# Plan

**1. Understand the Path Model of Blame**

The Path Model of Blame consists of stages that help determine whether a particular text express blame or praise. The key stages include:

* **Event Detection:** Identify whether a relevant event is present.
* **Agent Causality:** Determine if a moral agent is responsible for the event.
* **Foreseeability and Coercion:** Identify if the event was foreseeable by the agent or if coercion played a role.

**2. Prepare the Environment**

Ensure your environment is set up:

* Install necessary Python libraries: nltk, spacy, scikit-learn, StanfordNLP, etc.
* Load the necessary NLP models and tools like POS taggers, dependency parsers, and named entity recognition (NER) models.

**3. Text Preprocessing**

Start with typical text preprocessing steps:

* **Sentence Splitting and Tokenization:** Break the text into sentences and words.
* **Part-of-Speech (POS) Tagging:** Annotate each word with its POS tag.
* **Named Entity Recognition (NER):** Identify entities like people, organizations, etc.
* **Word Sense Disambiguation (WSD):** Resolve ambiguity for words with multiple meanings.

**4. Event Detection**

* **Verb-Object Pattern Identification:** Using dependency parsing, detect events by identifying verbs paired with their objects.
* **Polarity Detection:** Classify events as positive or negative based on sentiment analysis using lexicons (e.g., SentiWordNet).
* **Negation Handling:** Consider negations during sentiment analysis to correctly classify the event's polarity.

**5. Agent Causality Determination**

* **Identify Moral Agents:** Use NER to detect whether the subject is a person, organization, or country.
* **Causation Patterns:** Apply patterns like "NP verb NP" (noun-phrase, verb, noun-phrase) to identify causality.

**6. Foreseeability and Coercion**

* **Foreseeability:** Detect verbs that imply foreknowledge (e.g., communication verbs like "say", "tell").
* **Coercion:** Identify verbs or phrases indicating coercion (e.g., "forced", "pressured", "coerced").

**7. Classification Logic**

Based on the above features, classify the text as one of the following:

* **Blame:** The agent is held responsible for a negative event.
  + **Self-Blame:** The agent is the speaker themselves.
  + **Blame Others:** The agent is another person or entity.
* **Praise:** The agent is credited for a positive event.
  + **Self-Praise:** The agent is the speaker.
  + **Praise Others:** The agent is another person or entity.
* **Others:** No blame or praise is expressed.

**8. Corpus Annotation and Data Preparation**

* Use the ISEAR dataset or any similar corpus annotated for blame and praise.
* If you create a new dataset, follow annotation guidelines similar to the paper (label sentences as "blame", "praise", or "others").

**9. Rule Implementation**

Translate the above logic into Python code:

* Use conditionals and logical rules to classify text based on detected features.
* Implement checks for agent causality, foreseeability, and coercion.

**10. Evaluation**

* Compare your rule-based classifier against supervised models like SVM or Naive Bayes using standard metrics (precision, recall, F1-score).
* Perform 10-fold cross-validation to validate the performance.

**11. Error Handling and Refinement**

* Handle edge cases, such as sentences without clear verb-object pairs or ambiguous sentiment.
* Refine your rules based on evaluation results.

**12. Fine-Grained Classification**

* Extend your rules to distinguish between different types of blame and praise (self-blame, blame-others, etc.) based on agent pronouns and entity types.

**13. Testing and Optimization**

* Test on different types of text, including informal sources like social media.
* Optimize by incorporating additional linguistic features or expanding your verb and sentiment lexicons.

**14. Documentation and Reporting**

* Document your approach, including any assumptions or challenges encountered.
* Compare your results with those reported in the original paper, focusing on key insights.

By following this plan, you can replicate the blame/praise detection approach outlined in the paper while gaining a deeper understanding of rule-based NLP systems.

# Installed libraries

Python version: 3.9.19

1. **NLTK (Natural Language Toolkit)**
   * Used for tokenization, POS tagging, and other basic NLP tasks.
   * NLTK is used for tasks like word sense disambiguation (WSD) and integrating lexicons like WordNet and SentiWordNet.
   * Install: pip install nltk
   * Import:

import nltk

nltk.download('wordnet')

nltk.download('averaged\_perceptron\_tagger') nltk.download('sentiwordnet')

nltk.download('punkt')

nltk.download('opinion\_lexicon')

nltk.download('punkt\_tab')

nltk.download('averaged\_perceptron\_tagger\_eng')

 **nltk.download('wordnet')**

* **Why?**: WordNet is a lexical database that is crucial for tasks like word sense disambiguation (WSD) and exploring synonyms, hypernyms, and hyponyms. It’s a foundational resource in natural language processing, especially for understanding the semantics of words.

 **nltk.download('averaged\_perceptron\_tagger')**

* **Why?**: This resource provides a part-of-speech (POS) tagger, which is essential for annotating words with their grammatical roles (e.g., noun, verb, adjective). POS tagging is a key step in the pipeline for determining events, agent causality, and more.

 **nltk.download('sentiwordnet')**

* **Why?**: SentiWordNet is a sentiment lexicon derived from WordNet that provides sentiment polarity scores (positive, negative, objective) for words. It’s useful for the sentiment analysis component of the blame/praise detection task.

 **nltk.download('punkt')**

* **Why?**: The Punkt tokenizer is used for sentence splitting and word tokenization. It’s important for breaking down text into manageable units like sentences and words before applying other NLP tasks.

 **nltk.download('opinion\_lexicon')**

* **Why?**: The Opinion Lexicon contains lists of positive and negative opinion words, which are useful for sentiment analysis. It helps in classifying the sentiment polarity of text and complements SentiWordNet in determining whether the text expresses blame, praise, or neutrality.

1. **StanfordNLP**

**Stanza (formerly StanfordNLP)**

Stanza is the modern replacement for the Stanford NLP suite and provides tokenization, POS tagging, dependency parsing, and named entity recognition (NER).

Code:

pip install stanza

import stanza

stanza.download('en')

1. **scikit-learn**
   * Used for evaluating your classifier and implementing any supervised models for comparison.
   * Install: pip install scikit-learn
   * Import:

python

Copy code

from sklearn.model\_selection import cross\_val\_score

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.feature\_extraction.text import CountVectorizer

1. **WordNet (via NLTK)**

NLTK provides the necessary tools to perform WSD using WordNet.

1. **AFINN**
   * For sentiment analysis, a lexicon-based approach.
   * Install: pip install afinn
   * Import:

python

Copy code

from afinn import Afinn

1. **SentiWordNet (via NLTK)**
   * Another lexicon for sentiment analysis.
   * Install: It’s included in NLTK; you need to download the resource.

python

Copy code

nltk.download('sentiwordnet')

* + Import:

python

Copy code

from nltk.corpus import sentiwordnet as swn

1. **Pandas (for managing data)**
   * Install: pip install pandas
   * Import:

python

Copy code

import pandas as pd

1. **NumPy (for numerical operations)**
   * Install: pip install numpy
   * Import:

python

Copy code

import numpy as np

1. **Matplotlib or Seaborn (optional, for visualizing results)**
   * Install: pip install matplotlib seaborn
   * Import:

python

Copy code

import matplotlib.pyplot as plt

import seaborn as sns

sdfs

# What did I import

import nltk

import stanza

from afinn import Afinn

from sklearn.model\_selection import cross\_val\_score

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.feature\_extraction.text import CountVectorizer

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

 **import nltk**

* Handles various NLP tasks such as tokenization, word sense disambiguation (WSD), and working with lexicons like WordNet and SentiWordNet.

 **import stanza**

* Provides the core NLP tasks such as tokenization, POS tagging, dependency parsing, and named entity recognition (NER), all in one package.

 **from afinn import Afinn**

* Used for sentiment analysis by leveraging the AFINN lexicon, which assigns sentiment scores to words.

 **from sklearn.model\_selection import cross\_val\_score**

* Useful for performing cross-validation, especially if you’re evaluating your classifier’s performance.

 **from sklearn.metrics import classification\_report, confusion\_matrix**

* These functions provide evaluation metrics such as precision, recall, F1-score, and confusion matrices to assess the performance of your model.

 **from sklearn.feature\_extraction.text import CountVectorizer**

* Allows you to convert text data into a bag-of-words model, which is useful for vectorizing text if you experiment with traditional machine learning models.

 **import pandas as pd**

* Essential for managing and manipulating your dataset, especially if you’re working with structured data like CSV files.

 **import numpy as np**

* Handles numerical operations, such as array manipulations, and complements pandas.

 **import matplotlib.pyplot as plt**

* Provides plotting functions to visualize your data and results (e.g., confusion matrices, performance graphs).

 **import seaborn as sns**

* Enhances your data visualizations, especially for heatmaps (e.g., confusion matrices), distribution plots, and more.

# Notes

They use such bibs, which would be good to replace with spacy actually. But okay we replicate.