

Natural Language Processing Final Report

Chatbot with Fuzzy Regular Expressions

Authors:

Nurbek Mauletkhan

Marina Nurgaliyeva

1. Abstract

In this project, we developed a chatbot that uses fuzzy regular expressions to enhance its ability to understand and respond to user queries. Our primary focus was to create a responsive and accessible chatbot that could handle a variety of user inputs related to restaurant reservations, menu inquiries, and other common questions. By incorporating fuzzy matching algorithms, we aimed to improve the chatbot's accuracy and flexibility in understanding user intent, even with minor typographical errors or variations in phrasing. Additionally, we implemented accessibility features to support visually impaired users.

2. Introduction

2.1 Aim

The purpose of this project was to create an intelligent chatbot capable of handling restaurant-related queries with high accuracy and user-friendliness. We aimed to enhance the chatbot's natural language understanding capabilities using fuzzy regular expressions and provide an accessible interface for visually impaired users.

2.2 Scope

The scope of this project included:

- Developing a chatbot to handle queries about restaurant reservations, menu items, location, and operating hours.
- Implementing fuzzy matching techniques to improve the chatbot's ability to understand varied user inputs.
- Adding accessibility features such as high-contrast mode, large text, and voice interaction to support visually impaired users.
- Ensuring the chatbot provides accurate and relevant responses to user queries.

2.3 Methodology

To achieve our project goals, we used the following methodology and tools:

- **Programming Language:** Python
- **Framework:** Flask for web application development
- **Fuzzy Matching:** `thefuzz` library for implementing fuzzy regular expressions
- **Accessibility:** Web technologies such as HTML, CSS, and JavaScript for creating an accessible user interface
- **Voice Interaction:** Web Speech API for voice input and output
- **Data Storage:** JSON file for storing intents and responses

I. Theoretical Part

3. Theoretical Concepts

3.1 Natural Language Processing (NLP)

NLP involves the interaction between computers and humans through natural language. The goal is to enable computers to understand, interpret, and generate human language. In this project, we used fuzzy matching to handle variations in user input.

3.2 Fuzzy Matching

Fuzzy matching is a technique used to find matches that are approximate rather than exact. This is particularly useful in natural language processing where user inputs can have typographical errors or varied phrasing. We used the `fuzz` module from the `thefuzz` library to implement token set ratio matching.

3.3 Accessibility in Web Design

Accessibility involves making web content usable for as many people as possible, including those with disabilities. We followed the Web Content Accessibility Guidelines (WCAG) to implement high-contrast modes, larger text, and voice interaction features.

Additional Theoretical Concepts from DataCamp

3.4 Levenshtein Distance

The Levenshtein distance is a metric for measuring the difference between two sequences by counting the minimum number of operations needed to transform one sequence into the other. The operations include insertion, deletion, substitution, and switching of adjacent letters.

Example of calculating Levenshtein distance:

```
def levenshtein_distance(s1, s2):
    if len(s1) < len(s2):
        return levenshtein_distance(s2, s1)

    if len(s2) == 0:
        return len(s1)

    previous_row = range(len(s2) + 1)
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
            insertions = previous_row[j + 1] + 1
            deletions = current_row[j] + 1
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row

    return previous_row[-1]
```

3.5 Practical Examples

The `thefuzz` library, formerly known as FuzzyWuzzy, uses Levenshtein distance for calculating the closeness of strings. It provides several methods for different types of fuzzy matching:

- **Simple Ratio:** Measures the basic similarity between two strings.
- **Partial Ratio:** Measures similarity by comparing the shortest string to substrings of the longer string.
- **Token Sort Ratio:** Ignores word order and compares sorted token sets.
- **Token Set Ratio:** Ignores common tokens and focuses on unique parts of the strings.

Example code for simple and partial ratios:

```
from thefuzz import fuzz

name = "Kurtis Pykes"
full_name = "Kurtis K D Pykes"

print(f"Simple ratio: {fuzz.ratio(name, full_name)}") # 86
print(f"Partial ratio: {fuzz.partial_ratio(name, full_name)}") # 67
```

Example code for token sort and token set ratios:

```
full_name_reordered = "Kurtis Pykes K D"

print(f"Token sort ratio: {fuzz.token_sort_ratio(full_name_reordered, full_name)}") # 100
print(f"Token set ratio: {fuzz.token_set_ratio(name, full_name)}") # 100
```

II. Practical Part

4. Implementation and Results

4.1 Chatbot Development

We developed the chatbot using the Flask framework. The chatbot uses a JSON file to store intents and responses. User queries are processed by normalizing the text (lowercasing and removing punctuation) and using fuzzy matching to find the best matching intent.

Example code for the Flask application:

```
# Importing flask module in the project is mandatory
# An object of Flask class is our WSGI application.
from flask import Flask

# Flask constructor takes the name of
# current module (__name__) as argument.
app = Flask(__name__)
```

```
# The route() function of the Flask class is a decorator,
# which tells the application which URL should call
# the associated function.
@app.route('/')
# '/' URL is bound with hello_world() function.
def hello_world():
    return 'Hello World'

# main driver function
if __name__ == '__main__':

    # run() method of Flask class runs the application
    # on the local development server.
    app.run()
```

4.2 Fuzzy Matching Implementation

The `find_match_token_set_ratio` method in the `Chatbot` class uses fuzzy matching to compare user inputs with predefined patterns in the intents. This allows the chatbot to understand and respond to queries even when they are not phrased exactly as expected.

Example code for preprocessing text and finding the best match:

```
def preprocess_text(self, text):
    # Normalize text by converting to lowercase and removing punctuation
    text = text.lower()
    text = re.sub(r'^\w\s', '', text)
    return text

def find_match_token_set_ratio(self, user_input, threshold=75):
    user_input = self.preprocess_text(user_input)
    best_match = 0
    match = None

    for intent in self.intents:
        for pattern in intent['patterns']:
            pattern = self.preprocess_text(pattern)
            similarity = fuzz.token_set_ratio(user_input, pattern)
            if similarity >= threshold and similarity > best_match:
                best_match = similarity
                match = intent['tag']

    return match
```

4.3 Accessibility Features

We implemented several accessibility features:

- **High-Contrast Mode:** Provides better visibility for users with visual impairments.
- **Large Text:** Increases the font size for better readability.

- **Voice Interaction:** Allows users to input queries and receive spoken responses using the Web Speech API.

Example code for accessibility features:

```
<style>
  .high-contrast {
    background-color: #000;
    color: #fff;
  }
  .large-text {
    font-size: 1.5em;
  }
  .large-buttons #send-btn, .large-buttons #voice-btn, .large-buttons #read-
btn, .large-buttons #accessibility-btn {
    font-size: 1.5em;
    padding: 15px 25px;
  }
  .large-chat-box {
    height: 500px;
  }
</style>
<script>
function toggleAccessibility() {
  const chatContainer = document.getElementById('chat-container');
  const chatBox = document.getElementById('chat-box');
  chatContainer.classList.toggle('high-contrast');
  chatContainer.classList.toggle('large-text');
  chatContainer.classList.toggle('large-buttons');
  chatBox.classList.toggle('large-chat-box');
}
</script>
```

This code defines CSS classes for high-contrast mode and larger text, and a JavaScript function to toggle these classes.

5. Summary

The goal of creating a chatbot with enhanced natural language understanding and accessibility features was successfully achieved. The fuzzy matching algorithm significantly improved the chatbot's ability to understand varied user inputs. The accessibility features made the chatbot more user-friendly for visually impaired users. Future improvements could include expanding the range of intents and responses and integrating a database for more complex data handling.

6. Bibliography

- Jurafsky, D., & Martin, J. H. (2008). *Speech and Language Processing*.
- Manning, C. D., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*.
- Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*.
- Duckett, J. (2014). *HTML and CSS: Design and Build Websites*.
- Flanagan, D. (2011). *JavaScript: The Definitive Guide*.
- W3C. (2018). *Web Content Accessibility Guidelines (WCAG) 2.1*.
- W3C. (2014). *Web Speech API Specification*.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*.
- Bray, T. (2014). *The JavaScript Object Notation (JSON) Data Interchange Format*.
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2010). *Database System Concepts*.
- "Fuzzy String Matching in Python" - DataCamp. Available at:
\\url{https://www.datacamp.com/tutorial/fuzzy-string-python}

This report provides a comprehensive overview of the chatbot project, including theoretical foundations, practical implementation details, and a discussion of the results. By following this structure, we aimed to create a clear and informative document that explains the development and functionality of our chatbot.