CPSC481-01 Project Report:
Tuffy Chatbot AI Tutor

Team Members:
Zhihuang Chen
Justin Jhern
Taylor Kim
David Navarro

Github link: https://github.com/zChen-1/CPSC-481-Final-Project.git

Table of Contents

Problem Statement

A chatbot is a software application that can imitate human-like communication to interact with a user in a chat. Rule based chatbots are generally faster to train and easy to implement, but they are incapable of learning or adapting. They cannot improve their responses over time because they primarily rely on their predefined rules to interact with the user. Tuffy Chatbot AI Tutor project aims to implement a machine learning algorithm to provide better response to a question and user engagement with a friendly user interactivity. In addition, the chatbot focuses on educational purposes to provide easy access to artificial intelligence course information for students. It will provide a user with a walk-through of CPSC481 lessons, informative responses in which students ask for a specific topic, quizzes for lessons learned, and simple mathematical calculations.

Approach

Tuffy Chatbot AI Tutor utilizes TF-IDF Vectorization and Multinomial Naive Bayes algorithms. TF-IDF is used to convert text data into a matrix of TF-IDF features and MultinomialNB is used to calculate the probability of the term appearing in text. Although current datasets are not big enough to cover all the general questions which users may have, these simple datasets, created by team members, allowed us to train and test our model faster. There are four datasets created to train and test our model "definition.csv", "responses.csv", "training_data", and "training_data_keyWord.cvs".

Project is implemented using open-source libraries: flask (to integrate front-end and back-end), scikit-learn (train, test, pipeline, and evaluate model), pandas (load data), and sympy (to handle mathematical questions) to create a machine learning based chatbot. Programming languages used were HTML, CSS, JavaScript (front-end), and Python (back-end).

Software Description and Requirements

Before running please install these libraries/packages:

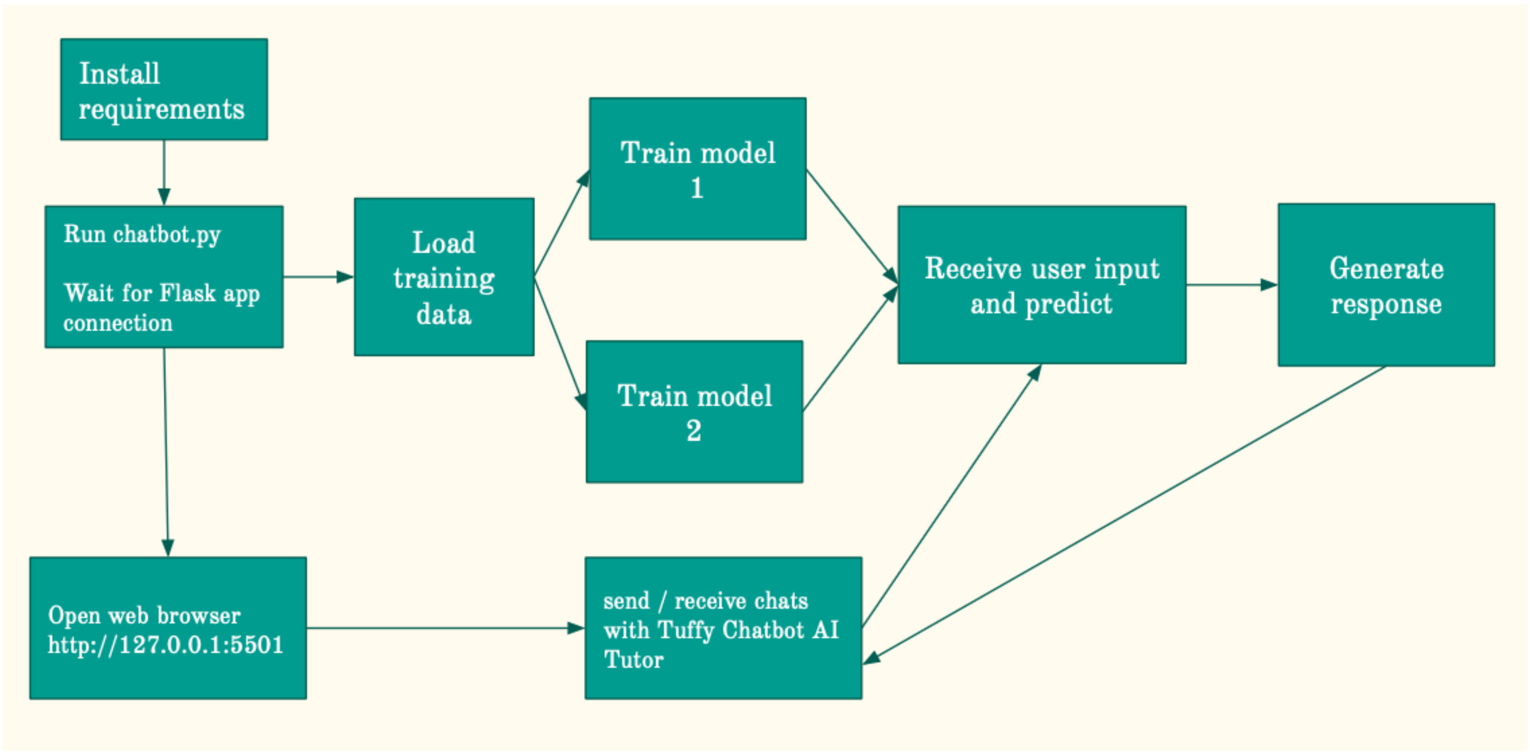
```
python.exe -m pip install --upgrade pip

pip install scikit-learn pandas flask
```

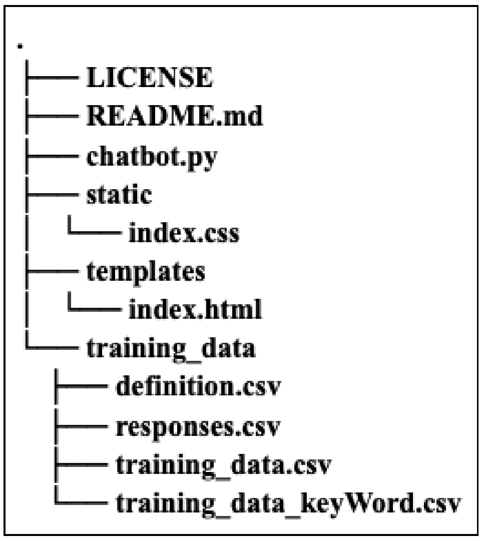
After installing required libraries/packages (to run):

1. Open terminal or IDE
2. Change directory to where downloaded file is located
3. Run chatbot.py (python3 chabot.py)

4. Wait until Flask app is connected
5. Open web browser with http://127.0.0.1:5501
6. Chat with TuffyChatbot

Diagram for components and input/output relationships:


Install requirements
Run chatbot.py
Wait for Flask app connection
Load training data
Train model 1
Train model 2
Receive user input and predict
Generate response
Open web browser http://127.0.0.1:5501
send / receive chats with Tuffy Chatbot AI Tutor



```
.
├── LICENSE
├── README.md
├── chatbot.py
├── static
│   └── index.css
├── templates
│   └── index.html
└── training_data
    ├── definition.csv
    ├── responses.csv
    ├── training_data.csv
    └── training_data_keyWord.csv
```

Folder/file Layout:

LICENSE: explains what is allowed to be done with the software.
README.md: requirements and how to run.
chatbot.py:

- main file to run and execute the program.
- responsible for integrating front-end and back-end via Flask.
- loads datasets to train and test.
- handles user inputs and sends its responses to UI.

static - index.css: used to style the UI (index.html).
templates - index.html: used to structure the website page and contents.
training_data

definition.cvs: contains pairs of keywords and definitions.
responses.cvs: contains pairs of intents and responses.
training_data.cvs: contains pairs of utterances and intents.
training_data_keyWord.csv: contains pairs of keywords and questions.

Model design for Natural Language Processing(NLP)

First, the model will retrieve the user's input from the front-end. When the user enters keywords for mathematical calculations such as "+”, "-”, "*”, "/”, "add”, "plus”, "subtract”, and "minus”, the model will calculate the user input.

```
# Check for math expression first
if re.search(r'x\b', user_input):  # Checks characters (like 'x'), but only work with 'x' now
    return handle_algebra(user_input)
if re.search(r'\b\d+\s*(\+|\-|\*|\/|add|plus|subtract|minus|multiply|times|divide|by)\s*\d+\b', user_input,
             re.IGNORECASE):
    return handle_math(user_input)
```

In addition, if the content of the user's input is non-computation content, the model will predict the content of the user's input. The model will predict that the content of the user's input is a "greeting" or "definition".

```
# Predict the intent
predicted_intent = model.predict([user_input])[0]
predicted_keyword = model_kw.predict([user_input])[0] if predicted_intent == 'definition' else None
```

```
# Handling different intents
if predicted_intent == 'greeting':
    return random.choice(response_dict.get('greeting', ["Hello! How can I assist you?"]))
elif predicted_intent == 'definition':
    return definition_dict.get(predicted_keyword, "I'm not sure how to respond to that.")
elif predicted_intent in response_dict:
    return random.choice(response_dict[predicted_intent])
else:
    return "I'm not sure how to respond to that."
```

When the model understands the content entered by the user, the model will predict the focus of the content and return a corresponding reply. For example, if the user asks about a definition, the model will predict the focus of the user's input (keyword) and give the corresponding answer (definition).

```
def find_definition(query):
    words = query.lower().split()
    for word in words:
        if word in definition_dict:
            return definition_dict[word]
    return None
```

Evaluation Metrics

Using the scikit-learn library, “sklearn.metrics.accuracy_score()”, our chatbot ended up with 73.08% for model accuracy and 61.54% accuracy for model to find the keyword in the sentence provided by the user. We’ve also tested by chatting with the chatbot with a variety of mathematical questions and other questions.

Downside of the chatbot is when the sentences entered by the user are similar, the keywords cannot be found accurately. This is because traditional models such as Naive Bayes combined with TF-IDF vectorization do not understand the context of words well. They process each word or term independently and cannot capture the order or proximity of words. For instance, when user type "AI", "A search" or "A* search", the model may misidentify them.

Conclusion and Future Work

We were able to learn how to implement machine learning into a chatbot, utilizing algorithms, creating datasets for training and testing a model, and integrating the back-end and front-end using the flask framework. Although our chatbot is not strong enough to provide all the functionalities we proposed and may interpret users’ questions incorrectly, we enjoyed working on this project to learn the above lessons from the project.

Future Work/Improvements

- We can expand the training data. Make sure the data set contains a diverse set of examples of each keyword, including the various ways users might refer to them.
- Use word embeddings like **Word2Vec** or **GloVe** instead of **TF-IDF** so that the model can capture the semantics of the word more effectively.
- Adopt models that use contextual embeddings such as **BERT**. These models aim to understand the context of each word in a sentence.
- Consider more complex frameworks that allow models to capture sequences and relationships between words, such as **LSTM** or **GRU**.
- Implement other functionalities such as questions other than definitions and quizzes for the students to take to provide instant feedback and engagement.

References

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

https://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html