INF 354 Notes

Advanced concepts

Contents

Theory behind RASA

RASA is an open-source framework for building conversational AI chatbots and virtual assistants. It provides a set of tools and libraries that allow developers to create sophisticated chatbot applications that can understand and respond to user messages in a natural language format.

RASA consists of two main components:

1. RASA NLU (Natural Language Understanding): This component handles the understanding of user messages by extracting intents (the user's intention or goal) and entities (specific information relevant to the intent) from the text. It uses machine learning techniques such as natural language processing (NLP) and natural language understanding (NLU) to parse and interpret user inputs.

2. RASA Core: This component focuses on dialogue management and handles the flow of conversation. It uses machine learning algorithms to predict the next best action based on the current state of the conversation and the extracted intent and entities from RASA NLU. RASA Core allows developers to design interactive conversational flows and handle complex dialogue scenarios.

RASA chatbots can be trained on real conversational data and continuously improved through an iterative process. The framework also supports integration with various channels like websites, messaging platforms, and voice assistants, allowing the chatbots to be deployed and interact with users across multiple platforms.

Overall, RASA provides a powerful and flexible platform for building intelligent chatbot applications with natural language understanding and conversational capabilities.

What can RASA do?

- Task oriented dialogue system
  - User wants to achieve task
  - Talk to automated system in two-way conversation
- RASA is a framework that makes it easier to build custom chatbots
- The core of building a Rasa assistant is providing examples that your system learns from
  - How do people say things?
  - How do conversations go? Patterns
- Highly customizable and support everything

State machines vs. neural methods

Understanding text

- NLU (Natural Learning Understanding) = raw text in, machine readable info out
  - Rule-based
  - Neural

Terminology

- Narrow assistant –defined skill-set
- Intents -are the goals or meaning the user is trying to convey
- Entities –are important keywords, that you want to capture, that the models extract from users message
- Custom Action -getting the assistant to do something for you. E.g., connect to a database, posting data to an API, send data to a spreadsheet
- Forms –uses slots to store values

Code walk through

1. pip install rasa (install rasa) **PYTHON NEEDED = python --version **
2. rasa -h (help)
3. rasa init
4. rasa shell (command line)
5. rasa interactive (interactive learning session)
6. rasa run --enable-api(enable api)
7. rasa run –enable-api --cors “*”
8. rasa run -m model --enable-api--cors"*" -p 5005 (run on specifcport with cors)

Basic code files to make Rasa run:

- nlu.yml (data file) – create intents
- domain.yml - Register intents and Write response templates
- stories.yml - Write stories using new intents and responses

Links:

https://www.opensourceforu.com/2022/01/using-the-rasa-framework-for-creating-chatbots/

Example

Ts:

```
import { Component, ViewChild } from '@angular/core';
import { FormGroup, FormControl, Validators } from '@angular/forms';
import { MessageService } from '../service/api.service';

export interface Message {
  type: string;
  message: string;
}

@Component({
  selector: 'app-chat-support',
  templateUrl: './chat-support.component.html',
  styleUrls: ['./chat-support.component.scss'],
})
export class ChatSupportComponent {
  isOpen = false;
  loading = false;
  messages: Message[] = [];
  chatForm = new FormGroup({
    message: new FormControl('', [Validators.required]),
  });
  @ViewChild('scrollMe') private myScrollContainer: any;

  constructor(private messageService: MessageService) {
  }

  openSupportPopup() {
    this.isOpen = !this.isOpen;
  }
```

```
sendMessage() {
  const sentMessage = this.chatForm.value.message!;
  this.loading = true;
  this.messages.push({
    type: 'user',
    message: sentMessage,
  });
  this.chatForm.reset();
  this.scrollToBottom();
  this.messageService.sendMessage(sentMessage).subscribe((response: any) => {
    for (const obj of response) {
      let value
      if (obj.hasOwnProperty('text') ) {
        value = obj['text']
        this.pushMessage(value)

      }
      if (obj.hasOwnProperty('image') ) {
        value = obj['image']
        this.pushMessage(value)
      }
    }
  });
}

pushMessage(message:string){
   this.messages.push({
      type: 'client',
      message: message,
```

```
      });
      this.scrollToBottom();
    }

    scrollToBottom() {
      setTimeout(() => {
        try {
          this.myScrollContainer.nativeElement.scrollTop =
            this.myScrollContainer.nativeElement.scrollHeight + 500;
        } catch (err) {}
      }, 150);
    }
}
```

HTML:

```
<div id="assistant">
  <button id="assistant-popup-button" (click)="openSupportPopup()">
    Chat Support?
  </button>
  <div id="assistant-popup" [style.display]="isOpen ? 'block' : 'none'">
    <div id="assistant-popup-header">
      Your friendly Assistant
      <button id="assistant-popup-close-button" (click)="openSupportPopup()">
       X
      </button>
    </div>
    <div id="assistant-popup-body">
      <div class="messages" #scrollMe>
        <div *ngFor="let message of messages" class="message">
          <div [class]="message.type">
            {{ message.message }}
```

```
          </div>
        </div>
        <div
          *ngIf="loading"
          class="message"
          style="width: 100%; display: block"
        >
          <div [class]="'client'">...</div>
        </div>
      </div>
    </div>
    <form id="assistant-popup-footer" [formGroup]="chatForm">
      <input
        formControlName="message"
        type="text"
        id="assistant-popup-input"
        placeholder="Type your message here..."
      />
      <button
        id="assistant-popup-submit-button"
        [disabled]="!chatForm.valid"
        (click)="sendMessage()"
      >
        Submit
      </button>
    </form>
  </div>
</div>
```