



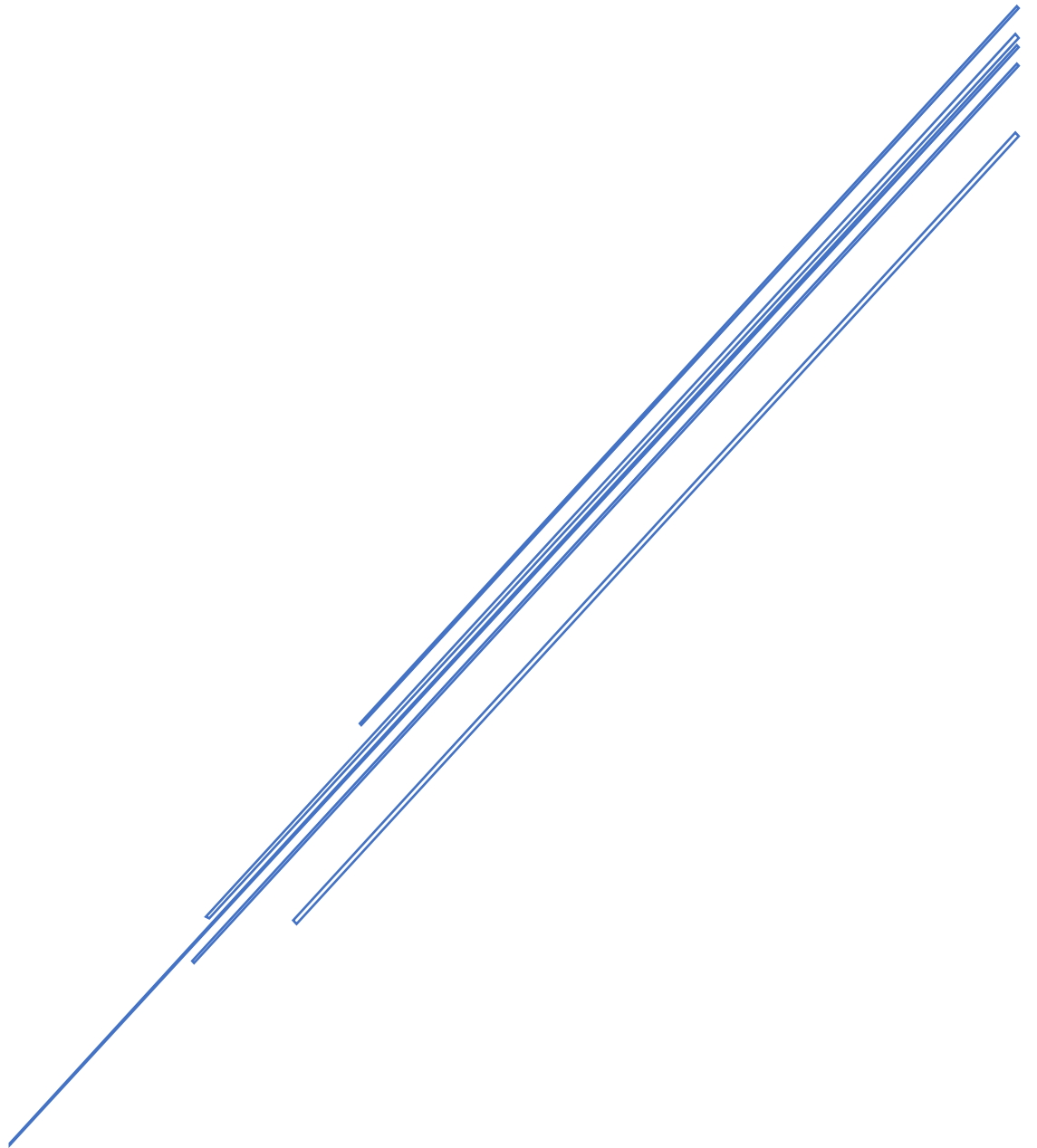
Chatbot en Python

Inteligencia Artificial

Reyes Gómez Norma Leticia

Pérez Ceja Jesús

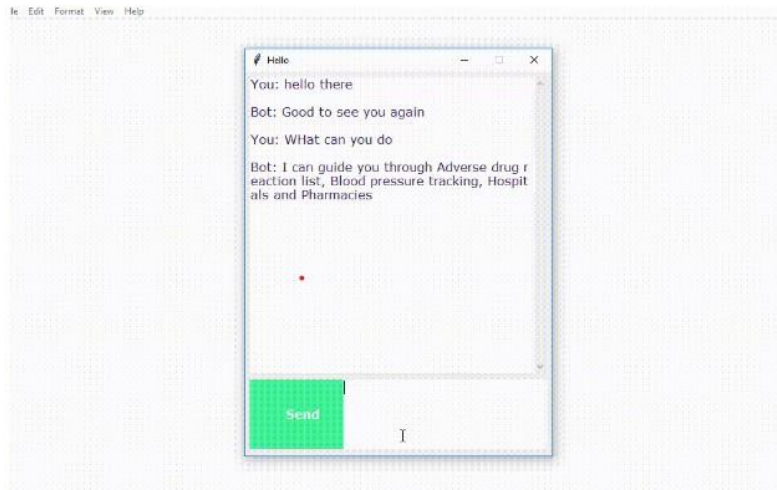
Manjarrez Hernandez Raul



Tecnológico Nacional de México
Instituto tecnológico de Iztapalapa

¿Qué es Chatbot?

Un Chatbot es un software inteligente que es capaz de comunicar y realizar acciones similares a un humano. Los Chatbot se utilizan mucho en la interacción con el cliente, el marketing en los sitios de redes sociales y la mensajería instantánea al cliente. Hay dos tipos básicos de modelos de Chatbot basados en cómo se construyen; Modelos basados en la recuperación y basados en la generación.



Requisitos previos

El proyecto requiere que tenga conocimiento de Python, Keras y procesamiento de lenguaje natural. Junto con ellos, vamos a utilizar algunos módulos de ayuda que se pueden descargar utilizando el comando python-pip.

```
pip install tensorflow
```

```
pip install keras
```

```
pip install pickle
```

```
pip install nltk
```

¿Cómo se realizó el Chatbot en Python?

Ahora vamos a ver la estructura de archivos y el tipo de archivos que vamos a crear:

- Intents.json – El archivo de datos que tiene patrones y respuestas predefinidos.
- train_chatbot.py – En este archivo Python, escribimos un script para construir el modelo y entrenar nuestro Chatbot.
- Words.pkl – Este es un archivo de pickle en el que almacenamos las palabras objeto Python que contiene una lista de nuestro vocabulario.
- Classes.pkl – El archivo de pickle de clases contiene la lista de categorías.
- Chatbot_model.h5 – Este es el modelo entrenado que contiene información sobre el modelo y tiene pesos de las neuronas.
- Chatgui.py – Esta es la secuencia de comandos de Python en la que implementamos la GUI para nuestro Chatbot. Los usuarios pueden interactuar fácilmente con el bot.

1. Importe y cargue el archivo de datos

En primer lugar, haga un nombre de archivo como train_chatbot.py. Importamos los paquetes necesarios para nuestro Chatbot e inicializamos las variables que usaremos en nuestro proyecto Python.

El archivo de datos está en formato JSON, por lo que usamos el paquete JSON para analizar el archivo JSON en Python.

```
1.  importar nltk
2.  de nltk.stem importación WordNetLemmatizer
3.  lemmatizer = WordNetLemmatizer()
4.  importar json
5.  pepinillo de importación
6.
7.  importar numpy como np
8.  de keras.models importar Secuencial
9.  de keras.layers import Dense, Activation, Dropout
10. de keras.optimizers importación SGD
11. importar al azar
12.
13. palabras[]
14. clases = []
15. documentos = []
16. ignore_words = ['?', '!']
17. data_file abierto ('intents.json'). leer()
18. intents = json. cargas(data_file)
```

2. Preprocesar datos

Aquí iteramos a través de los patrones y tokenizamos la oración usando la función nltk.word_tokenize() y anexamos cada palabra en la lista de palabras. También creamos una lista de clases para nuestras etiquetas.

```
1.  intención en intenciones['intenciones']:
2.  para el patrón en intención['patrones']:
3.
4.      #tokenize cada palabra
5.  w a nltk. word_tokenize(patrón)
6.  palabras. extender(w)
7.      #add documentos en el corpus
8.  documentos. anexar((w, intención['etiqueta']))
9.
10.     * Añadir a nuestra lista de clases
11.  si la intención['etiqueta'] no en las clases:
12.  clases. anexar(intención['etiqueta'])
```

Ahora lematizaremos cada palabra y eliminaremos palabras duplicadas de la lista.

```
1.  * lematizar, bajar cada palabra y eliminar duplicados
2.  palabras [ lemmatizer. lematize(w. inferior()) para w en palabras si w
no en ignore_words]
3.  palabras ordenadas (list(set(words)))
4.  * ordenar clases
5.  clases ordenadas(list(set(classes)))
6.  * Documentos: combinación entre patrones e intenciones
7.  imprimir (len(documentos), "documentos")
8.  * Clases - intenciones
9.  imprimir (len(clases), "clases",clases)
10. Palabras, todas las palabras, vocabulario y
11. print (len(palabras), "palabras lematizadas únicas",palabras)
12.
13. pepinillo. volcar(palabras,abierto('words.pkl','wb'))
14. pepinillo. volquete(clases,abierto('classes.pkl','wb'))
```

3. Crear datos de entrenamiento y pruebas

Ahora, vamos a crear los datos de entrenamiento en los que proporcionaremos la entrada y la salida. Nuestra entrada será el patrón y la salida será la clase a la que pertenece nuestro patrón de entrada. Pero el ordenador no entiende el texto, así que convertiremos el texto en números.

```
1. # create our training data
2. training = []
3. # create an empty array for our output
4. output_empty = [0] * len(classes)
5. # training set, bag of words for each sentence
6. for doc in documents:
7.     # initialize our bag of words
8.     bag = []
9.     # list of tokenized words for the pattern
10.    pattern_words = doc[0]
11.    # lemmatize each word - create base word, in attempt to represent
    related words
12.    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in
    pattern_words]
13.    # create our bag of words array with 1, if word match found in
    current pattern
14.    for w in words:
15.        bag.append(1) if w in pattern_words else bag.append(0)
16.
17.    # output is a '0' for each tag and '1' for current tag (for each
    pattern)
18.    output_row = list(output_empty)
19.    output_row[classes.index(doc[1])] = 1
20.
21.    training.append([bag, output_row])
22. # shuffle our features and turn into np.array
23. random.shuffle(training)
24. training = np.array(training)
25. # create train and test lists. X - patterns, Y - intents
26. train_x = list(training[:,0])
27. train_y = list(training[:,1])
28. print("Training data created")
```

4. Construir el modelo

Ahora vamos a construir una red neuronal profunda que tiene 3 capas. Usamos la API secuencial de Keras para esto. Guardemos el modelo como 'chatbot_model.h5'.

```
1. # Create model - 3 layers. First layer 128 neurons, second layer 64
    neurons and 3rd output layer contains number of neurons
2. # equal to number of intents to predict output intent with softmax
3. model = Sequential()
4. model.add(Dense(128, input_shape=(len(train_x[0]),),
    activation='relu'))
5. model.add(Dropout(0.5))
6. model.add(Dense(64, activation='relu'))
7. model.add(Dropout(0.5))
8. model.add(Dense(len(train_y[0]), activation='softmax'))
9.
10. # Compile model. Stochastic gradient descent with Nesterov accelerated
    gradient gives good results for this model
11. sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
12. model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=
    ['accuracy'])
13.
14. #fitting and saving the model
15. hist = model.fit(np.array(train_x), np.array(train_y), epochs=200,
    batch_size=5, verbose=1)
16. model.save('chatbot_model.h5', hist)
17.
18. print("model created")
```

5. Predecir la respuesta (Interfaz gráfica de usuario)

Ahora para predecir las oraciones y obtener una respuesta del usuario para permitirnos crear un nuevo archivo 'chatapp.py'.

Cargaremos el modelo entrenado y luego usaremos una interfaz gráfica de usuario que predecirá la respuesta del bot. El modelo solo nos dirá la clase a la que pertenece, por lo que implementaremos algunas funciones que identificarán la clase y luego recuperaremos una respuesta aleatoria de la lista de respuestas.

Una vez más importamos los paquetes necesarios y cargamos los archivos de pickle 'words.pkl' y 'classes.pkl' que hemos creado cuando entrenamos nuestro modelo:

```
1. import nltk
2. from nltk.stem import WordNetLemmatizer
3. lemmatizer = WordNetLemmatizer()
4. import pickle
5. import numpy as np
6.
7. from keras.models import load_model
8. model = load_model('chatbot_model.h5')
9. import json
10. import random
11. intents = json.loads(open('intents.json').read())
12. words = pickle.load(open('words.pkl', 'rb'))
13. classes = pickle.load(open('classes.pkl', 'rb'))
```

Para predecir la clase, tendremos que proporcionar información de la misma manera que lo hicimos durante el entrenamiento. Por lo tanto, crearemos algunas funciones que realizarán el preprocesamiento de texto y, a continuación, predeciremos la clase.

```
1. def clean_up_sentence(frase):
2.     * tokenizar el patrón - dividir las palabras en matriz
3.     sentence_words = nltk.word_tokenize(sentencia)
4.     * tallo cada palabra - crear forma corta para la palabra
5.     sentence_words = [lemmatizer.lemmatize(palabra.inferior()) para la
6.     palabra en sentence_words]
7.     * bolsa de retorno de palabras array: 0 o 1 para cada palabra en la
8.     bolsa que existe en la oración
9.
10. def bow(frase, palabras, show_details=Verdadero):
11.     * tokenizar el patrón
12.     sentence_words = clean_up_sentence(frase)
13.     * bolsa de palabras - matriz de N palabras, matriz de vocabulario
14.     bolsa = [0]*len(palabras)
15.     para s in sentence_words:
16.         para i,w in enumerate(palabras):
17.             si w == s:
18.                 * asignar 1 si la palabra actual está en la posición
19.                 de vocabulario
20.                 bolsa[i] = 1
21.     si show_details:
22.         impresión ("encontrado en la bolsa: %s" % w)
23.     retorno(np.matriz(bolsa))
24.
25. def predict_class(frase, modelo):
26.     * Filtrar las predicciones por debajo de un umbral
27.     p = arco(oración, palabras, show_details=False)
28.     res = modelo.predict(np.matriz([p]))[0]
29.     ERROR_THRESHOLD = 0.25
30.     resultados = [(i,r) para i,r in enumerate(res) si
31.     r>ERROR_THRESHOLD]
32.     * ordenar por fuerza de probabilidad
```

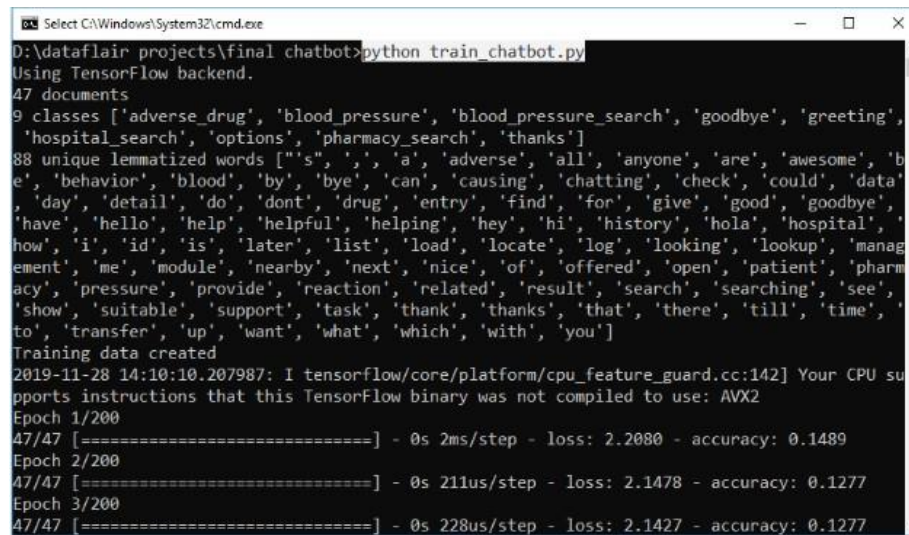
Ahora codificaremos una interfaz gráfica de usuario. Para ello, utilizamos la biblioteca Tkinter que ya viene en Python.

6. Ejecute el chatbot

Para ejecutar el chatbot, tenemos dos archivos principales; train_chatbot.py y chatapp.py.

Primero, entrenamos el modelo usando el comando en el terminal:

Python train_chatbot.py



```
Select C:\Windows\System32\cmd.exe
D:\dataflair projects\final chatbot>python train_chatbot.py
Using TensorFlow backend.
47 documents
9 classes ['adverse_drug', 'blood_pressure', 'blood_pressure_search', 'goodbye', 'greeting',
'hospital_search', 'options', 'pharmacy_search', 'thanks']
88 unique lemmatized words ['s', ',', 'a', 'adverse', 'all', 'anyone', 'are', 'awesome', 'b
e', 'behavior', 'blood', 'by', 'bye', 'can', 'causing', 'chatting', 'check', 'could', 'data',
'day', 'detail', 'do', 'dont', 'drug', 'entry', 'find', 'for', 'give', 'good', 'goodbye', '
have', 'hello', 'help', 'helpful', 'helping', 'hey', 'hi', 'history', 'hola', 'hospital', '
how', 'i', 'id', 'is', 'later', 'list', 'load', 'locate', 'log', 'looking', 'lookup', 'manag
ement', 'me', 'module', 'nearby', 'next', 'nice', 'of', 'offered', 'open', 'patient', 'pharm
acy', 'pressure', 'provide', 'reaction', 'related', 'result', 'search', 'searching', 'see',
'show', 'suitable', 'support', 'task', 'thank', 'thanks', 'that', 'there', 'till', 'time', '
to', 'transfer', 'up', 'want', 'what', 'which', 'with', 'you']
Training data created
2019-11-28 14:10:10.207987: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU su
ports instructions that this TensorFlow binary was not compiled to use: AVX2
Epoch 1/200
47/47 [=====] - 0s 2ms/step - loss: 2.2080 - accuracy: 0.1489
Epoch 2/200
47/47 [=====] - 0s 211us/step - loss: 2.1478 - accuracy: 0.1277
Epoch 3/200
47/47 [=====] - 0s 228us/step - loss: 2.1427 - accuracy: 0.1277
```

A continuación, para ejecutar la aplicación, ejecutamos el segundo archivo.

Python chatgui.py

El programa abre una ventana GUI donde se puede chatear fácilmente con el bot.

