First Aid Chatbot Companion: A Machine Learning-Based Medical Assistant

1. Introduction

This report presents the development of a First Aid Chatbot designed to assist users with medical inquiries related to first aid. The chatbot leverages a combination of machine learning, natural language processing (NLP), logical reasoning, and probabilistic reasoning to understand user queries and provide relevant responses. The system continuously improves its accuracy over time by learning from user feedback and retraining the model.

The chatbot classifies user input into predefined categories (intents) and provides medical advice based on these classifications. Additionally, the chatbot incorporates both rule-based reasoning and probabilistic models to give more accurate and context-sensitive answers.

2. Objectives

The main objectives of this project are:

- To create a chatbot that can provide first aid assistance.
- To implement machine learning techniques (using LinearSVC) for intent classification.
- To enhance the chatbot’s decision-making process using logical reasoning (e.g., if both "fever" and "headache" are detected, suggest the flu) and probabilistic reasoning (e.g., using Bayesian inference to predict illness probability).
- To incorporate a feedback learning mechanism that allows the chatbot to improve and adapt based on user interactions.

3. Libraries and Technologies Used

This chatbot utilizes several libraries to support various functionalities:

- **random**: For selecting random responses from a list of possible responses.
- **json**: For reading and writing data in JSON format, used to load the intent dataset.
- **sklearn (Scikit-learn)**:
    - **TfidfVectorizer**: Converts text data into numerical features based on term frequency and inverse document frequency (TF-IDF).
    - **LinearSVC**: A Support Vector Classifier (SVC) for classifying user inputs based on features extracted using TF-IDF.
    - **train_test_split**: Splits the dataset into training and testing sets to evaluate the model's performance.
    - **make_pipeline**: Combines the preprocessing and model fitting into a single workflow.
    - **accuracy_score, precision_score, recall_score, f1_score**: Metrics to evaluate the performance of the machine learning model.
- **nltk (Natural Language Toolkit)**:
    - **WordNetLemmatizer**: Reduces words to their base form (lemmatization).
    - **stopwords**: A list of common words like "the" and "a" that are removed during preprocessing.
    - **nltk.download**: Downloads necessary resources for tokenization and stopwords removal.

4. Methodology

4.1 Data Preprocessing

The input data is preprocessed to make it suitable for machine learning. The following steps are applied:

- **Tokenization**: The input sentences are split into words.

- **Lemmatization**: Words are reduced to their base forms (e.g., "running" becomes "run").
- **Stopword Removal**: Common words like "the" and "is" are removed since they don’t contribute meaningful information to the classification.

The data is loaded from a JSON file (**first-aid-datasets-updated-2.json**) that contains user input patterns, responses, and tags (intents). The dataset is then split into training and testing sets.

Preprocessing Code:

```
# Preprocess patterns for training

def preprocess_sentence(sentence):

    # Tokenize, lemmatize, and remove stop words

    tokens = sentence.split()

    tokens = [lemmatizer.lemmatize(word.lower()) for word in tokens if word.lower()
not in stop_words]

    return ' '.join(tokens)
```

4.2 Model Training

The machine learning model uses a TF-IDF vectorizer to convert user input into numerical features, which capture important words and their frequencies. A LinearSVC model is then used to classify these inputs into predefined intents.

The model is trained on the preprocessed dataset, and its performance is evaluated using metrics like accuracy, precision, recall, and F1-score.

Model Training Code:

```
# Create and train the machine learning model

model = make_pipeline(

    TfidfVectorizer(max_features=None, ngram_range=(1, 2)),

    LinearSVC()
)
model.fit(X_train, y_train)
```

4.3 Logical and Probabilistic Reasoning

The chatbot employs logical reasoning to provide medical advice based on specific conditions. For example, if both "fever" and "headache" are present, the chatbot might suggest that the user could have the flu.

Additionally, Bayesian reasoning is used to estimate the likelihood of illness based on symptoms. Each symptom is assigned a probability, and the probabilities are multiplied to calculate the overall chance of illness.

Code for Logical Reasoning:

```
# Logical Reasoning Rules
logical_rules = {
    ("fever", "headache"): "You might have the flu. Rest and hydrate.",
    ("cut", "bleeding"): "Apply pressure to stop the bleeding and clean the wound."
}

def logical_reasoning(user_input_tokens):
    """Perform logical reasoning based on input tokens."""
    for rule_conditions, conclusion in logical_rules.items():
        if all(condition in user_input_tokens for condition in rule_conditions):
            return conclusion
    return None
```

Code for Probabilistic Reasoning:

```
# Probabilistic Reasoning
def bayesian_reasoning(symptoms):
    """Perform probabilistic reasoning using a basic Bayesian model."""
    probabilities = {"fever": 0.8, "headache": 0.7, "nausea": 0.5}
    posterior = 1
    for symptom in symptoms:
        posterior *= probabilities.get(symptom, 0.1)
    return f"Based on symptoms, there is a {posterior * 100:.2f}% chance of illness."
```

4.4 Knowledge Representation (Ontology)

An ontology is used to map medical concepts such as "fever", "headache", and "bleeding" to related keywords. This helps the chatbot understand the context of user queries and respond more accurately.

Ontology Code:

```
ontology = {
    "fever": [],
    "bleeding": [],
    "headache": []
}
```

4.5 Feedback Learning Mechanism

The chatbot can improve over time by incorporating user feedback. If the chatbot is unable to confidently predict an intent, it asks for feedback, which is then added to the training data. The model is retrained with the updated data, allowing it to improve.

Feedback Learning Code:

```
def add_feedback_to_data(user_input, predicted_intent):
    """Add user feedback (new data) to the training set for future learning."""
    new_data = {"patterns": [user_input], "tag": predicted_intent, "responses": ["Thank you for your input!"]}
    data['intents'].append(new_data)  # Append to existing dataset
    patterns.append(preprocess_sentence(user_input))  # Preprocess and add pattern
    tags.append(predicted_intent)  # Add predicted intent tag
    # Retrain model with the updated data
    X_train_new, _, y_train_new, _ = train_test_split(patterns, tags, test_size=0.2, random_state=42)
    model.fit(X_train_new, y_train_new)
```

5. Results

The model was evaluated using the test dataset. The performance metrics are as follows:

- **Accuracy: 94.87%**
- **Precision: 0.94%**
- **Recall: 0.95%**
- **F1-Score: 0.94%**

These results demonstrate that the chatbot is capable of accurately classifying user input into predefined intents and providing relevant responses.

6. Conclusion

This First Aid Chatbot successfully combines machine learning, logical reasoning, and probabilistic reasoning to provide first aid advice. The chatbot uses NLP techniques for text preprocessing and LinearSVC for classification. Additionally, the feedback learning mechanism ensures continuous improvement based on user interactions.

The chatbot has the potential to be expanded to handle more medical inquiries, improve its reasoning capabilities, and provide more detailed responses based on an enhanced knowledge base.