Character Emotional Analysis Service with Python NLTK

20190253 Seungjun Park

20220049 Chaewoon Ki

1. Introduction

When reading a novel, detecting emotions of the characters and tracking the changes are crucial to understand a storyline of the novel. This project focuses on creating a Python-based program utilizing the Natural Language Toolkit (NLTK) to analyze and track the emotions experienced by characters in a novel.

Our Emotion Analyzer program detects characters' names and classifies their emotions into one of eight categories based on Plutchik's (1980) "Wheel of Emotions." The results are saved in a TSV file, facilitating detailed analysis of emotional shifts throughout the narrative.

This report will outline the development and performance of the program, detailing its components and the results of our experiments. Emotion Analyzer is a program designed to identify character names in a novel text and track the emotional changes of each character over time. Detailed code is open at https://github.com/Seungjun1127/NLPproject.

2. Approach

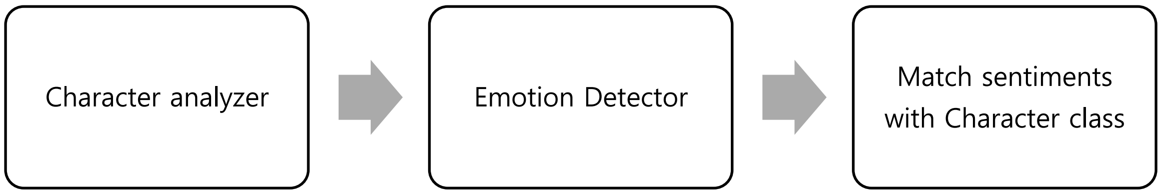

Figure 1: Main Structure of Emotion Analyzer

The development of the Emotion Analyzer is structured around three main components: Character Analyzer, Emotion Detector, and Sentiment Matching. Given a text, each sentence in the text undergoes those three stages. Each component plays a critical role in processing the text and providing accurate emotional insights.

The Character Analyzer identifies the names of characters in the sentence and creates a character object with the corresponding name. The Emotion Detector detects sentiments in each sentence and returns the emotions present in that sentence. Finally, the characters in the sentence and their emotions are stored together in a Character object.

2.1 Character Analyzer

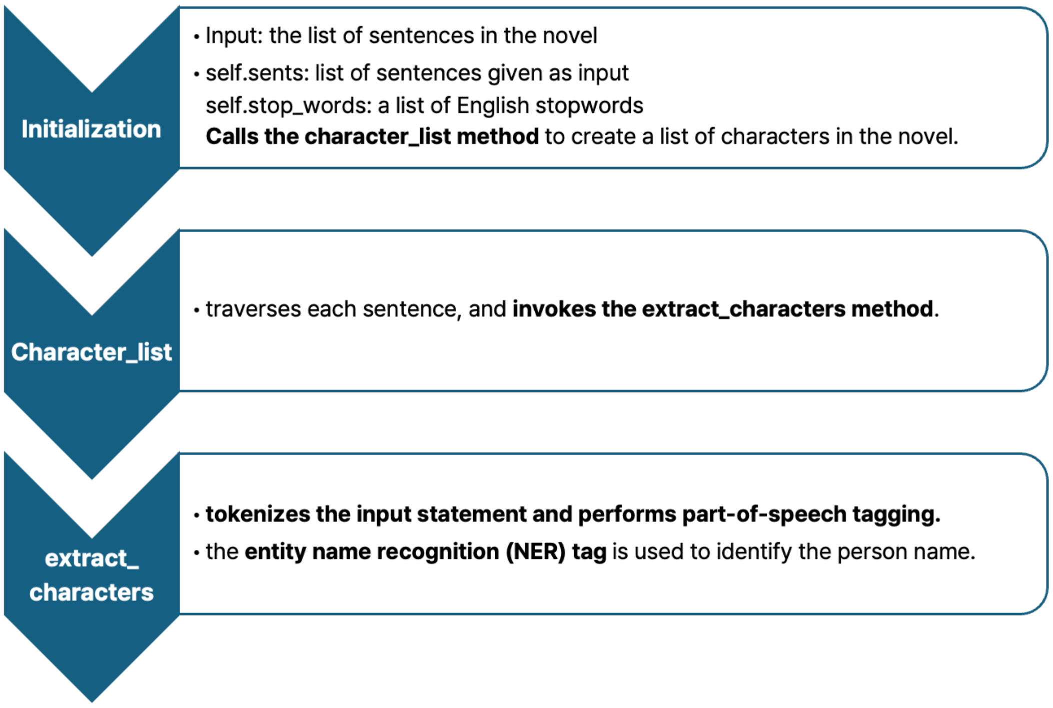
Initialization
• Input: the list of sentences in the novel
• self.sents: list of sentences given as input
self.stop_words: a list of English stopwords
Calls the character_list method to create a list of characters in the novel.
Character_list
• traverses each sentence, and invokes the extract_characters method.
extract_ characters
• tokenizes the input statement and performs part-of-speech tagging.
• the entity name recognition (NER) tag is used to identify the person name.

Figure 2: Main Structure of Character Analyzer

The Character Analyzer is designed to identify character names within sentences from the novel and prepare the data for subsequent emotional analysis. The process is as follows:

<u>Input Handling:</u> The Character Analyzer takes a list of sentences from the novel as input. These sentences are stored in the sents attribute of the Character Analyzer class.

<u>Character List Generation:</u> It calls the *character_list* method to generate a list of characters. This method iterates through each sentence and extracts potential character names using the *extract_characters* method. The Character Analyzer also maintains a list of English stop words, and eliminates them from the sentence while tokenizing. This list process is to filter out irrelevant words during the analysis.

<u>Tokenization and POS Tagging:</u> The *extract_characters* method tokenizes each sentence and performs Part-of-Speech (POS) tagging. Tokenization splits the sentence into individual words or tokens, while POS tagging assigns parts of speech (such as noun, verb, adjective) to each token.

<u>Named Entity Recognition (NER):</u> Using Named Entity Recognition (NER) tags, the method identifies and extracts names that are likely to be characters. NER is a crucial step as it helps distinguish proper names from common nouns and other parts of speech.

<u>Character Class Creation:</u> For each identified character, a Character class instance is created, storing the character's name as a string and initializing an empty dictionary to record the emotions felt by the character throughout the novel.

```
my_character=Character(“park”)
print(my_character.emtions)

defaultdict(<class ‘list>, {}) #initial emotion dictionary
defaultdict(<class ‘list>, {“joy”: [[0], [3]], “fear”: [[6]]}) #emotion
dictionary after detecting
```

Figure 3: Character Class and its emotion attribute

2.2 Emotion Detector

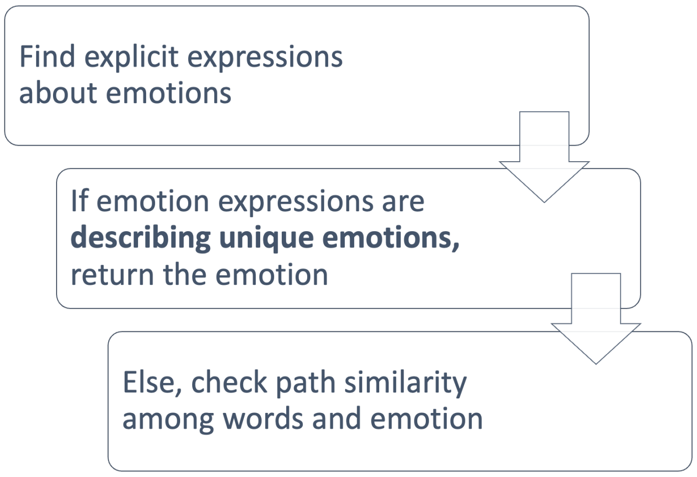
Find explicit expressions about emotions
If emotion expressions are **describing unique emotions,** return the emotion
Else, check path similarity among words and emotion

Figure 4: Pipeline of Emotion Detector

The Emotion Detector is responsible for identifying and classifying the emotions expressed in each sentence. Its operation involves several key steps:

Direct Emotion Expression Detection: The detector first checks for direct expressions of emotions within the sentence. If such expressions exist and point to a single, clear emotion, the detector immediately identifies and returns that emotion. For instance, in the sentence "I was so scared and anxious listening to that," both "scared" and "anxious" are directly associated with fear, allowing the detector to classify the emotion as "fear."

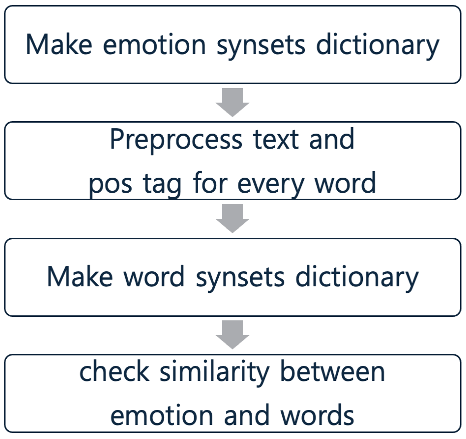
Make emotion synsets dictionary
Preprocess text and pos tag for every word
Make word synsets dictionary
check similarity between emotion and words

Figure 5: Pipepline of handling ambiguous expressions

Handling Ambiguous Expressions: In cases where direct expressions are ambiguous or absent, the Emotion Detector relies on a more sophisticated method. For example, in the sentence *"I felt a chill run down my spine as I walked through the dark alley,"* the emotion of fear is implied but not explicitly stated. To address such cases, the following steps are performed:

WordNet Synset Creation: A synset dictionary for each of the eight emotions (based on

Plutchik's model) is created using WordNet. Synsets are groups of synonyms that represent a single concept, which helps in understanding the context of words in a sentence.

Stop Word Removal and POS Tagging: The sentence is processed to remove stop words, and POS tagging is performed on the remaining words to identify their roles in the sentence.

Synset Matching: Each word in the sentence is matched against the emotion synset dictionary to calculate similarity scores. This involves creating synsets for each word in the sentence and comparing them with synsets related to the eight predefined emotions.

Emotion Classification: The similarity scores are analyzed to identify the emotion that appears most frequently among the words in the sentence. If the highest score points to a particular emotion, that emotion is assigned to the sentence. If multiple emotions have similar scores, indicating ambiguity, the detector returns None.

2.3 Sentiment Matching

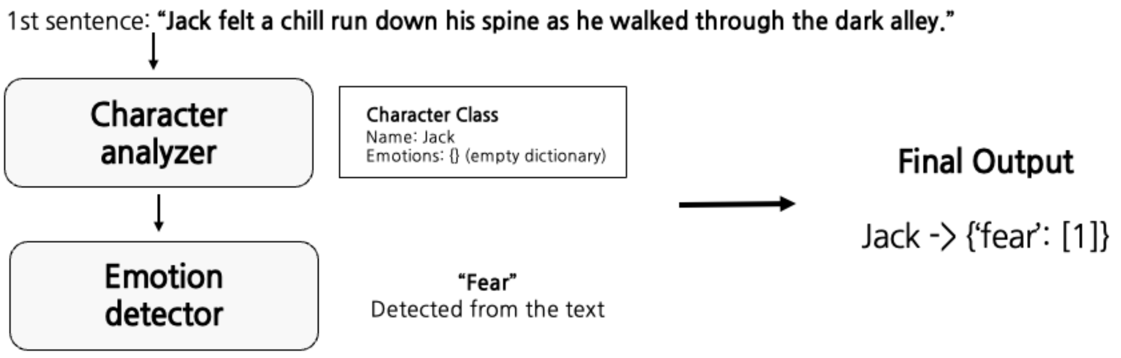
1st sentence: "Jack felt a chill run down his spine as he walked through the dark alley."
Character analyzer
Character Class
Name: Jack
Emotions: {} (empty dictionary)
Emotion detector
"Fear"
Detected from the text
Final Output
Jack -> {'fear': [1]}

Figure 6: Example of Sentiment Matching

Sentiment Matching integrates the output of the Character Analyzer and the Emotion Detector to create a comprehensive emotional profile for each character. The process includes:

Character and Emotion Integration: For each sentence, the characters identified by the Character Analyzer and the emotions detected by the Emotion Detector are combined. For instance, if the sentence "Jack felt a chill run down his spine as he walked through the dark alley" is processed, the Character Analyzer identifies "Jack" and the Emotion Detector assigns the emotion "fear."

Storing Emotional Data: This information is stored within the Character class instance for the respective character. An emotion dictionary within the Character class is updated to include the detected emotion and the index of the sentence where it was found. For example, the dictionary for "Jack" would be updated to reflect that fear was felt in sentence 1.

Cumulative Emotional Profile: As the program processes each sentence in the novel, it builds a cumulative profile of emotions for each character. This profile details which emotions were felt, when they were felt, and how frequently they occurred, providing a dynamic view of the character's emotional journey throughout the narrative.

3. Experiment & Results

4. Discussion & Conclusions

The development of the Character Emotional Analysis program marks a significant step towards automating the process of emotional analysis in literary texts. By implementing a robust Character Analyzer and a sophisticated Emotion Detector, we have created a tool that can identify and track the emotions of characters, providing valuable insights into the narrative structure.

However, there is room for improvement:

Nuanced Emotional Contexts: Enhancing the Emotion Detector to better capture nuanced emotional contexts and handle complex sentence structures would improve accuracy. This could involve more advanced natural language processing techniques and machine learning models.

Coreference Resolution: Improving the Character Analyzer to resolve coreferences (e.g., recognizing that "he" and "Jack" refer to the same character) would enhance the accuracy of character identification.

Advanced Emotion Detection: Utilizing Transformer models, such as those provided by Hugging Face, could significantly enhance the accuracy of emotion detection. These models are capable of understanding context and semantics at a much deeper level.

Our code is publicly available on GitHub, and we encourage the community to extend and refine this project further. We believe that with continued development, our Emotion Analyzer can become a powerful tool for literary analysis, educational purposes, and interactive storytelling applications.