UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

Facultad de Ingeniería

Ingeniería en Ciencias y Sistemas

Inteligencia Artificial

Manual Técnico

ChatBot

Maria Isabel Masaya Cordova

Lesther Kevin Federico López Miculax

Kevin Uriel Ramírez

Guatemala, Guatemala, 02 de Enero del 2025

Introducción

Este manual técnico describe el funcionamiento del chatbot desarrollado en el repositorio proporcionado. Se enfoca en los archivos clave chatbot.py, classes.pkl, new.py, words.pkl y describe fragmentos de código relevantes. El chatbot está diseñado para interactuar con usuarios en lenguaje natural, comprender sus intenciones (intents) y proporcionar respuestas predefinidas o realizar acciones específicas.

Objetivos

- Describir la arquitectura y el funcionamiento del chatbot.
- Detallar el proceso de entrenamiento del modelo de clasificación de intents.
- Explicar la función de cada archivo clave en el repositorio.
- Ilustrar el uso de fragmentos de código relevantes.

Resumen

El chatbot utiliza un modelo de red neuronal para clasificar las intenciones del usuario a partir de la entrada de texto. El modelo se entrena con un conjunto de datos de intents predefinidos y sus correspondientes patrones de frases. Una vez entrenado, el chatbot puede predecir el intent del usuario y proporcionar una respuesta adecuada.

Este código implementa un chatbot con una interfaz gráfica de usuario (GUI) utilizando la biblioteca Tkinter en Python. El chatbot utiliza un modelo de aprendizaje automático previamente entrenado para comprender y responder a las entradas del usuario.

Funcionalidades Principales:

- **Procesamiento del Lenguaje Natural (PNL):**
    - clean_up_sentence(sentence): Tokeniza la entrada del usuario y lematiza las palabras, preparándolas para el análisis.
    - bow(sentence, words): Crea una representación de "bolsa de palabras" (bag-of-words) de la entrada, que es un vector numérico que indica la presencia de palabras del vocabulario en la frase.
- **Predicción de Intención:**
    - predict_class(sentence): Utiliza el modelo de aprendizaje automático (chatbot_model.h5) para predecir la intención (intent) del usuario a partir de la representación de bolsa de palabras. Reduce el umbral de error (ERROR_THRESHOLD) para capturar más intenciones.

- **Generación de Respuesta:**
    - get_response(tag): Busca la intención predicha en el archivo intents2.json y devuelve una respuesta aleatoria de las definidas para esa intención.
    - generate_response(user_input): Combina la predicción de intención y la selección de respuesta para generar una respuesta completa al usuario.
- **Interfaz Gráfica de Usuario (GUI - Tkinter):**
    - ChatBotApp:
        - Inicializa la ventana principal y configura su apariencia (título, tamaño, colores, fuente).
        - Crea un área de texto (self.text_area) para mostrar la conversación, con una barra de desplazamiento (self.scrollbar).
        - Define etiquetas (tag_config) para formatear los mensajes del usuario y del bot, así como para mostrar código.
        - Crea un campo de entrada (self.entry_box) para que el usuario escriba sus mensajes.
        - Agrega un botón "Enviar" (self.send_button) para enviar los mensajes.
        - send_message(self): Gestiona el envío de mensajes, muestra el mensaje del usuario en el área de texto, genera una respuesta utilizando generate_response y muestra la respuesta del bot.
        - display_message(self, message, sender): Inserta el mensaje en el área de texto con el formato adecuado (usuario o bot) y maneja la visualización de código si el mensaje lo incluye.

Manual Técnico

Este manual técnico describe el funcionamiento del chatbot desarrollado en el repositorio proporcionado. Se enfoca en los archivos clave chatbot.py, classes.pkl, new.py, words.pkl y describe fragmentos de código relevantes.

chatbot.py

Este archivo contiene el código principal del chatbot.

Fragmentos de código:

```
def bag_of_words(s, words):
    bag = [0 for _ in range(len(words))]

    s_words = nltk.word_tokenize(s)
    s_words = [stemmer.stem(word.lower()) for word in s_words]

    for se in s_words:
        for i, w in enumerate(words):
            if w == se:
                bag[i] = 1

    return numpy.array(bag)
```

Función “Chat()”

Esta función convierte una frase en un vector numérico (bag-of-words) que representa la presencia de palabras del vocabulario en la frase. Se utiliza para la clasificación de intents.

```python
def chat():
    print("Start talking with the bot (type quit to stop)!")
    while True:
        inp = input("You: ")
        if inp.lower() == "quit":
            break

        results = model.predict([bag_of_words(inp, words)])[0]
        results_index = numpy.argmax(results)
        tag = labels[results_index]

        if results[results_index] > 0.7:
            for tg in data["intents"]:
                if tg['tag'] == tag:
                    responses = tg['responses']
            print(random.choice(responses))
        else:
            print("I didn't get that, try again.")
```

Esta función maneja la interacción con el usuario. Recibe la entrada del usuario, la procesa, predice el intent y genera una respuesta.

classes.pkl

Este archivo contiene una lista de Python con los intents del chatbot. Los intents se definen en un archivo JSON (intents.json) y se cargan en el chatbot.

Ejemplo de un intent:

```json
{
  "tag": "greeting",
  "patterns": [
    "Hi",
    "How are you",
    "Is anyone there?",
    "Hello",
    "Good day"
  ],
  "responses": [
    "Hello, thanks for visiting",
    "Good to see you again",
    "Hi there, how can I help?"
  ],
  "context_set": ""
}
```

new.py

Este archivo contiene el código para entrenar el modelo de clasificación de intents.

Fragmentos de código:

- **Creación de datos de entrenamiento:**

```
training = []
output_empty = [0] * len(classes)
for doc in documents:
    bag = []
    pattern_words = doc[0]
    pattern_words = [stemmer.stem(word.lower()) for word in pattern_words]
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)

    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1

    training.append([bag, output_row])
```

Este fragmento de código crea los datos de entrenamiento para el modelo. Convierte las frases de entrenamiento en vectores numéricos y crea las etiquetas correspondientes.

- **Entrenamiento del modelo:**

```
train_x = list(training[:,0])
train_y = list(training[:,1])
model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5, verbose=1)
model.save('chatbot_model.h5', hist)
```

Este fragmento entrena un modelo de red neuronal para clasificar los intents. Se define la arquitectura del modelo, se compila y se entrena con los datos de entrenamiento.

- **words.pkl**

Este archivo contiene una lista de Python con todas las palabras únicas (vocabulario) del chatbot. Estas palabras se extraen de las frases de entrenamiento en el archivo JSON (intents.json).

Ejemplo de la lista de palabras:

```
['Hola', 'como', 'estas', 'que', 'tal', 'bien', 'y', 'tu', 'mal', 'tengo', 'un', 'problema', 'puedes', 'ayudarme']
```

- **Predicción de la intención con umbral de error reducido:**

```
ERROR_THRESHOLD = 0.3  # Reducir umbral para captar más intenciones
results = [[i, r] for i, r in enumerate(res) if r > ERROR_THRESHOLD]
```

- **Combinar respuestas con saltos de línea:**

```
return "\n".join(intent['responses'])
```

- **Mostrar código con formato especial:**

```
self.text_area.insert(END, f"```\n{part.strip()}\n```\n\n", "code")
```

Lógica de respuesta y capacidad de aprendizaje:

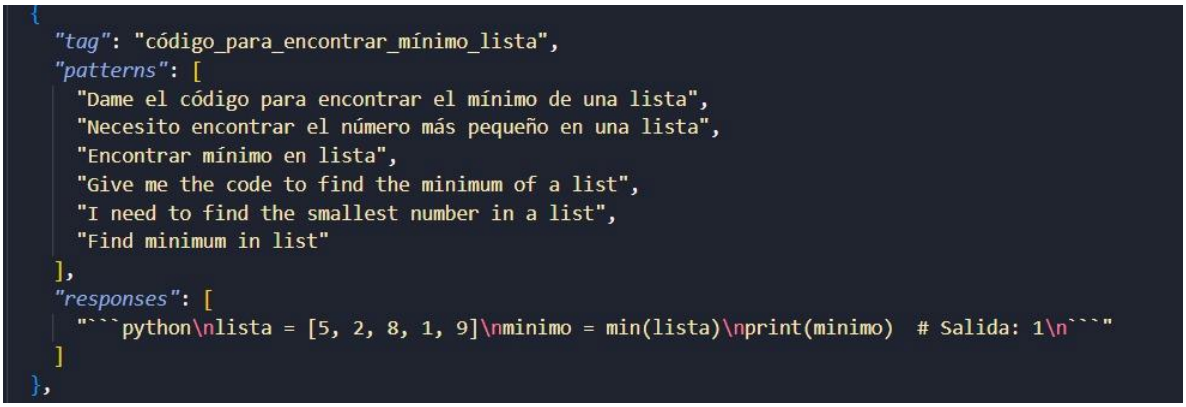
- El chatbot utiliza un modelo de aprendizaje automático (chatbot_model.h5) que ha sido entrenado previamente en un conjunto de datos de intenciones y respuestas (intents2.json).
- La función predict_class(sentence) utiliza este modelo para predecir la intención del usuario a partir de su entrada.
- El chatbot tiene la capacidad de aprender y mejorar sus respuestas a medida que se entrena con más datos. Al agregar más ejemplos de intenciones y respuestas al conjunto de datos y reentrenar el modelo, el chatbot puede comprender y responder a una gama más amplia de preguntas y solicitudes.

Lógica de respuesta

- El chatbot utiliza un modelo de aprendizaje automático (chatbot_model.h5) que ha sido entrenado previamente en un conjunto de datos de intenciones y respuestas (intents2.json). Este conjunto de datos contiene una serie de

ejemplos de frases que los usuarios podrían escribir, etiquetadas con la intención que representan.

- **Ejemplo de intents2.json:**



```
{
  "tag": "código_para_encontrar_mínimo_lista",
  "patterns": [
    "Dame el código para encontrar el mínimo de una lista",
    "Necesito encontrar el número más pequeño en una lista",
    "Encontrar mínimo en lista",
    "Give me the code to find the minimum of a list",
    "I need to find the smallest number in a list",
    "Find minimum in list"
  ],
  "responses": [
    "```python\nlista = [5, 2, 8, 1, 9]\nminimo = min(lista)\nprint(minimo)  # Salida: 1\n```"
  ]
},
```

En este fragmento de intents2.json, se define una intención específica etiquetada como código_para_encontrar_mínimo_lista. Esta intención está diseñada para responder a las solicitudes de los usuarios que buscan obtener el código para encontrar el valor mínimo dentro de una lista.

Análisis del intent:

- **tag:** código_para_encontrar_mínimo_lista
  - Esta etiqueta actúa como un identificador único para esta intención en particular.
- **patterns:**
  - Incluye una serie de frases en español e inglés que un usuario podría escribir para solicitar el código.
  - Ejemplos: "Dame el código para encontrar el mínimo de una lista", "Necesito encontrar el número más pequeño en una lista", "Find minimum in list".
- **responses:**
  - Contiene un fragmento de código Python que encuentra el mínimo de una lista.

En el intents2.json se agregan soluciones de código que pueden devolver respuestas en Python y Javascript, según sea la necesidad del usuario.

Cuando un usuario escribe un mensaje, el chatbot lo procesa y lo convierte en una representación numérica llamada "bolsa de palabras" (bag-of-words). Esta representación se utiliza como entrada para el modelo de aprendizaje automático, que predice la intención del usuario.

Función predict_class(sentence):

```
def predict_class(sentence):
    p = bow(sentence, words)  # Crear la bolsa de palabras
    res = model.predict(np.array([p]))[0]  # Predecir la intención
    ERROR_THRESHOLD = 0.3  # Umbral de error
    results = [[i, r] for i, r in enumerate(res) if r > ERROR_THRESHOLD]
    results.sort(key=lambda x: x[1], reverse=True)
    return classes[results[0][0]] if results else None
```

Una vez que se ha predicho la intención, el chatbot busca en el archivo intents2.json la respuesta correspondiente a esa intención. Si encuentra la intención, selecciona una respuesta aleatoria de las definidas para esa intención y la devuelve al usuario.

Función get_response(tag):

```
def get_response(tag):
    for intent in intents['intents']:
        if intent['tag'] == tag:
            return "\n".join(intent['responses'])  # Combinar respuestas con salto
    return "Lo siento, no entiendo tu pregunta."
```

Capacidad de aprendizaje

El chatbot tiene la capacidad de aprender y mejorar sus respuestas a medida que se entrena con más datos. Al agregar más ejemplos de intenciones y respuestas al conjunto de datos intents2.json y reentrenar el modelo chatbot_model.h5, el chatbot puede comprender y responder a una gama más amplia de preguntas y solicitudes.

Proceso de aprendizaje:

1. Se agregan nuevos ejemplos de frases y sus correspondientes intenciones al archivo intents2.json y intents4.json
2. Se ejecuta el script de entrenamiento (new.py) para reentrenar el modelo de aprendizaje automático con los nuevos datos.
3. El nuevo modelo, con una capacidad de comprensión mejorada, se guarda como chatbot_model.h5.

Funcionamiento general del chatbot

1. El chatbot carga los intents, las palabras y el modelo de clasificación.
2. El usuario ingresa una frase.
3. La frase se convierte en un vector numérico (bag-of-words).
4. El modelo predice el intent de la frase.
5. Si la predicción tiene una confianza alta, el chatbot selecciona una respuesta aleatoria del intent correspondiente.
6. Si la confianza es baja, el chatbot pide al usuario que reformule la pregunta.

Este manual técnico proporciona una descripción general del funcionamiento del chatbot y de los archivos clave involucrados. Para obtener información más detallada, se recomienda consultar el código fuente en el repositorio.

El proceso de aprendizaje implica actualizar este archivo intents2.json y reentrenar el modelo de aprendizaje automático. Los pasos son los siguientes:

- Agregar nuevos ejemplos: Se agregan nuevas frases a la lista patterns de las intenciones existentes o se crean nuevas intenciones con sus correspondientes patterns y responses.

  Ejemplo:

```
{
  "tag": "saludo_informal",
  "patterns": ["Qué onda", "Hola que hace", "Cómo va"],
  "responses": ["¡Hola! ¿Cómo estás?", "Qué tal, ¿en qué te puedo ayudar?"]
}
```

Reentrenar el modelo: Se ejecuta el script new.py para reentrenar el modelo de aprendizaje automático. Este script procesa el archivo intents2.json y crea un nuevo modelo que incluye las nuevas intenciones y frases.

- **Cargar los datos de intents2.json:**

```
with open("intents2.json", encoding='utf-8') as file:
    data = json.load(file)
```

- **Crear las listas de palabras, etiquetas, documentos y salidas:**

```
words = []
labels = []
docs_x = []
docs_y = []

for intent in data["intents"]:
    for pattern in intent["patterns"]:
        wrds = nltk.word_tokenize(pattern)
        words.extend(wrds)
        docs_x.append(wrds)
        docs_y.append(intent["tag"])

    if intent["tag"] not in labels:
        labels.append(intent["tag"])
```

- **Preprocesar las palabras (stemming):**

```
words = [stemmer.stem(w.lower()) for w in words if w != "?"]
words = sorted(list(set(words)))
```

- **Crear los datos de entrenamiento:**

```
training = []
output = []
out_empty = [0 for _ in range(len(labels))]

for x, doc in enumerate(docs_x):
    bag = []
    wrds = [stemmer.stem(w.lower()) for w in doc]
    for w in words:
        bag.append(1) if w in wrds else bag.append(0)

    output_row = out_empty[:]
    output_row[labels.index(docs_y[x])] = 1

    training.append(bag)
    output.append(output_row)
```

- **Entrenar el modelo de red neuronal:**

```
model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5, verbose=1)
model.save('chatbot_model.h5', hist)
```

- **Guardar el nuevo modelo:** El modelo reentrenado se guarda como chatbot_model.h5, reemplazando al modelo anterior.

Al actualizar el archivo intents2.json y reentrenar el modelo, el chatbot puede aprender nuevas formas de interpretar las entradas del usuario y ofrecer respuestas más precisas y relevantes. Este proceso de aprendizaje continuo permite que el chatbot se adapte a las necesidades cambiantes de los usuarios y mejore su capacidad de comunicación.

Conclusión

El chatbot desarrollado en el repositorio es una herramienta básica pero funcional para la interacción con usuarios en lenguaje natural. Puede ser mejorado en el futuro añadiendo más intents, entrenando el modelo con más datos y utilizando técnicas de procesamiento de lenguaje natural más avanzadas.