Identifying Rhetorical Devices in Literature Using AI

—

By Anirudh B., Vivian T., Cindy L., Kritik J.

Focus

For our project, we focused on placing “rhetoric” tags in the “Memorias” file (list shown on the right). This memoir from the 14th century* was written in Castilian Spanish**.

* due to the age of the text, we were unsure of the extent to which the LLMs could identify all rhetorical devices.
** we did not translate it while preprocessing.

Language/ Poetics:
Rhetoric “rhet”

1. Captatio Benevolentiae (captatio)
2. Colloquialism
3. Pathos (emotion)
4. Logos (reason)
5. Ethos (character)
6. Allegory
7. Ekphrasis (art)
8. Metaphor
9. Sign / symbol
10. Exegesis (Hermeneutics/ Gloss)
11. Parallelism
12. Didactics
13. Invective
14. Amplification / Polish
15. Anaphora
16. Antithesis
17. Apostrofe (person) / Invocation (God)
18. Exclamation / Ecphonesis
19. Polyptoton (repetition of the root of a word)
20. Hypophora (rhetorical question)
21. Orality / Literacy

Method 1

BERT

BERT is a transformer-based model used for NLP (Natural Language Processing).

Process

1) Parse the XML files using Python, filtering away non-rhetorical tags.
2) Extract sentences and token-level labels, where each token is tagged according to whether it corresponds with a rhetorical label.
3) Separate sentences and tokenize using SpaCy
4) Align tokens with their respective labels from the XML annotations.

Trial and Error

BERT was originally developed and trained on English language datasets, making it unsuitable for recognizing rhetorical features in **Castilian Spanish** literature.

Instead, we tried using:

- **BETO**, a BERT model **trained on a large Spanish corpus**

- **RoBERTa**, an advanced version of the BERT model that was **trained on significantly larger, multilingual datasets**.

Conclusions

Despite these workarounds, implementing the models still proved difficult:

- the **original annotations were sparse**

- they **did not include every rhetorical device**

With **limited data to train on**, using the BETO and RoBERTa models **did not seem like an optimal way to produce accurate annotations**.

Method 2

Meta’s Llama 7B

Process

We took two pathways:

1) Zero-Shot Prompting
 Load LLaMA's pretrained weights, pull your TEI taxonomy from the XML, and craft a simple prompt for each sentence asking "Which tag fits this text?", then wrap its reply back into the segment tags without any model training.

2) Supervised Fine-Tuning
 Convert each segment span in your XML into a "sentence→label" example, add a lightweight classification head to LLaMA, and train briefly so it learns your exact TEI labels—then run new sentences through this tuned model for more consistent annotations.

Trial and Error

- Zero-shot prompting could often guess common devices correctly, but struggled with rarer labels and Spanish phrasing.
- Local LLaMA load times on a laptop proved slow and memory-intensive; smaller checkpoints (e.g. BLOOMZ-560m) booted faster but could lose accuracy.
- Fine-tuning LLaMA-7B on just ~500 XML spans required careful hyperparameter tuning. Too many epochs led to over-fitting, too few would leave some devices undetected.
- Prompt variations may improve results in some cases, but it could lead to inconsistency across paragraphs.

Conclusions

Despite of relentless efforts, and having a clear plan of both the pathways to implement the Llama, we were not able to finish both of them because of Llama taking an extremely long time to load on my laptop. The major reason for this is probably storage issues, for example, a very overburdened cache, so I plan on closing most applications and then re running my program. We should be able to resolve the issue since it's not a major one, but weren’t able to do so in the given time.

Method 3

NLP using spaCy and SciKit Learn

Background

- Utilized **sk-learn CRF algorithm** for classifying phrases by rhetorical device.
- Takes partially annotated XML file and the original text pdf as inputs and outputs an annotated XML file

```
# Step 1: Extract text from PDF
pdf_reader = PdfReader("/content/drive/MyDrive/Midterm Project/
text = ""
for page in pdf_reader.pages:
    page_text = page.extract_text()
    if page_text:
        text += page_text.replace('\n', ' ')

# Optional cleaning: remove extra spaces
text = re.sub(r'\s+', ' ', text)

# Step 2: Parse annotations from XML
tree = ET.parse("/content/drive/MyDrive/Midterm Project/Midterm
root = tree.getroot()
annotations = []

for seg in root.iter('seg'):
    if 'type' in seg.attrib and seg.text:
        label = f"{seg.attrib['type']}:{seg.attrib.get('subtype
        ann_text = seg.text.strip()
        annotations.append({'label': label, 'text': ann_text})

# Step 3: Align spans to character positions
for ann in annotations:
    match = re.search(re.escape(ann['text']), text)
    if match:
        ann['start'] = match.start()
        ann['end'] = match.end()
    else:
        ann['start'] = None
        ann['end'] = None

# Step 4: Process tokens with spaCy
nlp = spacy.load('en_core_web_sm')  # Or 'es_core_news_sm' if S
```

Conclusions

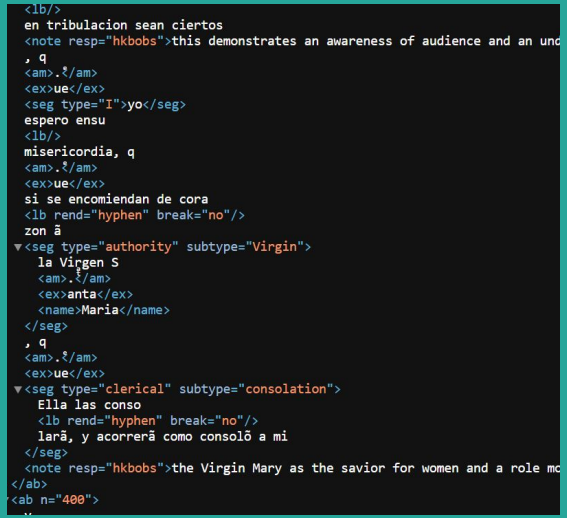
- Model has an inbuilt accuracy metric, receiving a 97% accuracy score in predicting new labels.
- Data ended up incorrectly skewing the results, struggled with texts other than Memorias.

Method 4

Google Gemini API

Background

- Uses a **similar pipeline to the NLP model**.
- Utilizes **Gemini API for the backend machine learning** instead of python libraries
- **Accuracy and responses improved** due to using an LLM like Gemini



Conclusions

- Intuitive inputting system for files and the output file retains context allowing for easier parsing.
- Constrained by amount of Gemini credits available for free.