  - **Category**: Specifies the category associated with the identified span.
  - **Polarity**: Indicates the sentiment associated with the span.

The **categories** used to tag spans were predefined and cover most of the topics discussed in COP-related tweets. The selected categories were:

- **GREEN ENERGY**
- **SUSTAINABILITY**
- **ENVIRONMENTAL ISSUE**
- **CONFERENCE**
- **STAKEHOLDER**
- **ORGANIZATION**
- **POLICY AND STRATEGIES**

As for **polarity**, or the sentiment associated with each span, I decided to use three classes:

- **POSITIVE**
- **NEUTRAL**
- **NEGATIVE**

A particularly challenging phase was the identification of **overlapping spans**, which are text sequences belonging to different spans with different categories but overlapping each other.

To handle this, I employed various **prompting techniques** to better capture this phenomenon:

- **Zero-shot**
- **Few-shot**
- **Chain of Thought**
- **Zero-Shot Chain of Thought**

Each of these approaches influenced the model's ability to generate samples with varying degrees of effectiveness. The most successful techniques were **Few-shot** and **Chain of Thought**, as they enabled the model to identify more spans and, crucially, to detect many overlapping spans.

The final choice fell on **Few-shot prompting**, as it provided a **better distribution of categories** in the dataset.

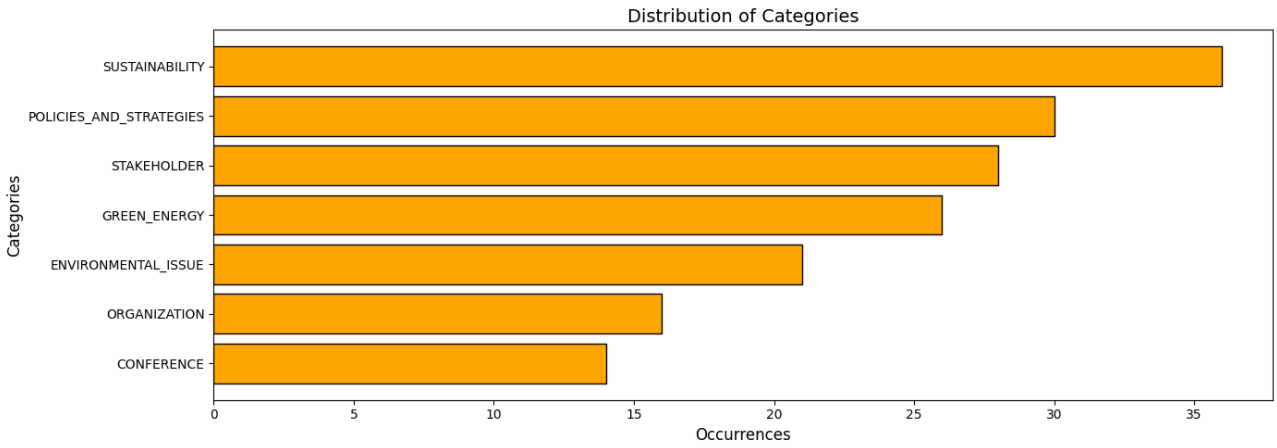


Figure 1: Few-shot distribution of span classes

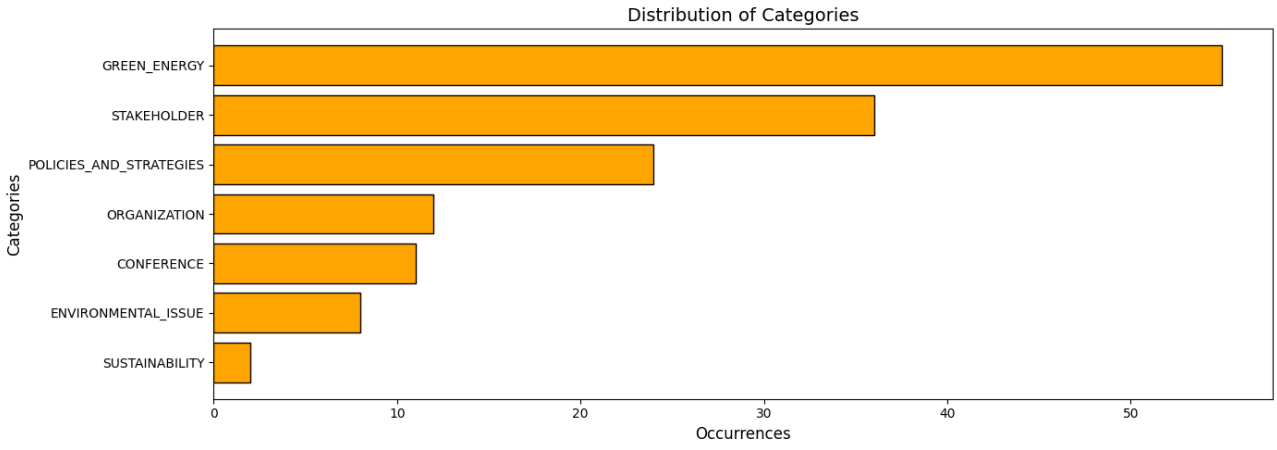


Figure 2: Chain of Thought distribution of span classes

Data embedding and Fine tuning

Add Offset

Before performing the **fine-tuning** of the selected model, I carried out a **data preparation phase** to ensure that the data could be correctly used as input to the model.

The first step was to **add offsets** to the identified spans as a **post-processing** step, meaning after generation. This process introduced two new attributes to the sample structure:

- **Start**
  Indicates the starting position of the span within the text.
- **End**
  Indicates the ending position of the span within the text.

These two new tags were essential for the following steps.

Data Embedding

Once the modifications were applied, the next step was to **split the dataset** into **train, test, and validation sets**. Each set was then converted into a **one-hot representation**.

The one-hot representation was carried out following these steps:

1. **Tokenizing** the text using a tokenizer.
2. For each identified span, the **offset** was used to correctly map tokens to their corresponding spans.
3. Each token was **assigned one or more labels** using a **one-hot encoding** representation.
4. To tag each token, I applied the **IOB (Inside-Outside-Beginning) tagging scheme**, effectively **doubling** the number of class labels. For example, the **GREEN ENERGY** category resulted in <I_GREEN_ENERGY, B_GREEN_ENERGY>. This tagging strategy was particularly useful for helping the model recognize **overlapping spans**.
5. At this point, each token was represented by a **one-hot vector**, where the **i-th position** contained a **1** if the token belonged to a certain class. Importantly, multiple elements of the vector could be active due to the presence of overlapping spans.

Aspect-Based Sentiment Analysis (ABSA) requires that each entity identified in the text be classified both **by category** and **by sentiment**.

Thus, we define two tasks:

- **Task 1: Span Categorization**
- **Task 2: Sentiment Analysis**

The model needs to process **two types of embeddings**:

1. **Category labels embedding**
2. **Sentiment labels embedding**

Both are transformed into **one-hot vectors** associated with the tokens for simplicity. However, the **IOB tagging scheme** was used only for the **category classification task**.

Example of a **one-hot vector** used for **span categorization**:

- [[0,0,1,0,0…….,1,0],[0,1,0,1,0,……..],….]

The **maximum sequence length** defined by the tokenizer was **256**, meaning the model can handle examples up to **256 characters long**.

Token	Labels
<s>	[]
Day	['B-CONFERENCE']
1	['I-CONFERENCE']
Ke	['I-CONFERENCE']
yn	['I-CONFERENCE']
ote	['I-CONFERENCE']
Panel	['I-CONFERENCE']
:	[]
Se	['B-POLICIES_AND_STRATEGIES']
ism	['I-POLICIES_AND_STRATEGIES']
ic	['I-POLICIES_AND_STRATEGIES']
Shift	['I-POLICIES_AND_STRATEGIES']
in	['I-POLICIES_AND_STRATEGIES']
Energy	['I-POLICIES_AND_STRATEGIES']
Canadian	['B-SUSTAINABILITY', 'B-ENVIRONMENTAL_ISSUE']
Oil	['I-SUSTAINABILITY', 'I-ENVIRONMENTAL_ISSUE']
Sands	['I-SUSTAINABILITY', 'I-ENVIRONMENTAL_ISSUE']
in	['I-SUSTAINABILITY']
Transition	['I-SUSTAINABILITY']

Token	Labels
<s>	['neutral']
Day	['neutral']
1	['neutral']
Ke	['neutral']
yn	['neutral']
ote	['neutral']
Panel	['neutral']
:	['neutral']
Se	['positive']
ism	['positive']
ic	['positive']
Shift	['positive']
in	['positive']
Energy	['positive']
Canadian	['positive']
Oil	['neutral']
Sands	['neutral']
in	['neutral']
Transition	['neutral']

Figure 3: Example of overlapping spans, token belong to different classes

Define model and metrics

The final step before fine-tuning was to define the model to be used.

The selected model was **RoBERTa-base**, but it was **redefined for a multitask classification** task.

The essential modifications were:

- **Addition of two classification heads**
    - To handle both classification tasks separately.
- **Modification of the loss function**
    - Implementation of **Focal Loss** using **Binary Cross Entropy**, given the binary nature of the vectors.

The two classification heads output the **logits** for the two tasks, making the model **multitask**, while the use of **Focal Loss** was crucial in handling **class imbalance**.

This adjustment ensured that **underrepresented examples** in the dataset carried more weight in the model's **loss computation**. It is important to note that, in the **sentiment analysis task**, the class imbalance was already present in the data itself, meaning it was not introduced by the **Mistral-based sample generation** process.

Given the **difference in difficulty** between the two tasks (**the first task being more complex due to the presence of multiple labels**), I introduced a **balancing parameter (alpha)**, assigning **greater weight to the first task** in the loss calculation.

To assess the model's performance after fine-tuning, I selected the following **evaluation metrics**:

- **Precision**
- **Recall**
- **F1-score**

These metrics were computed using the **macro-averaging** approach, meaning they were calculated **class by class** to accurately measure the model's performance.

Finally, the **aggregated scores** were computed to obtain an **overall evaluation** of the two tasks.

Fine tuning

For the **fine-tuning phase**, I relied on the **Hugging Face transformers library**, using the **Trainer module** to handle the entire process from **training to testing**.

The final **training dataset** consisted of approximately **4,000 examples**, while the **validation** and **test sets** contained around **500 examples** each.

Below are the **training parameters** used:

```
training_args = TrainingArguments(
    output_dir= MODEL_PATH,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    save_total_limit= 2,
    learning_rate=3.5e-5,
    per_device_train_batch_size = 16,
    per_device_eval_batch_size = 16,
    num_train_epochs= 10,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
    load_best_model_at_end=True,
    metric_for_best_model="overall_macro_f1",
    seed = 12345,
    resume_from_checkpoint = True
)
```

Test and Inference

In the final phase of the project, I **tested the model after fine-tuning**.

The results were **modest**, confirming that the **first task (span categorization)** was more challenging for the model, as expected. This was likely due to **class imbalance** and the **limited number of examples**, which were not sufficient for the model to achieve high performance.

Regarding the **F1-score**, the graph shows that the **overall F1-score** is approximately **0.8**.

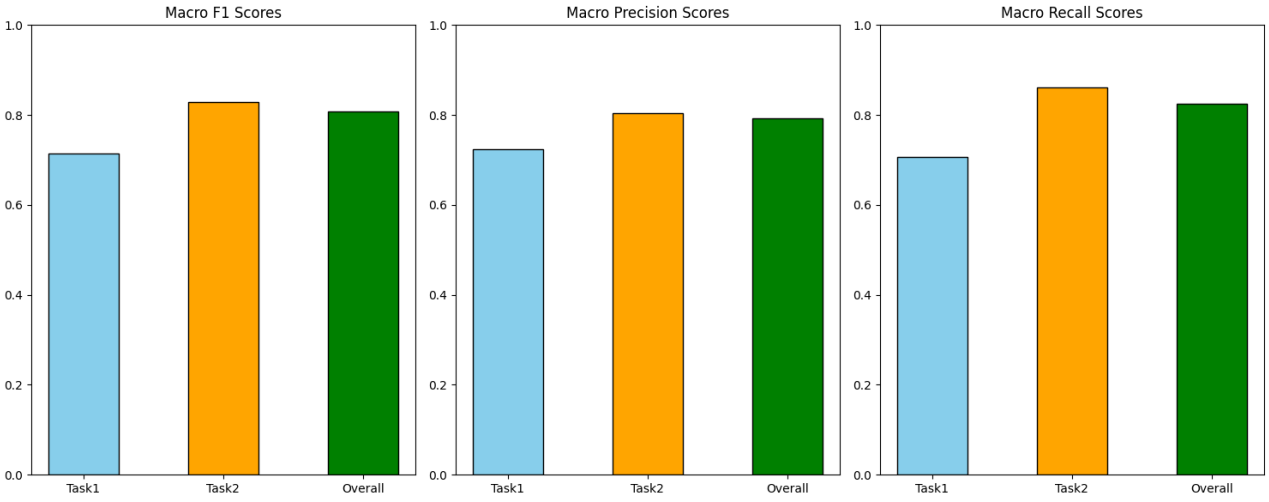


Figure 4: Macro scores

Now, let's take a closer look at the **second graph**, which illustrates the **model's performance for each class** in **Task 1 (Span Categorization)**.

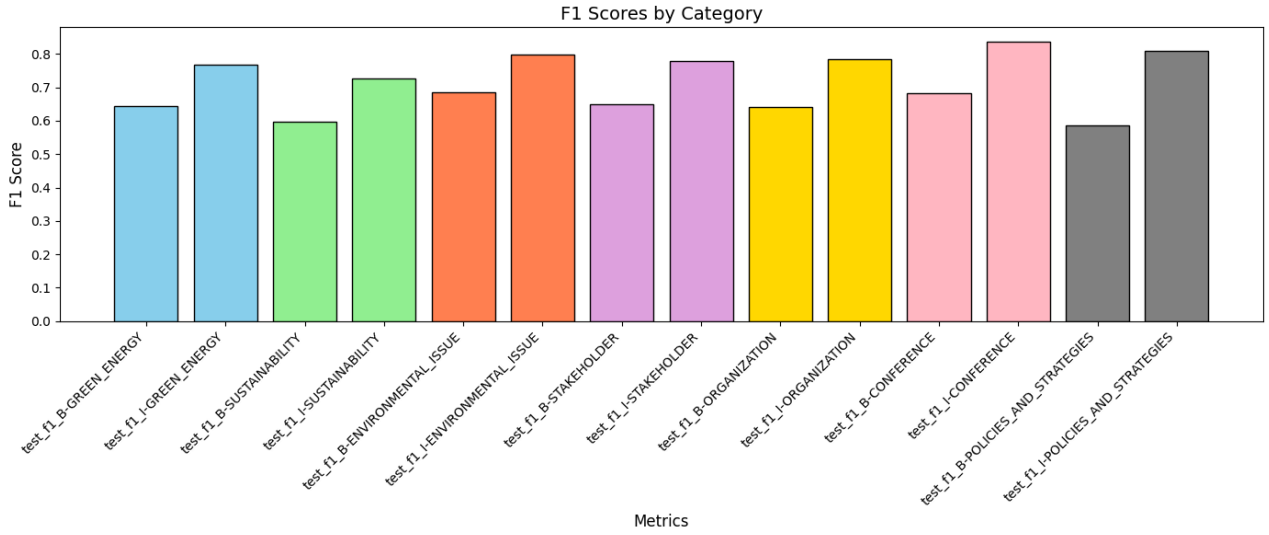


Figure 5: F1 scores for every class of task 1

We can clearly see that the model **struggled more with the B classes**, which were **less represented** compared to the **I classes**.

How about the aggregated metrics for the categories labels?

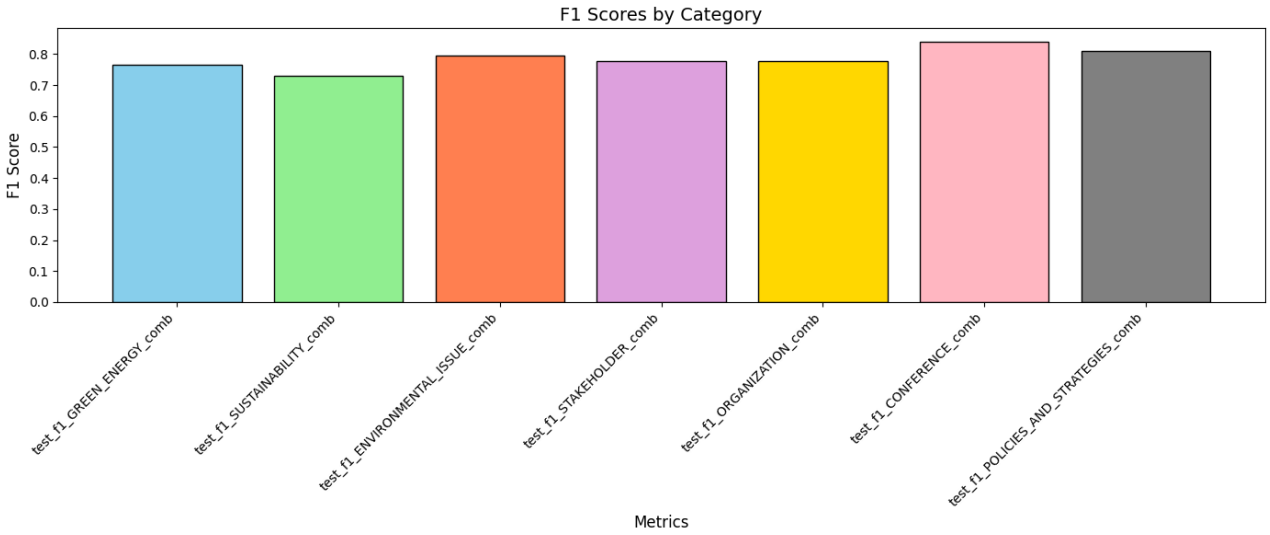


Figure 6: F1 scores for each category

Look how in this case the F1 scores is much higher, because we aggregate the B and I tag scores for each category; In this way, even a token that belongs to class B and is classified as I, as long as the token corresponds to the same class, will result in a correct prediction

How does the model perform in identifying **spans** and their associated **sentiment**?

To evaluate this, I provided the following **input text** to the model:

"During COP27, world leaders emphasized the urgent need for increased climate financing to support vulnerable nations in adapting to the impacts of global warming."

Here the result:


```
[{'start': 7,
  'end': 12,
  'polarity': 'neutral',
  'category': 'CONFERENCE',
  'span': 'COP27'},
 {'start': 14,
  'end': 27,
  'polarity': 'neutral',
  'category': 'STAKEHOLDER',
  'span': 'world leaders'},
 {'start': 43,
  'end': 86,
  'polarity': 'positive',
  'category': 'POLICIES_AND_STRATEGIES',
  'span': 'urgent need for increased climate financing'},
 {'start': 136,
  'end': 161,
  'polarity': 'negative',
  'category': 'ENVIRONMENTAL_ISSUE',
  'span': 'impacts of global warming'}]
```

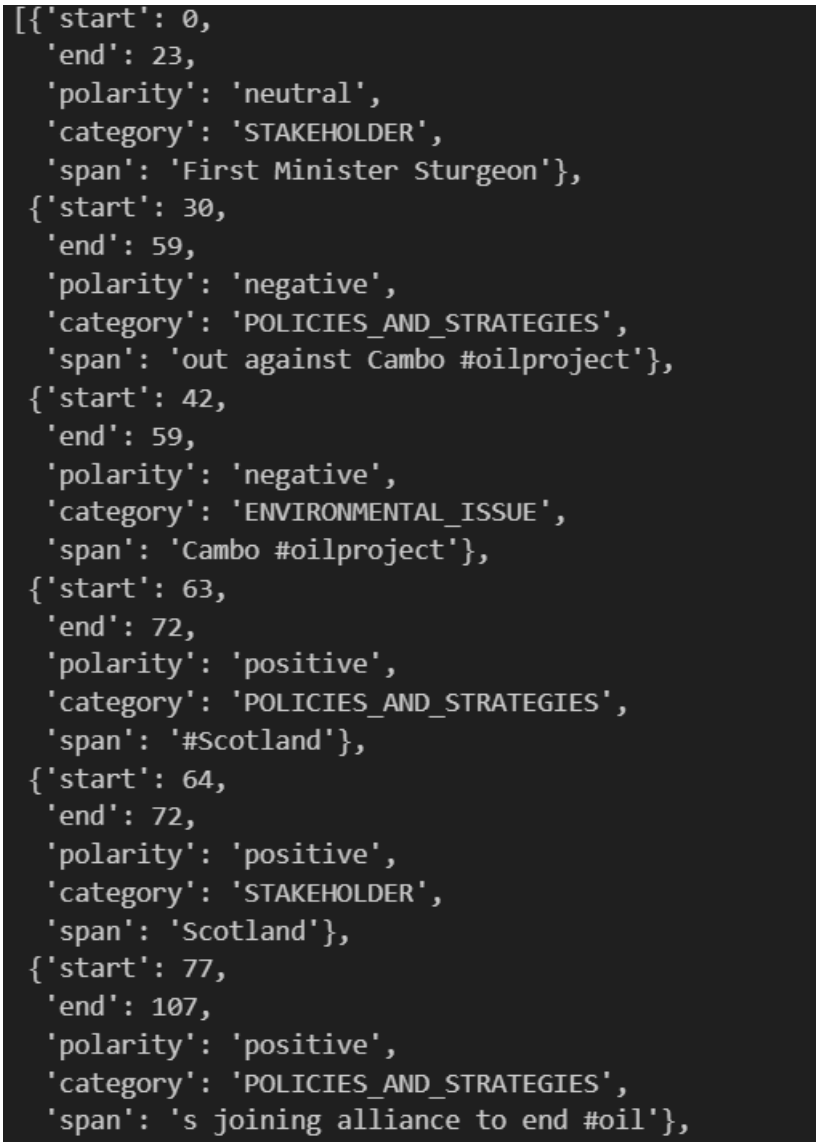

The model **successfully identified** the spans and their corresponding **categories** correctly.

In the previous example, there was **no presence of overlapping spans**.

Now, testing with another example:

"First Minister Sturgeon comes out against Cambo #oilproject as #Scotland mulls joining alliance to end #oil and #gas use -#OffshoreEnergy #netzero #energytransition #fossilfuels #COP26 #GlasgowClimatePact"

The model's response was:



It is noted how a particular span, "**out against Cambo #oilproject**" falls into two different categories, namely **POLICIES_AND_STRATEGIES** and **ENVIRONMENTAL ISSUE**, this shows how the model was able to identify overlapping spans."