CHATBOT BUILDING FOR A PHARMACY COMPANY FOR CLIENT HANDLING TASK 24*7

Submitted by:Adithya Santhilal

Date:

Library Reference:

- **nltk**: Natural Language Toolkit library for text processing tasks like tokenization and lemmatization.
- **json**: Used to load and work with JSON data (intents in this case).
- **pickle**: Used for serializing and deserializing Python objects (words and classes).
- **numpy**: Provides numerical computing functionality.
- **tensorflow and keras**: Deep learning libraries for building and training the neural network model.
- **tkinter**: Python's built-in GUI library for creating the chatbot interface.

Executive Summary:

This code builds a basic chatbot system capable of understanding user queries related to predefined intents and responding with relevant information. It leverages machine learning techniques for intent classification and text processing methods for preparing training data.

Methodology:

1. **Data Preprocessing:**
    - Loads intents data from a JSON file.
    - Defines a set of custom intents representing user goals (e.g., "Refill_Prescription").
    - Prepares training data by tokenizing user queries (splitting into words) and lemmatizing words (converting them to their base form) for better model generalization.

    - Creates a "bag of words" representation for each query, indicating the presence or absence of each word in the vocabulary.

2. **Model Training:**
    - Builds a multi-layer neural network model using Keras.
    - Trains the model on the prepared training data, where the bag of words representation serves as the input and the corresponding intent label is the target output.
3. **Chatbot Interaction:**
    - Defines functions to process user input, predict the user's intent using the trained model, and retrieve a response based on the predicted intent.
    - Utilizes a graphical user interface (GUI) built with Tkinter to allow users to interact with the chatbot by typing messages and receiving responses.

Overall, this code demonstrates a practical implementation of building a chatbot system using machine learning and natural language processing techniques.

Additional Notes:

- The code utilizes pre-trained libraries like NLTK and Keras, requiring their installation before running.
- The specific details of the intents and responses would need to be customized based on the intended domain of the chatbot.
- The model training process might require adjustments to hyperparameters (e.g., number of epochs) for optimal performance.

Code:

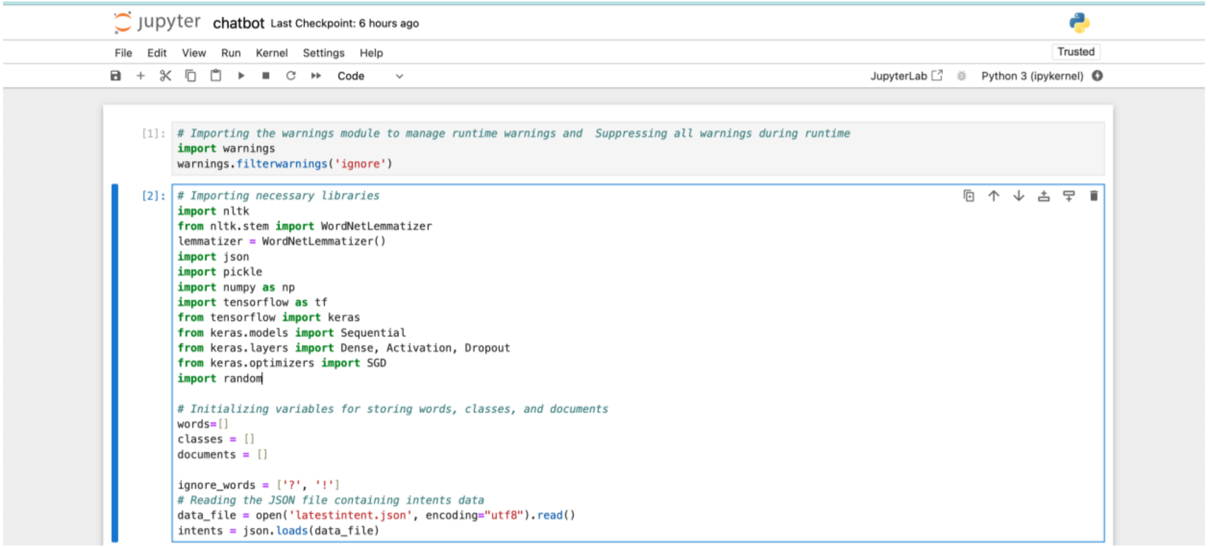
```
# Importing the warnings module to manage runtime warnings and  Suppressing all
warnings during runtime
import warnings
warnings.filterwarnings('ignore')

# Importing necessary libraries
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import json
import pickle
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import SGD
import random

# Initializing variables for storing words, classes, and documents
words=[]
classes = []
documents = []

ignore_words = ['?', '!']
# Reading the JSON file containing intents data
data_file = open('latestintent.json', encoding="utf8").read()
intents = json.loads(data_file)

#display intents
Intents

nltk.download('punkt') # Download the required NLTK data

# Loop through each intent and its patterns
for intent in intents['intents']:
    for pattern in intent['patterns']:
        #tokenize each word in the pattern
        w = nltk.word_tokenize(pattern)
        words.extend(w)  # Extend the words list with tokenized words
       # Append the tokenized pattern and its intent tag to documents
        documents.append((w, intent['tag']))
     # Add the intent tag to the classes list if it's not already there
        if intent['tag'] not in classes:
            classes.append(intent['tag'])

# Download the WordNet corpus for lemmatization
nltk.download('wordnet')

# Lemmatize words, convert to lowercase, and remove words in ignore_words list
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
# Remove duplicates, sort, and convert to a sorted list of unique words (vocabulary)
words = sorted(list(set(words)))
# Sort the classes (intents)
classes = sorted(list(set(classes)))
# Print statistics about the documents, classes, and words
print (len(documents), "documents")
print (len(classes), "classes", classes)
print (len(words), "unique lemmatized words", words)
# Save the words and classes to pickle files for later use
pickle.dump(words,open('words.pkl','wb'))
pickle.dump(classes,open('classes.pkl','wb'))

import numpy as np
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Dropout

# Assume you have defined 'words', 'classes', and 'documents' appropriately

# Create training data
training = []
output_empty = [0] * len(classes)

for doc in documents:
    bag = []
    pattern_words = doc[0]
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]
    # Create bag of words array
    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])

# Shuffle training data
random.shuffle(training)

# Separate input (X) and output (Y)
train_x = np.array([t[0] for t in training])  # Extract bag of words as input
train_y = np.array([t[1] for t in training])  # Extract output row as target

# Pad sequences to ensure uniform length
train_x = pad_sequences(train_x, padding='post', maxlen=88)

# Print shape of training data
print("Training data shape:", train_x.shape)

# Define model architecture
model = Sequential()
```

```
model.add(Dense(128, input_shape=(train_x.shape[1],), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(classes), activation='softmax'))

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Train model
model.fit(train_x, train_y, epochs=200, batch_size=5, verbose=1)

# Save model
model.save('chatbot_model.h5')
print("Model created and saved.")

import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import pickle
import numpy as np
from keras.models import load_model
model = load_model('chatbot_model.h5')   # Load the pre-trained chatbot model
import json
import random
intents = json.loads(open('latestintent.json', encoding="utf8").read())  # Load intents
from JSON file
# Load words and classes from pickle files
words = pickle.load(open('words.pkl','rb'))
classes = pickle.load(open('classes.pkl','rb'))

def clean_up_sentence(sentence):
    # tokenize the pattern - split words into array
    sentence_words = nltk.word_tokenize(sentence)
    # stem each word - create short form for word
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in
sentence_words]
    return sentence_words

# return bag of words array: 0 or 1 for each word in the bag that exists in the
sentence
```

```
def bow(sentence, words, show_details=True):
    # tokenize the pattern
    sentence_words = clean_up_sentence(sentence)
    # bag of words - matrix of N words, vocabulary matrix
    bag = [0]*len(words)
    for s in sentence_words:
        for i,w in enumerate(words):
            if w == s:
                # assign 1 if current word is in the vocabulary position
                bag[i] = 1
                if show_details:
                    print ("found in bag: %s" % w)
    return(np.array(bag))

def predict_class(sentence, model):
    # filter out predictions below a threshold
    p = bow(sentence, words,show_details=False)
    res = model.predict(np.array([p]))[0]
    ERROR_THRESHOLD = 0.25
    results = [[i,r] for i,r in enumerate(res) if r>ERROR_THRESHOLD]
    # sort by strength of probability
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
    return return_list

def getResponse(ints, intents_json):
    tag = ints[0]['intent']   # Extract the predicted intent tag
    list_of_intents = intents_json['intents']   # Extract all intents from the JSON object
    for i in list_of_intents:
        if(i['tag']== tag):    # Find the matching intent tag
            result = random.choice(i['responses'])      # Randomly select a response from
the matched intent
            break
    return result

def chatbot_response(msg):
    ints = predict_class(msg, model)   # Predict the intent of the input message using
a classifier
    res = getResponse(ints, intents)    # Retrieve a response based on the predicted
intent
    return res     # Return the generated response
```

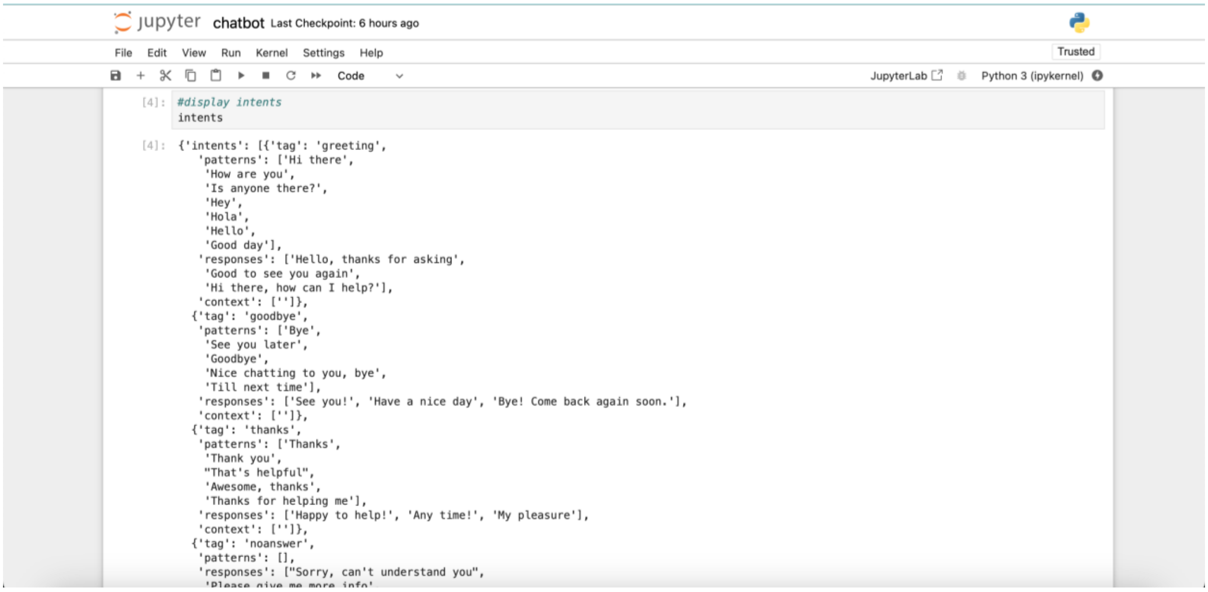
```
import tkinter
from tkinter import *

# Function to send the message
def send():
    # Get the message from the EntryBox, strip any extra whitespace, and clear the EntryBox
    msg = EntryBox.get("1.0",'end-1c').strip()
    EntryBox.delete("0.0",END)

     # If the message is not empty
    if msg != '':
         # Enable ChatLog for editing
        ChatLog.config(state=NORMAL)
         # Insert user's message into ChatLog
        ChatLog.insert(END, "You: " + msg + '\n\n')
         # Configure text properties for user's message
        ChatLog.config(foreground="#442265", font=("Verdana", 12 ))

        # Get response from the chatbot based on user's message
        res = chatbot_response(msg)
        # Scroll ChatLog to the bottom to show the latest messages
        ChatLog.insert(END, "Bot: " + res + '\n\n')

        ChatLog.config(state=DISABLED)
        ChatLog.yview(END)


# Create a new tkinter window
base = Tk()
base.title("Hello")
base.geometry("400x500")
base.resizable(width=FALSE, height=FALSE)

# Create ChatLog widget to display messages
ChatLog = Text(base, bd=0, bg="white", height="8", width="50", font="Arial",)

ChatLog.config(state=DISABLED)

#Bind scrollbar to Chat window
scrollbar = Scrollbar(base, command=ChatLog.yview, cursor="heart")
ChatLog['yscrollcommand'] = scrollbar.set
```

```
#Create Button to send message
SendButton = Button(base, font=("Verdana",12,'bold'), text="Send", width="12",
height=5,
                    bd=0, bg="#32de97", activebackground="#3c9d9b",fg='#ffffff',
                    command= send )

#Create the box to enter message
EntryBox = Text(base, bd=0, bg="white",width="29", height="5", font="Arial")
#EntryBox.bind("<Return>", send)


#Place all components on the screen
scrollbar.place(x=376,y=6, height=386)
ChatLog.place(x=6,y=6, height=386, width=370)
EntryBox.place(x=128, y=401, height=90, width=265)
SendButton.place(x=6, y=401, height=90)

base.mainloop()
```


jupyter chatbot Last Checkpoint: 6 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

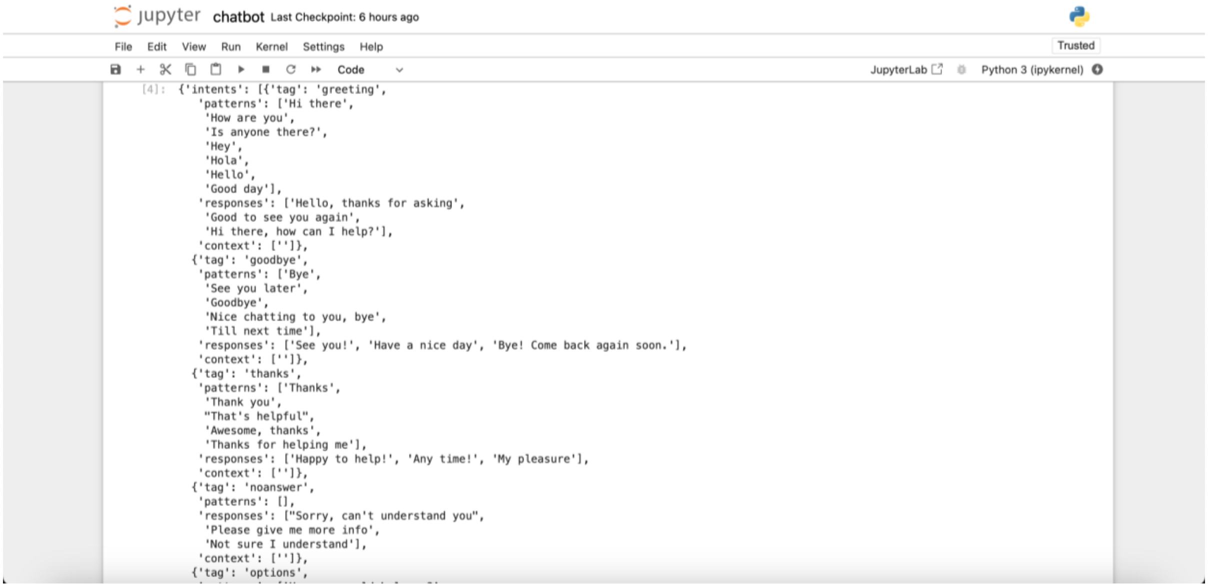
[1]:
```
# Importing the warnings module to manage runtime warnings and  Suppressing all warnings during runtime
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
# Importing necessary libraries
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import json
import pickle
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import SGD
import random

# Initializing variables for storing words, classes, and documents
words=[]
classes = []
documents = []

ignore_words = ['?', '!']
# Reading the JSON file containing intents data
data_file = open('latestintent.json', encoding="utf8").read()
intents = json.loads(data_file)
```


jupyter chatbot Last Checkpoint: 6 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

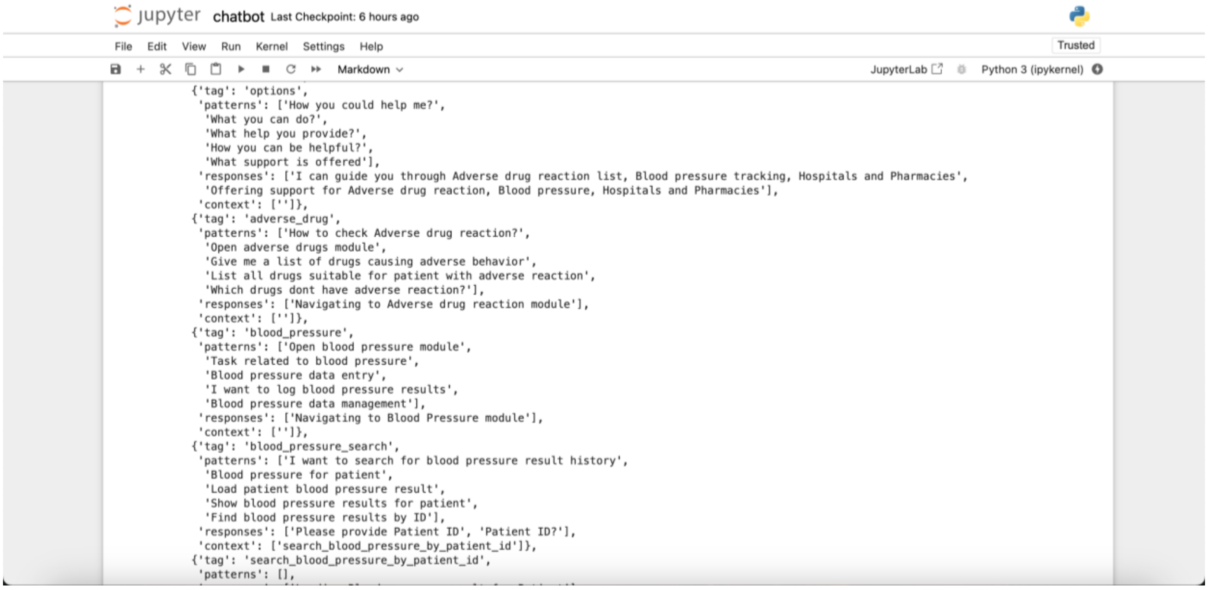
[4]:
```
#display intents
intents
```

[4]:
```
{'intents': [{'tag': 'greeting',
   'patterns': ['Hi there',
    'How are you',
    'Is anyone there?',
    'Hey',
    'Hola',
    'Hello',
    'Good day'],
   'responses': ['Hello, thanks for asking',
    'Good to see you again',
    'Hi there, how can I help?'],
   'context': ['']},
  {'tag': 'goodbye',
   'patterns': ['Bye',
    'See you later',
    'Goodbye',
    'Nice chatting to you, bye',
    'Till next time'],
   'responses': ['See you!', 'Have a nice day', 'Bye! Come back again soon.'],
   'context': ['']},
  {'tag': 'thanks',
   'patterns': ['Thanks',
    'Thank you',
    "That's helpful",
    'Awesome, thanks',
    'Thanks for helping me'],
   'responses': ['Happy to help!', 'Any time!', 'My pleasure'],
   'context': ['']},
  {'tag': 'noanswer',
   'patterns': [],
   'responses': ["Sorry, can't understand you",
    'Please give me more info',
```

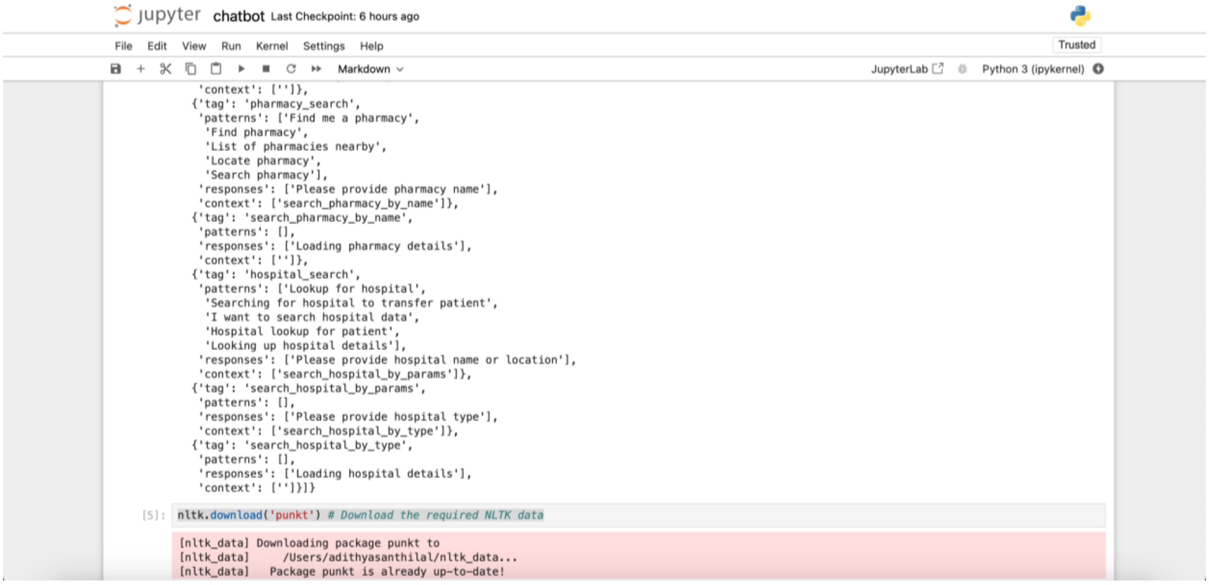

jupyter chatbot Last Checkpoint: 6 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

[4]:
```
{'intents': [{'tag': 'greeting',
   'patterns': ['Hi there',
    'How are you',
    'Is anyone there?',
    'Hey',
    'Hola',
    'Hello',
    'Good day'],
   'responses': ['Hello, thanks for asking',
    'Good to see you again',
    'Hi there, how can I help?'],
   'context': ['']},
  {'tag': 'goodbye',
   'patterns': ['Bye',
    'See you later',
    'Goodbye',
    'Nice chatting to you, bye',
    'Till next time'],
   'responses': ['See you!', 'Have a nice day', 'Bye! Come back again soon.'],
   'context': ['']},
  {'tag': 'thanks',
   'patterns': ['Thanks',
    'Thank you',
    "That's helpful",
    'Awesome, thanks',
    'Thanks for helping me'],
   'responses': ['Happy to help!', 'Any time!', 'My pleasure'],
   'context': ['']},
  {'tag': 'noanswer',
   'patterns': [],
   'responses': ["Sorry, can't understand you",
    'Please give me more info',
```

```
    'Not sure I understand'],
   'context': ['']},
  {'tag': 'options',
```


jupyter chatbot Last Checkpoint: 6 hours ago
File Edit View Run Kernel Settings Help
Trusted
Markdown
JupyterLab
Python 3 (ipykernel)

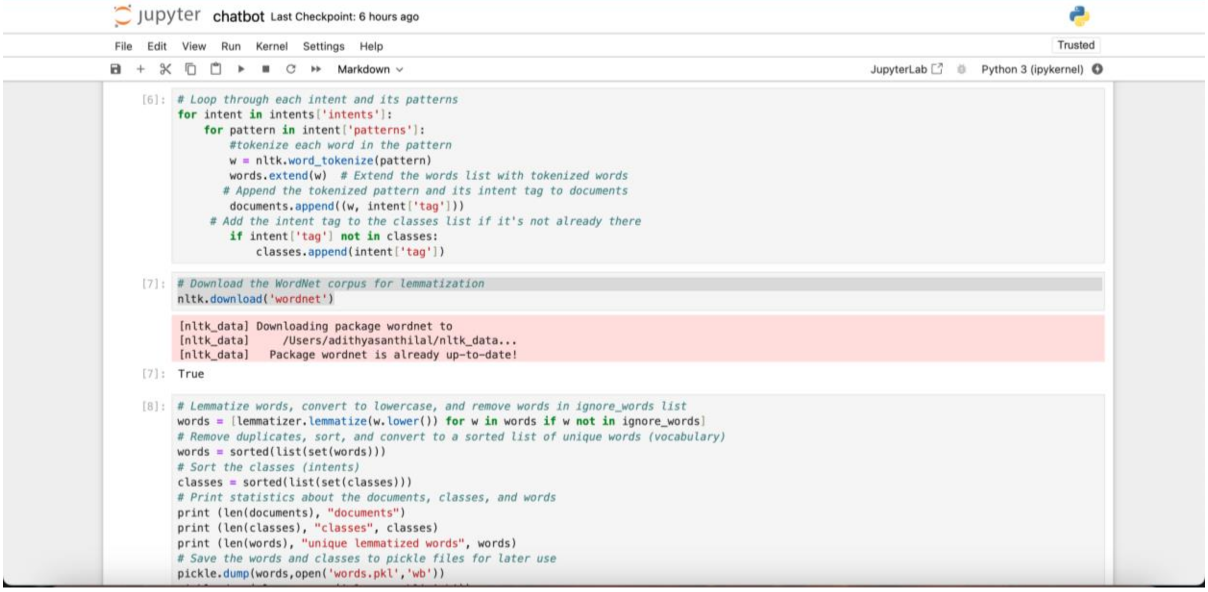
```
        {'tag': 'options',
         'patterns': ['How you could help me?',
          'What you can do?',
          'What help you provide?',
          'How you can be helpful?',
          'What support is offered'],
         'responses': ['I can guide you through Adverse drug reaction list, Blood pressure tracking, Hospitals and Pharmacies',
          'Offering support for Adverse drug reaction, Blood pressure, Hospitals and Pharmacies'],
         'context': ['']},
        {'tag': 'adverse_drug',
         'patterns': ['How to check Adverse drug reaction?',
          'Open adverse drugs module',
          'Give me a list of drugs causing adverse behavior',
          'List all drugs suitable for patient with adverse reaction',
          'Which drugs dont have adverse reaction?'],
         'responses': ['Navigating to Adverse drug reaction module'],
         'context': ['']},
        {'tag': 'blood_pressure',
         'patterns': ['Open blood pressure module',
          'Task related to blood pressure',
          'Blood pressure data entry',
          'I want to log blood pressure results',
          'Blood pressure data management'],
         'responses': ['Navigating to Blood Pressure module'],
         'context': ['']},
        {'tag': 'blood_pressure_search',
         'patterns': ['I want to search for blood pressure result history',
          'Blood pressure for patient',
          'Load patient blood pressure result',
          'Show blood pressure results for patient',
          'Find blood pressure results by ID'],
         'responses': ['Please provide Patient ID', 'Patient ID?'],
         'context': ['search_blood_pressure_by_patient_id']},
        {'tag': 'search_blood_pressure_by_patient_id',
         'patterns': [],
```


jupyter chatbot Last Checkpoint: 6 hours ago
File Edit View Run Kernel Settings Help
Trusted
Markdown
JupyterLab
Python 3 (ipykernel)

```
         'context': ['']},
        {'tag': 'pharmacy_search',
         'patterns': ['Find me a pharmacy',
          'Find pharmacy',
          'List of pharmacies nearby',
          'Locate pharmacy',
          'Search pharmacy'],
         'responses': ['Please provide pharmacy name'],
         'context': ['search_pharmacy_by_name']},
        {'tag': 'search_pharmacy_by_name',
         'patterns': [],
         'responses': ['Loading pharmacy details'],
         'context': ['']},
        {'tag': 'hospital_search',
         'patterns': ['Lookup for hospital',
          'Searching for hospital to transfer patient',
          'I want to search hospital data',
          'Hospital lookup for patient',
          'Looking up hospital details'],
         'responses': ['Please provide hospital name or location'],
         'context': ['search_hospital_by_params']},
        {'tag': 'search_hospital_by_params',
         'patterns': [],
         'responses': ['Please provide hospital type'],
         'context': ['search_hospital_by_type']},
        {'tag': 'search_hospital_by_type',
         'patterns': [],
         'responses': ['Loading hospital details'],
         'context': ['']}]}
```

```
[5]: nltk.download('punkt') # Download the required NLTK data
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/adithyasanthilal/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```


jupyter chatbot Last Checkpoint: 6 hours ago
File Edit View Run Kernel Settings Help
Trusted
Markdown
JupyterLab
Python 3 (ipykernel)

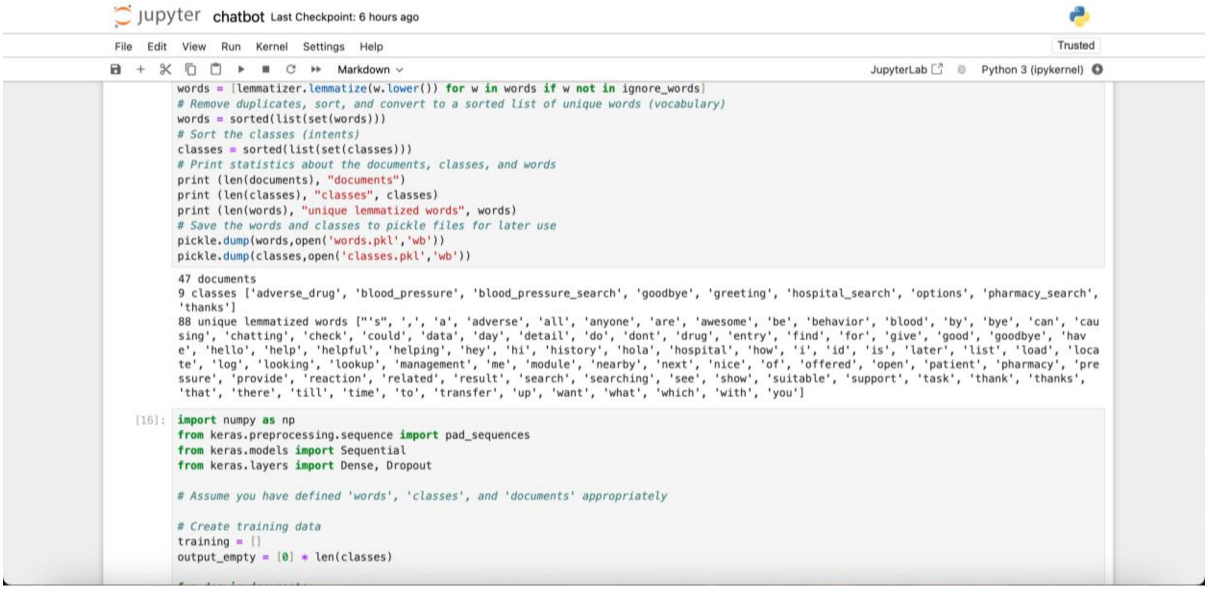
```
[6]: # Loop through each intent and its patterns
     for intent in intents['intents']:
         for pattern in intent['patterns']:
             #tokenize each word in the pattern
             w = nltk.word_tokenize(pattern)
             words.extend(w)  # Extend the words list with tokenized words
            # Append the tokenized pattern and its intent tag to documents
             documents.append((w, intent['tag']))
           # Add the intent tag to the classes list if it's not already there
             if intent['tag'] not in classes:
                 classes.append(intent['tag'])
```

```
[7]: # Download the WordNet corpus for lemmatization
     nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/adithyasanthilal/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
[7]: True
```

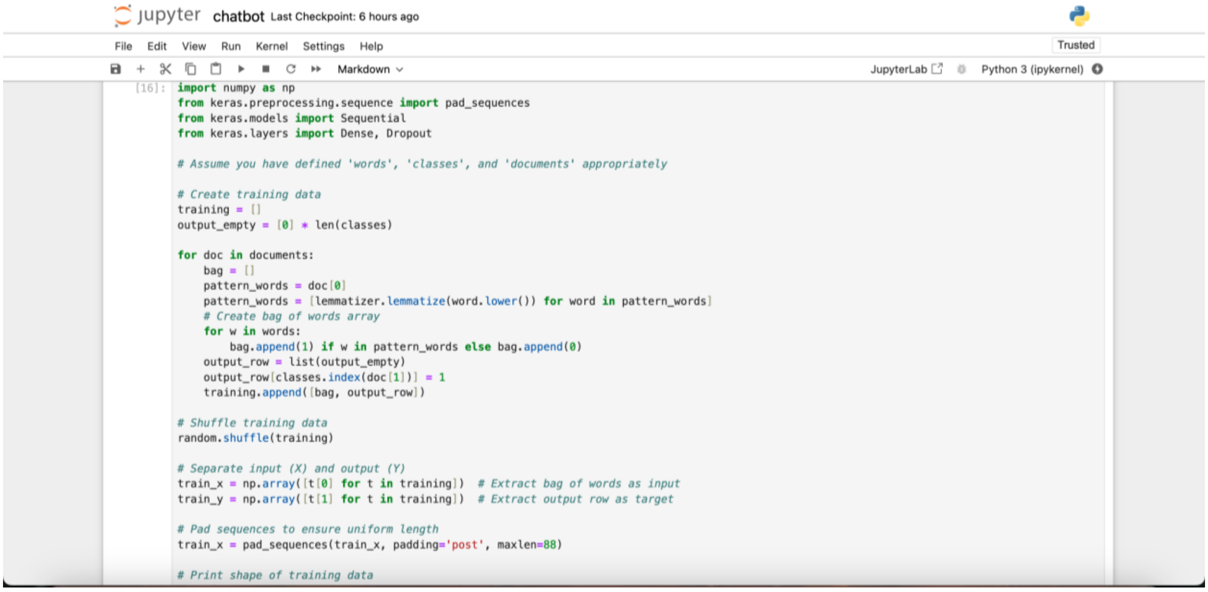
```
[8]: # Lemmatize words, convert to lowercase, and remove words in ignore_words list
     words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
     # Remove duplicates, sort, and convert to a sorted list of unique words (vocabulary)
     words = sorted(list(set(words)))
     # Sort the classes (intents)
     classes = sorted(list(set(classes)))
     # Print statistics about the documents, classes, and words
     print (len(documents), "documents")
     print (len(classes), "classes", classes)
     print (len(words), "unique lemmatized words", words)
     # Save the words and classes to pickle files for later use
     pickle.dump(words,open('words.pkl','wb'))
```


jupyter chatbot Last Checkpoint: 6 hours ago
File Edit View Run Kernel Settings Help
Trusted
Markdown
JupyterLab
Python 3 (ipykernel)

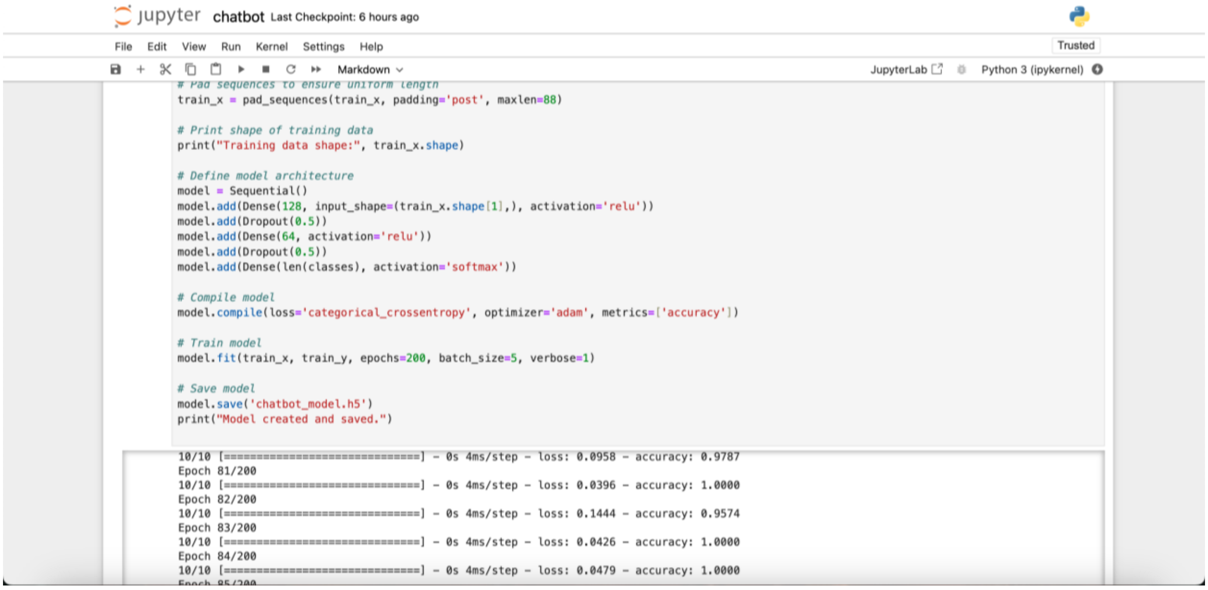
```
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
# Remove duplicates, sort, and convert to a sorted list of unique words (vocabulary)
words = sorted(list(set(words)))
# Sort the classes (intents)
classes = sorted(list(set(classes)))
# Print statistics about the documents, classes, and words
print (len(documents), "documents")
print (len(classes), "classes", classes)
print (len(words), "unique lemmatized words", words)
# Save the words and classes to pickle files for later use
pickle.dump(words,open('words.pkl','wb'))
pickle.dump(classes,open('classes.pkl','wb'))
```

```
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', 'be', 'behavior', 'blood', 'by', 'bye', 'can', 'cau
sing', 'chatting', 'check', 'could', 'data', 'day', 'detail', 'do', 'dont', 'drug', 'entry', 'find', 'for', 'give', 'good', 'goodbye', 'hav
e', 'hello', 'help', 'helpful', 'helping', 'hey', 'hi', 'history', 'hola', 'hospital', 'how', 'i', 'id', 'is', 'later', 'list', 'load', 'loca
te', 'log', 'looking', 'lookup', 'management', 'me', 'module', 'nearby', 'next', 'nice', 'of', 'offered', 'open', 'patient', 'pharmacy', 'pre
ssure', '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']
```

```
[16]: import numpy as np
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Dropout

# Assume you have defined 'words', 'classes', and 'documents' appropriately

# Create training data
training = []
output_empty = [0] * len(classes)
```

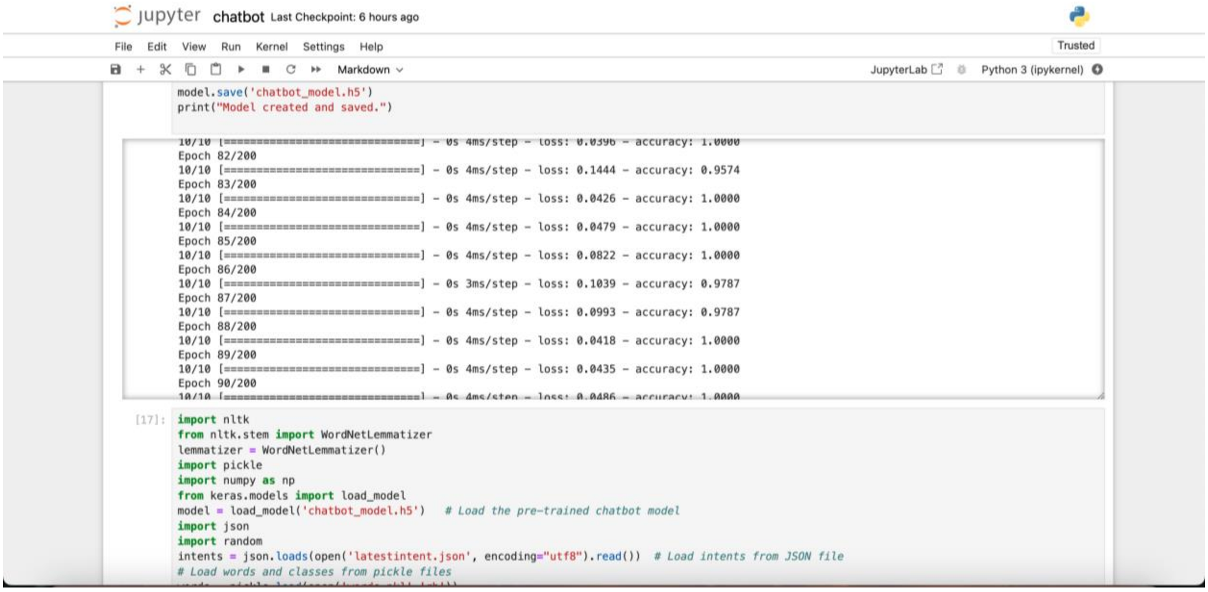

jupyter chatbot Last Checkpoint: 6 hours ago
File Edit View Run Kernel Settings Help
Trusted
Markdown
JupyterLab
Python 3 (ipykernel)

```
[16]: import numpy as np
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Dropout

# Assume you have defined 'words', 'classes', and 'documents' appropriately

# Create training data
training = []
output_empty = [0] * len(classes)

for doc in documents:
    bag = []
    pattern_words = doc[0]
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]
    # Create bag of words array
    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])

# Shuffle training data
random.shuffle(training)

# Separate input (X) and output (Y)
train_x = np.array([t[0] for t in training])  # Extract bag of words as input
train_y = np.array([t[1] for t in training])  # Extract output row as target

# Pad sequences to ensure uniform length
train_x = pad_sequences(train_x, padding='post', maxlen=88)

# Print shape of training data
```


jupyter chatbot Last Checkpoint: 6 hours ago
File Edit View Run Kernel Settings Help
Trusted
Markdown
JupyterLab
Python 3 (ipykernel)

```
# Pad sequences to ensure uniform length
train_x = pad_sequences(train_x, padding='post', maxlen=88)

# Print shape of training data
print("Training data shape:", train_x.shape)

# Define model architecture
model = Sequential()
model.add(Dense(128, input_shape=(train_x.shape[1],), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(classes), activation='softmax'))

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train model
model.fit(train_x, train_y, epochs=200, batch_size=5, verbose=1)

# Save model
model.save('chatbot_model.h5')
print("Model created and saved.")
```

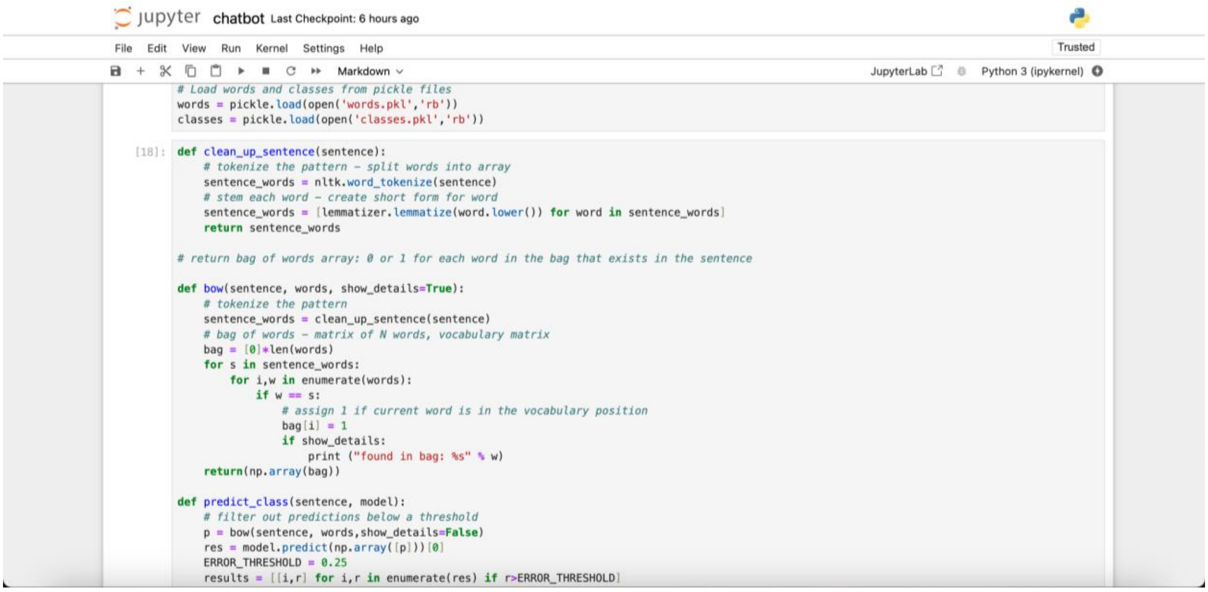
```
10/10 [==============================] - 0s 4ms/step - loss: 0.0958 - accuracy: 0.9787
Epoch 81/200
10/10 [==============================] - 0s 4ms/step - loss: 0.0396 - accuracy: 1.0000
Epoch 82/200
10/10 [==============================] - 0s 4ms/step - loss: 0.1444 - accuracy: 0.9574
Epoch 83/200
10/10 [==============================] - 0s 4ms/step - loss: 0.0426 - accuracy: 1.0000
Epoch 84/200
10/10 [==============================] - 0s 4ms/step - loss: 0.0479 - accuracy: 1.0000
Epoch 85/200
```


jupyter chatbot Last Checkpoint: 6 hours ago
File Edit View Run Kernel Settings Help
Trusted
Markdown
JupyterLab
Python 3 (ipykernel)

```
model.save('chatbot_model.h5')
print("Model created and saved.")
```

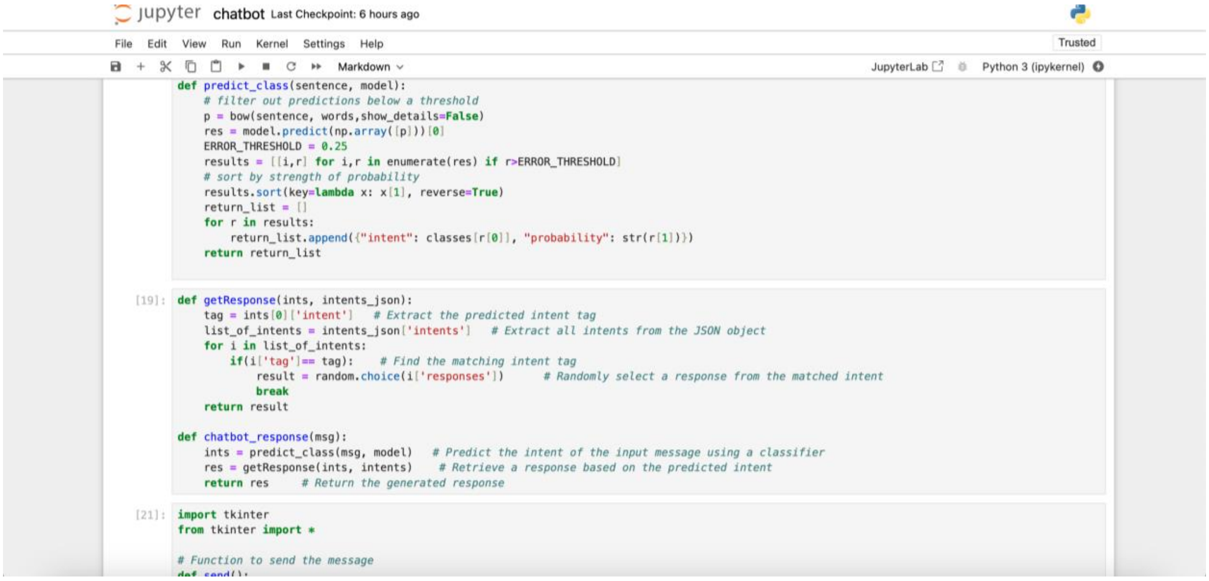
```
10/10 [==============================] - 0s 4ms/step - loss: 0.0396 - accuracy: 1.0000
Epoch 82/200
10/10 [==============================] - 0s 4ms/step - loss: 0.1444 - accuracy: 0.9574
Epoch 83/200
10/10 [==============================] - 0s 4ms/step - loss: 0.0426 - accuracy: 1.0000
Epoch 84/200
10/10 [==============================] - 0s 4ms/step - loss: 0.0479 - accuracy: 1.0000
Epoch 85/200
10/10 [==============================] - 0s 4ms/step - loss: 0.0822 - accuracy: 1.0000
Epoch 86/200
10/10 [==============================] - 0s 3ms/step - loss: 0.1039 - accuracy: 0.9787
Epoch 87/200
10/10 [==============================] - 0s 4ms/step - loss: 0.0993 - accuracy: 0.9787
Epoch 88/200
10/10 [==============================] - 0s 4ms/step - loss: 0.0418 - accuracy: 1.0000
Epoch 89/200
10/10 [==============================] - 0s 4ms/step - loss: 0.0435 - accuracy: 1.0000
Epoch 90/200
10/10 [==============================] - 0s 4ms/step - loss: 0.0486 - accuracy: 1.0000
```

```
[17]: import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import pickle
import numpy as np
from keras.models import load_model
model = load_model('chatbot_model.h5')   # Load the pre-trained chatbot model
import json
import random
intents = json.loads(open('latestintent.json', encoding="utf8").read())  # Load intents from JSON file
# Load words and classes from pickle files
```


jupyter chatbot Last Checkpoint: 6 hours ago
File Edit View Run Kernel Settings Help
Trusted
Markdown
JupyterLab
Python 3 (ipykernel)

```
# Load words and classes from pickle files
words = pickle.load(open('words.pkl','rb'))
classes = pickle.load(open('classes.pkl','rb'))
```

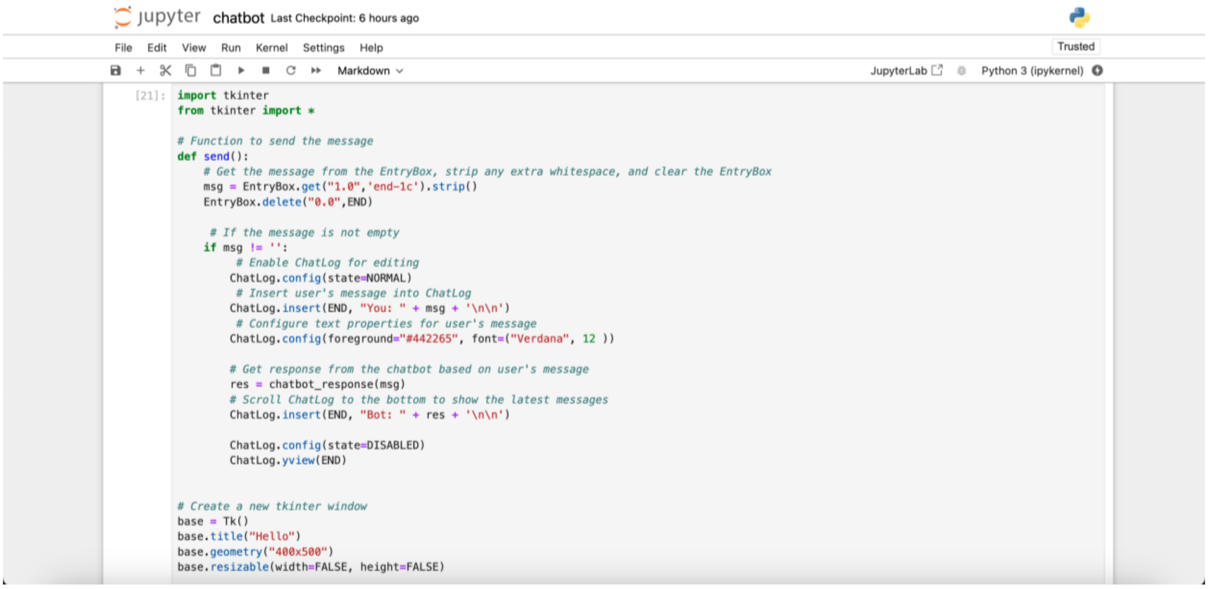
```
[18]: def clean_up_sentence(sentence):
    # tokenize the pattern - split words into array
    sentence_words = nltk.word_tokenize(sentence)
    # stem each word - create short form for word
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words]
    return sentence_words

# return bag of words array: 0 or 1 for each word in the bag that exists in the sentence

def bow(sentence, words, show_details=True):
    # tokenize the pattern
    sentence_words = clean_up_sentence(sentence)
    # bag of words - matrix of N words, vocabulary matrix
    bag = [0]*len(words)
    for s in sentence_words:
        for i,w in enumerate(words):
            if w == s:
                # assign 1 if current word is in the vocabulary position
                bag[i] = 1
                if show_details:
                    print ("found in bag: %s" % w)
    return(np.array(bag))

def predict_class(sentence, model):
    # filter out predictions below a threshold
    p = bow(sentence, words,show_details=False)
    res = model.predict(np.array([p]))[0]
    ERROR_THRESHOLD = 0.25
    results = [[i,r] for i,r in enumerate(res) if r>ERROR_THRESHOLD]
```


jupyter chatbot Last Checkpoint: 6 hours ago
File Edit View Run Kernel Settings Help
Trusted
Markdown
JupyterLab
Python 3 (ipykernel)

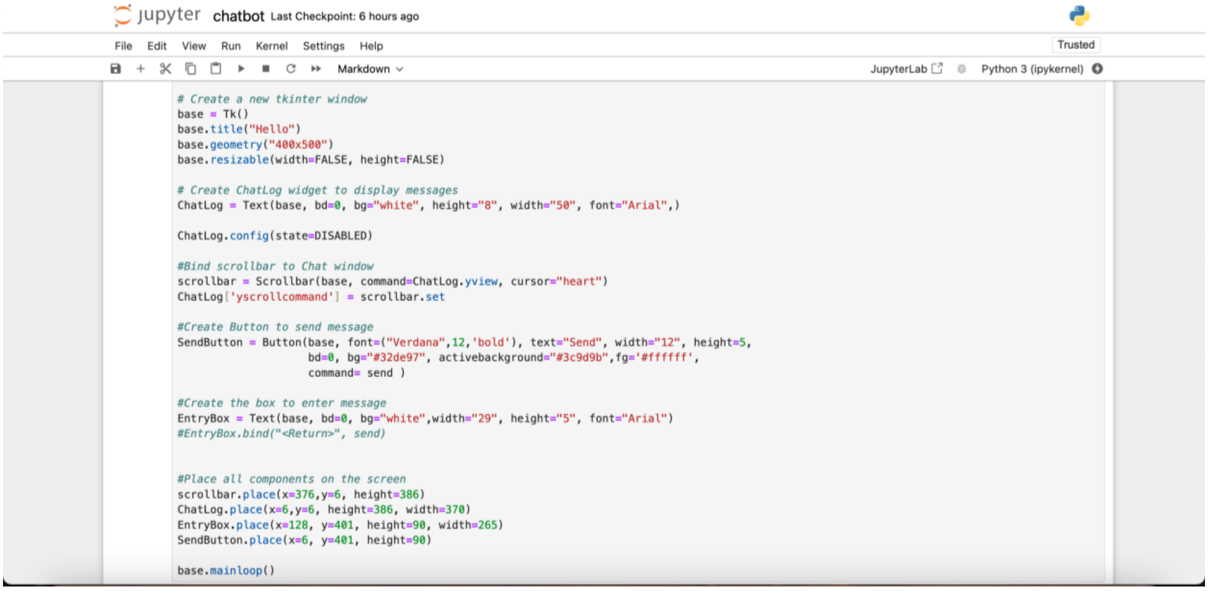
```
def predict_class(sentence, model):
    # filter out predictions below a threshold
    p = bow(sentence, words,show_details=False)
    res = model.predict(np.array([p]))[0]
    ERROR_THRESHOLD = 0.25
    results = [[i,r] for i,r in enumerate(res) if r>ERROR_THRESHOLD]
    # sort by strength of probability
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
    return return_list
```

```
[19]: def getResponse(ints, intents_json):
    tag = ints[0]['intent']   # Extract the predicted intent tag
    list_of_intents = intents_json['intents']   # Extract all intents from the JSON object
    for i in list_of_intents:
        if(i['tag']== tag):    # Find the matching intent tag
            result = random.choice(i['responses'])      # Randomly select a response from the matched intent
            break
    return result

def chatbot_response(msg):
    ints = predict_class(msg, model)   # Predict the intent of the input message using a classifier
    res = getResponse(ints, intents)    # Retrieve a response based on the predicted intent
    return res     # Return the generated response
```

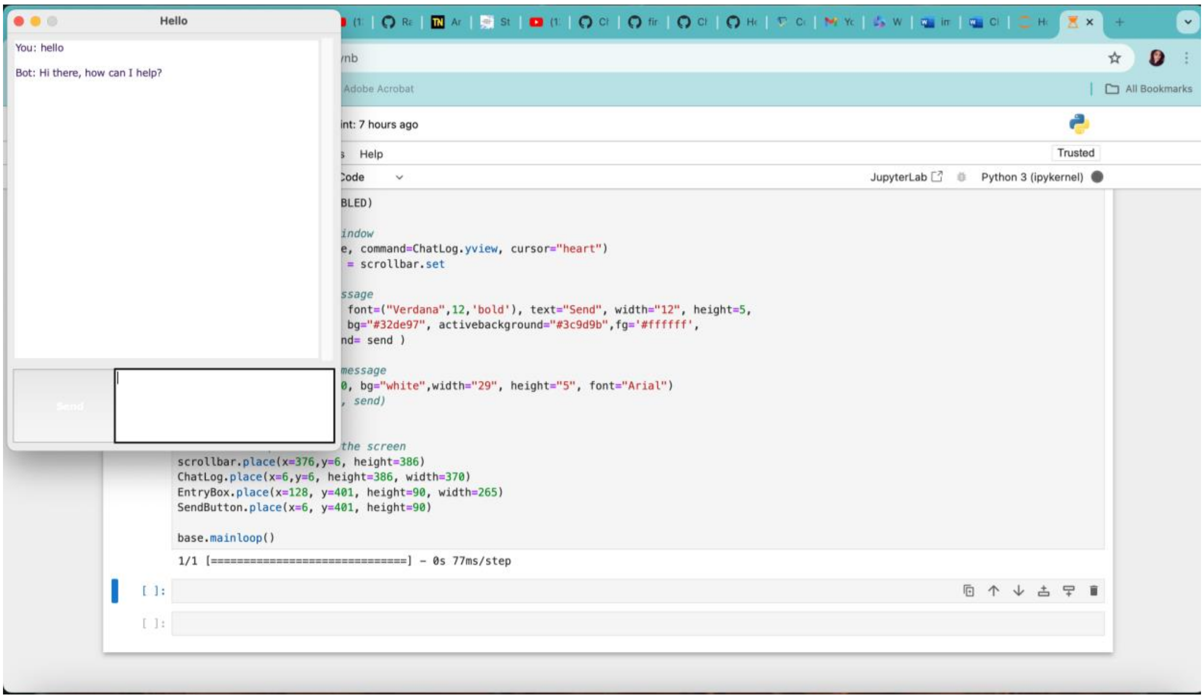
```
[21]: import tkinter
from tkinter import *

# Function to send the message
def send():
```



```python
import tkinter
from tkinter import *

# Function to send the message
def send():
    # Get the message from the EntryBox, strip any extra whitespace, and clear the EntryBox
    msg = EntryBox.get("1.0",'end-1c').strip()
    EntryBox.delete("0.0",END)

     # If the message is not empty
    if msg != '':
         # Enable ChatLog for editing
        ChatLog.config(state=NORMAL)
         # Insert user's message into ChatLog
        ChatLog.insert(END, "You: " + msg + '\n\n')
         # Configure text properties for user's message
        ChatLog.config(foreground="#442265", font=("Verdana", 12 ))

        # Get response from the chatbot based on user's message
        res = chatbot_response(msg)
        # Scroll ChatLog to the bottom to show the latest messages
        ChatLog.insert(END, "Bot: " + res + '\n\n')

        ChatLog.config(state=DISABLED)
        ChatLog.yview(END)


# Create a new tkinter window
base = Tk()
base.title("Hello")
base.geometry("400x500")
base.resizable(width=FALSE, height=FALSE)
```



```python
# Create a new tkinter window
base = Tk()
base.title("Hello")
base.geometry("400x500")
base.resizable(width=FALSE, height=FALSE)

# Create ChatLog widget to display messages
ChatLog = Text(base, bd=0, bg="white", height="8", width="50", font="Arial",)

ChatLog.config(state=DISABLED)

#Bind scrollbar to Chat window
scrollbar = Scrollbar(base, command=ChatLog.yview, cursor="heart")
ChatLog['yscrollcommand'] = scrollbar.set

#Create Button to send message
SendButton = Button(base, font=("Verdana",12,'bold'), text="Send", width="12", height=5,
                    bd=0, bg="#32de97", activebackground="#3c9d9b",fg='#ffffff',
                    command= send )

#Create the box to enter message
EntryBox = Text(base, bd=0, bg="white",width="29", height="5", font="Arial")
#EntryBox.bind("<Return>", send)


#Place all components on the screen
scrollbar.place(x=376,y=6, height=386)
ChatLog.place(x=6,y=6, height=386, width=370)
EntryBox.place(x=128, y=401, height=90, width=265)
SendButton.place(x=6, y=401, height=90)

base.mainloop()
```



You: hello

Bot: Hi there, how can I help?

```
BLED)

indow
e, command=ChatLog.yview, cursor="heart")
 = scrollbar.set

ssage
 font=("Verdana",12,'bold'), text="Send", width="12", height=5,
 bg="#32de97", activebackground="#3c9d9b",fg='#ffffff',
nd= send )

message
0, bg="white",width="29", height="5", font="Arial")
, send)


the screen
scrollbar.place(x=376,y=6, height=386)
ChatLog.place(x=6,y=6, height=386, width=370)
EntryBox.place(x=128, y=401, height=90, width=265)
SendButton.place(x=6, y=401, height=90)

base.mainloop()
```

```
1/1 [==============================] - 0s 77ms/step
```

```
[ ]:
[ ]:
```