Chat file

```
lemmatizer = WordNetLemmatizer()
intents = json.loads(open('intents2.json').read())



# paterns
words = pickle.load(open('words.pkl', 'rb'))

# tags
classes = pickle.load(open('classes.pkl', 'rb'))

model = load_model('chatbot_model.h5')

"""
The code initializes the WordNetLemmatizer from NLTK, loads the intents data from a JSON file,
and loads the preprocessed data from pickle files.
It loads the words and classes used for training the model and loads the pre-trained model.
"""
```

intents:{

'intents': [{'tag': 'greeting',

'patterns': ['Hi', 'How are you', 'Is anyone there?', 'Hello', 'Good day', 'Whats up', 'Good morning', 'Good evening', 'hello', 'hey', "what's up"],

'responses': ['Hello! How can I assist you today?', 'Good to see you! How may I help you?', 'Hi there, how can I assist you?']

, 'context_set': ''}

, {'tag': 'goodbye', 'patterns': ['cya', 'See you later', 'Goodbye', 'I am Leaving', 'Have a Good day', 'bye', 'i have to go', 'gotta go'], 'responses': ['Sad to see you go. Have a great day!', 'Talk to you later. Take care!', 'Goodbye! Have a wonderful day!'], 'context_set': ''}, {'tag': 'age', 'patterns': ['how old', 'how old are you', 'what is your age', 'how old are you', 'age?'], 'responses': ["I am a virtual assistant, so I don't have an age!", "I'm an AI-powered bot, so age doesn't apply to me!"], 'context_set': ''}, {'tag': 'name', 'patterns': ['what is your name', 'what should I call you', 'whats your name?', 'who are you?'],

'responses': ['You can call me CallBot.', "I'm CallBot!", "I'm your friendly CallBot."], 'context_set': ''}, {'tag': 'help', 'patterns': ['Id like to ask something', 'what can you do', 'can you help me?', 'can i tell you something'], 'responses': ["I'm here to help you! How can I assist you today?", 'I can help you with a wide range of inquiries. What do you need assistance with?', "I'm here to assist you. Please let me know how I can help."], 'context_set': ''}, {'tag': 'customer_service', 'patterns': ['I have a question about my order', 'I need assistance with a product', 'Can you help me with a billing issue?', 'I want to provide feedback', 'I need technical support', 'Can you transfer me to a live agent?'], 'responses': ['Sure, I can assist you with that. Please provide me with more details about your question or issue.', "Of course! I'm here to help. Please let me know the specific problem or question you have.", "I'll do my best to assist you. Please provide me with more information about your request."], 'context_set': ''

}]}

type: <class 'dict'>

==========

words picke :["'s", 'Can', 'Good', 'Goodbye', 'Have', 'Hello', 'Hi', 'How', 'I', 'Id', 'Is', 'Leaving', 'See', 'Whats', 'a', 'about', 'age', 'agent', 'am', 'anyone', 'are', 'ask', 'assistance', 'billing', 'bye', 'call', 'can', 'cya', 'day', 'do', 'evening', 'feedback', 'go', 'got', 'have', 'hello', 'help', 'hey', 'how', 'i', 'is', 'issue', 'later', 'like', 'live', 'me', 'morning', 'my', 'name', 'need', 'old', 'order', 'product', 'provide', 'question', 'should', 'something', 'support', 'ta', 'technical', 'tell', 'there', 'to', 'transfer', 'up', 'want', 'what', 'whats', 'who', 'with', 'you', 'your']

type:<class 'list'>

==========

classes picke :['age', 'customer_service', 'goodbye', 'greeting', 'help', 'name']

type:<class 'list'>

==========

2023-10-30 10:38:25.206675: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.

To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.

Help on Sequential in module keras.src.engine.sequential object:

class Sequential(keras.src.engine.functional.Functional)

| Sequential(layers=None, name=None)

|

| `Sequential` groups a linear stack of layers into a `tf.keras.Model`.

|

| `Sequential` provides training and inference features on this model.

|

| Examples:

|

| ```python

| model = tf.keras.Sequential()

| model.add(tf.keras.Input(shape=(16,)))

| model.add(tf.keras.layers.Dense(8))

|

| # Note that you can also omit the initial `Input`.

```
| # In that case the model doesn't have any weights until the first call
| # to a training/evaluation method (since it isn't yet built):
| model = tf.keras.Sequential()
| model.add(tf.keras.layers.Dense(8))
| model.add(tf.keras.layers.Dense(4))
| # model.weights not created yet
|
| # Whereas if you specify an `Input`, the model gets built
| # continuously as you are adding layers:
| model = tf.keras.Sequential()
| model.add(tf.keras.Input(shape=(16,)))
| model.add(tf.keras.layers.Dense(4))
| len(model.weights)
| # Returns "2"
|
| # When using the delayed-build pattern (no input shape specified), you can
| # choose to manually build your model by calling
| # `build(batch_input_shape)`:
| model = tf.keras.Sequential()
| model.add(tf.keras.layers.Dense(8))
| model.add(tf.keras.layers.Dense(4))
| model.build((None, 16))
| len(model.weights)
| # Returns "4"
|
| # Note that when using the delayed-build pattern (no input shape specified),
```

| # the model gets built the first time you call `fit`, `eval`, or `predict`,
| # or the first time you call the model on some input data.
| model = tf.keras.Sequential()
| model.add(tf.keras.layers.Dense(8))
| model.add(tf.keras.layers.Dense(1))
| model.compile(optimizer='sgd', loss='mse')
| # This builds the model for the first time:
| model.fit(x, y, batch_size=32, epochs=10)
| ```
|
| Method resolution order:
| Sequential
| keras.src.engine.functional.Functional
| keras.src.engine.training.Model
| keras.src.engine.base_layer.Layer
| tensorflow.python.module.module.Module
| tensorflow.python.trackable.autotrackable.AutoTrackable
| tensorflow.python.trackable.base.Trackable
| keras.src.utils.version_utils.LayerVersionSelector
| keras.src.utils.version_utils.ModelVersionSelector

about this model type:<class 'keras.src.engine.sequential.Sequential'>

==========
==========

GO! Bot is running!

```
"""

def clean_up_sentence(sentence):

    # Tokenize the sentence into individual words
    sentence_words = nltk.word_tokenize(sentence)

    # Lemmatize each word to its base form
    sentence_words = [lemmatizer.lemmatize(word) for word in sentence_words]
    return sentence_words
```

How are you ?

sentence in clean_up_sentence: How are you ? ,

type:<class 'str'>

==========

==========

sentence_words in clean_up_sentence: ['How', 'are', 'you', '?']

, type: <class 'list'>

==========

==========

```
def bag_of_words(sentence):
    # Clean up the sentence
    sentence_words = clean_up_sentence(sentence)

    # Create a bag of words representation
    bag = [0] * len(words)
    for w in sentence_words:
        for i, word in enumerate(words):
            if word == w:
                bag[i] = 1

    return np.array(bag)
```

sentance_words in bag_of_words: ['How', 'are', 'you', '?']

, type:<class 'list'>

==
==

bag [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0] type:<class 'list'>

==
==

bag array [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]

type:<class 'list'>

==
==

```python
def predict_class(sentence):
    # Convert the sentence into a bag of words
    bow = bag_of_words(sentence)

    # Predict the intent using the loaded model
    res = model.predict(np.array([bow]))[0]

    ERROR_THRESHOLD = 0.25
    # Filter out predictions below the error threshold
    results = [[i, r] for i, r in enumerate(res) if r > ERROR_THRESHOLD]

    # Sort the results by probability in descending order
    results.sort(key=lambda x: x[1], reverse=True)

    return_list = []
    for r in results:
        # Get the corresponding intent and its probability
        return_list.append({'intent': classes[r[0]], 'probability': str(r[1])})

    return return_list
```

bow in predict_class: [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]

type:<class 'numpy.ndarray'>

==
====================================

1/1 [==============================] - 0s 70ms/step

res in predict_class: [0.01277256 0.00935072 0.0377065 0.41800627
0.01631615 0.5058478]   type:<class 'numpy.ndarray'>

==
====================================

results: [[5, 0.5058478], [3, 0.41800627]]

type:<class 'list'>

==
====================================

return_list in predict_class: [{'intent': 'name', 'probability': '0.5058478'},
{'intent': 'greeting', 'probability': '0.41800627'}]

, type:<class 'list'>

==
====================================

ints:[{'intent': 'name', 'probability': '0.5058478'},

{'intent': 'greeting', 'probability': '0.41800627'}]

type: <class 'list'>

```python
def get_response(intents_list, intents_json):
    # Get the predicted intent
    tag = intents_list[0]['intent']

    list_of_intents = intents_json['intents']

    for i in list_of_intents:

        if i['tag'] == tag:

            # Randomly choose a response from the matched intent
            result = random.choice(i['responses'])
            break

    return result
```

tag in get_response: name

, type: <class 'str'>

==========

list_of_intents in get_response: [{

'tag': 'greeting',

'patterns': ['Hi', 'How are you', 'Is anyone there?', 'Hello', 'Good day', 'Whats up', 'Good morning', 'Good evening', 'hello', 'hey', "what's up"],

'responses': ['Hello! How can I assist you today?', 'Good to see you! How may I help you?', 'Hi there, how can I assist you?'],

'context_set': ''},

{'tag': 'goodbye',

'patterns': ['cya', 'See you later', 'Goodbye', 'I am Leaving', 'Have a Good day', 'bye', 'i have to go', 'gotta go'], 'responses': ['Sad to see you go. Have a great day!', 'Talk to you later. Take care!', 'Goodbye! Have a wonderful day!'], 'context_set': ''},

{'tag': 'age',

'patterns': ['how old', 'how old are you', 'what is your age', 'how old are you', 'age?'], 'responses': ["I am a virtual assistant, so I don't have an age!", "I'm an AI-powered bot, so age doesn't apply to me!"], 'context_set': ''},

{'tag': 'name',

'patterns': ['what is your name', 'what should I call you', 'whats your name?', 'who are you?'], 'responses': ['You can call me CallBot.', "I'm CallBot!", "I'm your friendly CallBot."], 'context_set': ''}, {'tag': 'help',

'patterns': ['Id like to ask something', 'what can you do', 'can you help me?', 'can i tell you something'], 'responses': ["I'm here to help you! How can I assist you today?", 'I can help you with a wide range of inquiries. What do you need assistance with?', "I'm here to assist you. Please let me know how I can help."], 'context_set': ''},

{'tag': 'customer_service',

'patterns': ['I have a question about my order', 'I need assistance with a product', 'Can you help me with a billing issue?', 'I want to provide feedback', 'I need technical support', 'Can you transfer me to a live agent?'], 'responses': ['Sure, I can assist you with that. Please provide me with more details about your question or issue.', "Of course! I'm here to help. Please let me know the specific problem or question you have.", "I'll do my best to assist you. Please provide me with more information about your request."], 'context_set': ''}]

, type: <class 'list'>

==========

result after random in get_response: I'm your friendly CallBot.

type: <class 'str'>

==========

The final response:

I'm your friendly CallBot