Integrative Task II

Andrés Arango

Icesi University

Algorithms and programming III

Cali, Colombia

May 31th, 2024

Introduction

As in all integrative task, the idea is to apply the concepts seen in class, in this one the concepts based on the design of expert systems.

Objective:

The objective of this project is to develop a medium-complexity expert system using Python and the Experta library to create a chatbot for psychological healthcare. The system will utilize Bayesian networks implemented with pgmpy to provide personalized support and guidance to users experiencing psychological distress or seeking mental health assistance.

Statement

Accessing mental health services can be challenging for many individuals due to barriers such as stigma, cost, and availability of resources. A chatbot-based expert system can offer a confidential and accessible platform for users to receive psychological support, guidance, and resources.

Methodology

Expert System: An expert system is a computer program designed to emulate the decision making of a human expert in a specific field. It uses a knowledge base and logical rules to solve complex problems, offering solutions and advice as a human specialist would. These systems combine stored data and inference processes to perform diagnoses, recommendations or analysis. They are widely used in medicine, engineering, finance and other fields where specialized expertise is required to make informed and accurate decisions.

Bayesian Networks: Bayesian networks are a tool used in expert systems to represent and reason about uncertainty. A Bayesian network is a probabilistic model that uses an acyclic directed graph to show the dependence relationships between a set of variables. Each node of the graph represents a variable, and the directed edges indicate direct influences between them. Each node also has a conditional probability table that quantifies the relationships between the variables.

In the context of expert systems, Bayesian networks are valuable because they allow the integration of expert knowledge with observational data to make inferences and decisions under uncertainty. For example, in a medical expert system, a Bayesian network can help diagnose diseases based on symptoms and medical tests, taking into account the probability of various conditions. These networks allow beliefs and predictions to be updated as new information is obtained, providing a robust and flexible way to handle uncertainty and complexity in various domains.

Experta: The Python Experta library is a tool designed for the implementation of expert and rule-based systems. It facilitates the creation of systems that emulate the human decision-making process through the use of logical rules. With Experta, developers can

define a set of rules, facts and conditions to model domain-specific knowledge. The library evaluates these rules and draws conclusions from the available facts, automating complex decision making. It is useful in applications such as medical diagnostics, recommender systems and process automation, enabling intelligent, rule-based systems to be built efficiently.

Pgmpy: The pgmpy library in Python is a tool for working with probabilistic graphical models, such as Bayesian networks and Markov networks. It allows to build, manipulate and perform inferences on these models efficiently. With pgmpy, users can define network structures, learn parameters from data, and perform probabilistic inference tasks to answer questions about the variables in the model. It is useful in areas such as artificial intelligence, machine learning and scientific research, facilitating uncertainty management and probability-based decision making in complex systems.

Anxiety: Is an emotional response to perceived threat or danger. It can be adaptive in situations of real risk, but when excessive or persistent, it can become an anxiety disorder. Symptoms include constant worry, irrational fear, palpitations, sweating, and difficulty concentrating. In psychological illness contexts, anxiety can be debilitating and significantly affect quality of life.

Depression: Is a mood disorder characterized by profound and persistent sadness, loss of interest or pleasure in activities, changes in appetite and sleep, and feelings of worthlessness or guilt. Depression can be caused by biological, genetic, environmental and psychological factors. In severe cases, it can lead to suicidal thoughts and requires professional intervention for management.

Sadness: Is a normal and transitory emotion in response to negative events or losses. However, when sadness is intense, prolonged and affects daily functioning, it may be a symptom of depression. In the context of psychological illness, it is important to differentiate between normal and pathological sadness.

Stress: Is the body's response to perceived demands or challenges, known as stressors. It can be positive (eustress) when it motivates and enhances performance, or negative (distress) when it exceeds a person's coping capacity. Chronic stress can contribute to the development of mental health problems such as anxiety and depression, as well as physical problems such as cardiovascular disease.

Fatigue: Is a feeling of extreme tiredness and lack of energy that is not relieved by rest. In the context of psychological illness, it may be a symptom of depression, anxiety, or sleep disorders. Fatigue can affect work performance and daily activities, and often requires medical evaluation to determine its underlying cause.

Insomnia: Is difficulty falling asleep, staying asleep, or waking up too early and not being able to go back to sleep. It can be a symptom of anxiety disorders and depression.

Chronic insomnia affects quality of life, mood and cognitive performance, and often requires treatment to improve sleep hygiene and address underlying causes.

Happiness: is a positive emotional state associated with contentment, joy and well-being. It is a key component of mental health and can counteract the negative effects of anxiety, depression and stress. The pursuit of happiness includes pleasurable activities, positive relationships and a sense of purpose in life.

Requirements Analysis

Requirements analysis is a critical phase in the development of the chatbot system based on an expert systems architecture, designed to assist in the identification and management of psychological problems such as anxiety, depression, stress and others. This section details the functional and non-functional requirements necessary for the development of the system.

Functional Requirements

1. User Registration and Authentication:

Description: The system must allow users to register with a unique username and authenticate their identity in subsequent sessions.

Related Code:

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), nullable=False)
    created_at = db.Column(db.DateTime, default=db.func.current_timestamp())
```

2. Conversation Management:

Description: The system must allow to initiate, store and manage conversations between users and the chatbot.

Related Code:

```
class Conversation(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    started_at = db.Column(db.DateTime, default=db.func.current_timestamp())
```

3. Message Storage:

Description: The system should store all messages exchanged in each conversation for later analysis.

Related Code:

```
class Message(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    conversation_id = db.Column(db.Integer, db.ForeignKey('conversation.id'),
nullable=False)
    sender = db.Column(db.String(80), nullable=False)
    content = db.Column(db.Text, nullable=False)
    created_at = db.Column(db.DateTime, default=db.func.current_timestamp())
```

4. Chatbot Interaction:

Description: The chatbot must be able to interpret and respond to user input based on predefined rules and natural language processing algorithms.

Related Code:

```
def chat():
    user_input = request.form['user_input']
    # Processing and interpretation of the user's message
    response = generate_response(user_input)
    # Storage of the message and response in the database
    ```
```

**Non-functional Requirements**

1. Scalability:

**Description:** The system must be able to handle multiple users and conversations simultaneously without degrading performance.

**Implementation:** Use of SQLite for the database, with the possibility of migrating to a more robust database system such as PostgreSQL in the future.

2. Security:

**Description:** Measures should be implemented to protect user information, including encryption of sensitive data and secure session management.

**Implementation:** Use of best practices for secure handling of passwords and session data in Flask.

3. Usability:

**Description:** The chatbot interface should be intuitive and easy to use, providing a pleasant and effective user experience.
```

Implementation: Design of a simple and clean web interface using Flask for the backend and HTML/CSS for the frontend.

4. Performance:

Description: The system must be fast and responsive, ensuring that chatbot responses are generated and delivered in real time.

Implementation: Optimization of database queries and efficient message processing.

Technical Specifications

1. **Development Framework:**

Description: use of Flask, a Python microframework, to develop the web application.

Implementation:

```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///chatbot.db'
db = SQLAlchemy(app)
```

2. **Database:**

Description: Use of SQLite for data storage.

Implementation:

```
db.create_all()
```

3. **User Interaction:**

Description: Interaction is handled via web forms and HTTP requests.

Implementation:

```
@app.route('/chat', methods=['POST'])
def chat():
    user_input = request.form['user_input']
```

Requirements analysis is critical to ensure that the system meets the expectations and needs of the users. By clearly identifying functional and non-functional requirements, and specifying the technologies and methodologies to be used, a solid foundation is established for the effective development, implementation and testing of the chatbot system. This methodical approach ensures that the final product will be robust, secure and capable of providing meaningful support to users in managing their psychological problems.

Knowledge Acquisition and Representation

In this section, the “experta” library was used to define facts, rules and a knowledge engine that will guide the chatbot's interaction with users.

1. Definition of Facts:

Fact classes were created to represent messages, explanations, symptoms, emotions, and contexts. For example:

```
class Message(Fact):
    pass
```

2. Definition of Rules:

Fact-based rules were defined to handle different situations and generate appropriate responses. For example:

```
@Rule(Message(content="Hello"))
def greet(self):
    self.declare(Fact(response="Hello! How can I help you?"))
```

3. Bayesian Inference:

A Bayesian inference function was implemented to predict emotions and symptoms based on emotional and situational contexts. For example:

```
def infer_response(self, emotion, symptom, context):
    # Lógica de inferencia bayesiana
```

4. Psychological Explanations:

Rules were added to provide explanations for psychological problems and family and work contexts. For example:

```
@Rule(Symptom(type='depression'))
def explain_depression(self):
    self.declare(Fact(explanation="Depression can be caused by a combination
of factors..."))
```

Knowledge acquisition and representation through expert library and Bayesian inference enables the chatbot to interpret user input, understand their emotional and situational state, and provide relevant and insightful responses. The ability to explain psychological issues and contexts helps users better understand their experiences and find ways to manage them in a

healthy way. This methodology strengthens the functionality of the chatbot as an effective and comprehensive psychological support tool.

System Design

The system design of the chatbot is based on client-server architecture, where the server implements the business logic and the interaction with the database, while the client (user interface) provides the interface for users to interact with the chatbot.

Client-Server Architecture:

- The server acts as a Knowledge Engine that processes user input, applies business rules and generates appropriate responses. It uses the expert library to represent and manipulate knowledge, as well as Bayesian inference to predict emotions and symptoms.
- The client provides an intuitive user interface where users can send messages to the chatbot and receive responses. It was implemented using Flask, a Python library for web development.

System Components:

- Backend (Server):

The server is composed of several classes and rules defined in Python, using the expert library for the rule engine and Bayesian inference.

SQLAlchemy was used to interact with the SQLite database and store user profiles, conversations and messages.

- Frontend (Client):

The client consists of a web interface built with HTML, CSS and JavaScript, which communicates with the server through HTTP requests using AJAX.

A chat interface was implemented that displays chatbot and user messages, and allows new messages to be sent.

Interaction Flow:

- When a user sends a message, the client sends it to the server via an HTTP request.
- The server processes the message using business rules and Bayesian inference, and generates a response.
- The response is sent back to the client and displayed in the chat interface.

Session and Data Management:

- Flask-Session was used to manage user sessions on the server and maintain conversation state.

- User data, conversations and messages are stored in the SQLite database and accessed through SQLAlchemy.

Scalability and Maintainability:

- The system design allows for easy scalability by adding new rules and functionality to the Knowledge Engine.
- The separation of frontend and backend business logic facilitates future maintenance and expansion of the system.
- In summary, the chatbot system design provides a robust and modular architecture that facilitates effective user interaction, data management and future system scalability.

Implementation Details

In this section, we describe the implementation details of the chatbot, including knowledge representation using a Bayesian network and rule-based business logic.

Bayesian Network:

- The Python pgmpy library was used to build a Bayesian network that models the relationship between user emotion, context, and associated symptoms.
- The network consists of three nodes: emotion, context and symptom, with directional arcs representing the probabilistic dependencies between them.
- Conditional probability distributions (CPDs) were defined for each node, specifying the probabilities of the states conditioned by their parents in the network.
- Inference is performed using the variable elimination method to predict the most likely emotion and symptom given the emotional state and context provided by the user.

Chatbot logic:

- Chatbot logic was implemented using the Python expert library to define production rules.
- A Chatbot class was created that inherits from KnowledgeEngine to declare rules based on user messages and generate corresponding responses.
- The rules are designed to handle a variety of emotional and contextual situations, such as anxiety, depression, stress, and family or work problems.
- Each rule triggers a Bayesian inference function to predict the chatbot's most likely response based on the user's emotional state and context.

Integration with Flask:

- Chatbot logic was integrated with a Flask server to provide an interactive web interface.

- Routes and controllers were created to handle client requests, send messages to the chatbot and receive responses.
- A chat interface was implemented in the web interface using HTML, CSS and JavaScript to display chatbot and user messages in an orderly fashion.

Data Persistence:

- SQLAlchemy was used to interact with a SQLite database to store user profiles, conversations, and messages.
- Data models were implemented to represent users, conversations and messages, and relationships between them were configured to facilitate data access and manipulation.

Together, these implementation details provide a robust and efficient system that enables effective user-chatbot interaction by integrating rule-based knowledge representation and probabilistic inference using a Bayesian network.

Deployment

The project deployment was carried out following a modular and scalable approach, using modern technologies and specific tools for each component of the system. The key aspects of the deployment are detailed below:

Backend development:

- Flask, a lightweight Python web development framework, was used to build the chatbot backend.
- Routes and controllers were created to handle incoming HTTP requests, such as sending messages to the chatbot and receiving responses.
- SQLAlchemy was used to interact with the SQLite database and perform CRUD (Create, Read, Update, Delete) operations on user profiles, conversations and messages.
- Data models were implemented using the User, Conversation and Message classes to represent the database structure and establish relationships between them.

Frontend development:

- An intuitive and responsive user interface was designed using HTML, CSS and JavaScript to interact with the chatbot.
- A chat interface was created that displays chatbot and user messages in an orderly fashion, with additional functionality such as automatic scrolling down and notifications of new messages.

Knowledge Representation:

- The Python expert library was used to define production rules and represent the chatbot's business logic.
- Classes were created to represent domain relevant facts, such as Message, Symptom, Emotion and Context, and rules were defined to infer chatbot responses based on these facts.

Probabilistic Inference:

- A Bayesian network was implemented using the Python pgmpy library to model the relationship between user emotion, context, and associated symptoms.
- Conditional probability distributions (CPDs) were defined for each node in the network, specifying the probabilities of the states conditioned by their parents.
- Variable elimination method was used to perform inference and predict the most likely emotion and symptom given the emotional state and context provided by the user.

Integration and Deployment:

- The backend and frontend were integrated into a single system and deployed in a production environment, such as a web server or cloud platform.
- Extensive testing was performed to ensure the chatbot functioned correctly on different browsers and devices, as well as to validate its accuracy and effectiveness in interacting with users.

In summary, the project implementation covered multiple aspects, from backend and frontend development to knowledge representation and probabilistic inference, all with the goal of creating an effective and efficient chatbot for the care of mental health problems.

Conclusion

In conclusion, the development of this chatbot represents a significant step towards improving the accessibility of psychological and mental health support services. The combination of artificial intelligence techniques, such as knowledge representation and probabilistic inference, with modern web development tools has enabled the creation of an interactive and responsive platform for users seeking guidance and support in times of emotional distress.

As for future work, there are several areas of improvement and expansion that could be explored. For example, a feedback system could be implemented to evaluate the effectiveness and accuracy of the chatbot's responses, thus enabling its continuous learning and refinement. In addition, it would be beneficial to integrate real-time sentiment analysis functionalities to proactively detect and respond to changes in the user's emotional state. Finally, the integration of emerging technologies, such as advanced natural language processing and artificial emotional intelligence, could be considered to provide an even more personalized and

effective experience for users. In summary, the potential for expansion and improvement of this chatbot is vast and represents an exciting opportunity to further advance the field of digital mental health.