P O L I T E C H N I K A R Z E S Z O W S K A
im. Ignacego Łukasiewicza
WYDZIAŁ MATEMATYKI I FIZYKI STOSOWANEJ

Wprowadzenie do programowania w języku Python

Dokumentacja Projektu

Żak Maciej

Rzeszów, 2022

Spis treści

Temat projektu

Budowa prostego chat-bota.

Zamysł projektu

Głównym zamysłem projektu jest, stworzenie prostego chatboota, który będzie analizował wprowadzone dane/zdania oraz na bazie tej analizy udzielał odpowiedniej odpowiedzi.

Opis użytych bibliotek:

random - Biblioteka random w Pythonie dostarcza funkcje do generowania pseudolosowych liczb i wybierania losowych elementów z kolekcji. Można użyć jej do generowania liczb całkowitych, liczb zmiennoprzecinkowych, wyboru elementów z listy lub losowego dobierania kombinacji lub permutacji. Biblioteka ta jest często używana w programowaniu do symulacji, gier i testów.

json - Biblioteka json w Pythonie pozwala na przekształcanie danych w formacie JSON na struktury danych Python (np. słowniki, listy) i na odwrót. JSON (JavaScript Object Notation) jest popularnym formatem przesyłania i przechowywania danych, często stosowanym w internecie. Dzięki bibliotece json można łatwo parsować i generować pliki JSON w kodzie Python.

pickle - Biblioteka pickle w Pythonie pozwala na serializację i deserializację obiektów Python. Serializacja polega na przekształceniu obiektów (np. list, słowników) na postać binarną, która może być zapisana do pliku lub przesłana przez sieć. Deserializacja to proces odwrotny, polegający na przekształceniu danych binarnych z powrotem na obiekty Python. Biblioteka pickle jest często używana do zapisywania stanu programu lub do przesyłania danych między różnymi procesami.

numpy - Biblioteka NumPy w Pythonie zapewnia wysokiej jakości struktury danych i narzędzia do przetwarzania numerycznego. Główną jej zaletą jest wsparcie dla wielowymiarowych tablic (nazywanych "macierzami" lub "tablicami numpy"), które pozwalają na efektywne przeprowadzanie operacji matematycznych na dużych zbiorach danych. Biblioteka ta jest często używana w naukach przyrodniczych, inżynierii oraz uczeniu maszynowym. Posiada wiele funkcji pozwalających na operowanie na tablicach oraz wykonywanie operacji matematycznych, statystycznych oraz algebraicznych.

Opis pliku tworzącego model

Na początku kodu importowane są niezbędne biblioteki, takie jak "random", "json", "pickle", "numpy" oraz "nltk" z modułem "WordNetLemmatizer". Następnie za pomocą biblioteki "json" wczytywane są dane wejściowe z pliku "intents.json", które zawierają intencje oraz odpowiedzi chatbota dla danych intencji.

Tworzymy dwa pliki picle, „words” oraz „classes”. Pliki pickle to pliki, które pozwalają na zapisywanie obiektów Pythona w formacie binarnym. W tym przypadku, plik words.pkl zawiera listę słów, które zostały znalezione w intencjach z pliku intents.json. Lista ta jest przetworzona przez lematyzację i oczyszczona z symboli interpunkcyjnych. Plik classes.pkl zawiera natomiast listę klas intencji, które zostały znalezione w pliku intents.json. Te pliki pickle służą do zapisywania stanu modelu po treningu, dzięki czemu możliwe jest późniejsze wykorzystanie modelu bez konieczności ponownego treningu.

Następnie kod przygotowuje dane do treningu, tworząc reprezentację bag-of-words dla każdego dokumentu oraz odpowiedni wektor wyjściowy. Dane są losowo tasowane i przygotowywane jako dane wejściowe oraz wyjściowe dla modelu.

Tworzymy model sieci neuronowej, który jest używany jako część chatbota. Model jest oparty na bibliotece Tensorflow i składa się z trzech warstw gęstych (Dense) z funkcjami aktywacji ReLU. Warstwa wejściowa ma 128 neuronów, druga warstwa ma 64 neurony, a trzecia warstwa ma tyle neuronów, ile jest klas (intencji) w danych wejściowych. Są to warstwy ukryte. Na końcu jest warstwa wyjściowa, która ma liczbę neuronów równą liczbie klas i funkcję aktywacji softmax. Warstwy te są połączone ze sobą w taki sposób, że każda warstwa jest połączona z następną warstwą. W warstwie ukrytych dodano Dropout, który ma na celu zapobieganie przeuczeniu.

Model jest trenowany na danych wejściowych, które są przetwarzane i przygotowywane wcześniej w pliku json. Trenowanie odbywa się przez 200 epok, a rozmiar paczki to 5. Następnie model jest zapisywany do pliku. Na końcu kodu wyświetlany jest komunikat "Done", co oznacza, że proces przygotowywania danych do treningu oraz trenowanie modelu zakończył się pomyślnie.

Implementacja kodu trainingDATA.py

Importowanie odpowiednich bibliotek

```
import random
import json
import pickle
import numpy as np

import nltk
from nltk.stem import WordNetLemmatizer

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import SGD
```

Inicjalizacja słownika do odmiany wyrazów

```
lemmatizer = WordNetLemmatizer()
```

Wczytanie danych wejściowych

```
intents = json.loads(open('intents.json').read())
```

Przygotowanie listy wyrazów, klas i dokumentów

```
words = []
classes = []
documents = []
ignore_letters = ['?', '!',',','.']
```

Przetwarzanie intencji, w tym celu używamy pętli FOR

```
for intent in intents['intents']:
    for pattern in intent['patterns']:
        word_list = nltk.word_tokenize(pattern)
        words.extend(word_list)
        documents.append((word_list,intent['tag']))
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
```

Lematyzacja, czyli sprowadzenie danych do formy bazowej oraz oczyszczenie listy wyrazów

```
words = [lemmatizer.lemmatize(word) for word in words if word not in ignore_letters]
words = sorted(set(words))
```

Sortowanie klas

```
classes = sorted(set(classes))
```

Zapisanie listy wyrazów i klas do pliku

```
pickle.dump(words, open('words.pkl', 'wb'))
pickle.dump(classes, open('classes.pkl', 'wb'))
```

Przygotowanie danych do treningu

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

for document in documents:
    bag =[]
    word_patterns = document[0]
    word_patterns = [lemmatizer.lemmatize(word.lower()) for word in word_patterns]
    for word in words:
        bag.append(1) if word in word_patterns else bag.append(0)

    output_row = list(output_empty)
    output_row[classes.index(document[1])] = 1
    training.append([bag, output_row])
```

Losowanie danych treningowych

```
random.shuffle(training)
```

Przygotowanie danych wejściowych i wyjściowych

```
train_x = np.array([np.array(i[0]) for i in training])
train_y = np.array([np.array(i[1]) for i in training])
```

Tworzenie modelu

```
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(learning_rate=0.01, 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('chatbotmodel.h5', hist)

print('Done')
```

Opis pliku main.py

Funkcja main jest główną funkcją w projekcie czatbota. W pierwszej kolejności wczytuje ona dane wejściowe, takie jak intenty, słowa i klasy, które zostały wcześniej przygotowane i zapisane w plikach json, pickle. Wczytanie słownika do odmiany wyrazów oraz modelu, który został już wytrenowany.

Następnie, funkcja ta zawiera kilka pomocniczych funkcji, takie jak "clean_up_sentence" (która przygotowuje ciąg znaków do dalszej analizy), "bag_of_words" (która przekształca zdanie na wektor bag-of-words) i "predict_class" (która przewiduje klasę na podstawie wektora bag-of-words).

"get_response" funkcja jest odpowiedzialna za pobieranie odpowiedzi na podstawie klasy, która została przewidziana przez "predict_class" funkcję. Pętla główna programu, która jest zawarta w tym fragmencie kodu, umożliwia użytkownikowi wprowadzanie wiadomości, a następnie przetwarzanie jej przez powyższe funkcje i wyświetlenie odpowiedzi czatbota.

Program kończy się po wprowadzeniu wiadomości "bye" lub "Goodbye" przez użytkownika.

Implementacja kodu main.py

Zaczynamy od zaimportowania odpowiednich bibliotek

```
import random
import json
import pickle
import numpy as np

import nltk
from nltk.stem import WordNetLemmatizer


from tensorflow.keras.models import load_model
```

Inicjalizacja słownika do odmiany wyrazów

```
lemmatizer = WordNetLemmatizer()
```

Wczytywanie danych wejściowych oraz wczytanie modelu

```
intents = json.loads(open('intents.json').read())

words = pickle.load(open('words.pkl', 'rb'))
classes = pickle.load(open('classes.pkl', 'rb'))
model = load_model('chatbotmodel.h5')
```

Tokenizacja zdania i lematyzacja wyrazów

```
def clean_up_sentence(sentence):
    sentence_words = nltk.word_tokenize(sentence)
    sentence_words = [lemmatizer.lemmatize(word)  for word in sentence_words]
    return sentence_words
```

Przekształcenie zdania na wektor bag-of-words

```
def bag_of_words(sentence):
    sentence_words= clean_up_sentence(sentence)
    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)
```

Predykcja klasy

```
def predict_class(sentence):
    bow = bag_of_words(sentence)
    res = model.predict(np.array([bow]))[0]
    ERROR_THRESHOLD = 0.25
    results = [[i,r] for i, r in enumerate(res) if r > ERROR_THRESHOLD]

    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
```

Pobranie odpowiedzi na podstawie klasy

```
def get_response(intents_list, intents_json):
    tag= intents_list[0]['intent']
    list_of_intents =intents_json['intents']
    for i in list_of_intents:
        if i['tag'] == tag:
            result = random.choice(i['responses'])
            break
    return result
```

Głowna pętla programu

```
while True:
    message = input("| You: ")
    if message == "bye" or message == "Goodbye":
        ints = predict_class(message)
        res = get_response(ints, intents)
        print("| Bot:", res)
        print("|==================== The Program End here! ====================|")
        exit()

    else:
        ints = predict_class(message)
        res = get_response(ints, intents)
        print("| Bot:", res)
```

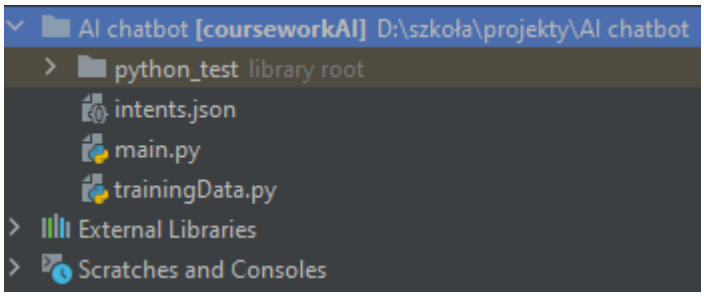
Opis pliku intens.json

Plik intens.json inicjalizujemy jako dane wejściowe w plikach main oraz trainingDATA. Plik JSON zawiera listę "intents", które służą do określenia intencji użytkownika i odpowiedzi na nią. Każdy "intent" składa się z "tag" (etykiety), "patterns" (wzorów) i "responses" (odpowiedzi). Etykiety określają temat intencji, wzory to frazy, które mogą być używane przez użytkownika, a odpowiedzi to teksty, które są wyświetlane przez system.


```
{"intents": [
  {"tag": "greetings",
  "patterns": ["hello","hey","hi","good day","Greetings","what's up?","how is it going?"],
  "responses": ["Hello!","Hey!","What can I do for you?"]
  },

  {"tag": "name",
  "patterns": ["what is your name","name","what's your name","who are you","what should I call you"],
  "responses": ["You can call me bobot","I'm bobot","I'm bobot your virtual assistant"]
  },

  {"tag": "goodbye",
  "patterns": ["cya","See you later","Goodbye","I am leaving","Have a Good Day","bye","see ya"],
  "responses": ["Sad to see you go :(","Talk you later","Goodbye"]
  },

  {"tag": "invalid",
    "patterns": ["","gvsd","asbhk"],
    "responses": ["Sorry, can't understand you", "Please give me more info", "Not sure I understand"]
  },

  {"tag": "thanks",
    "patterns": ["Thanks", "Thank you", "That's helpful", "Awesome, thanks", "Thanks for helping me"],
    "responses": ["Happy to help!", "Any time!", "My pleasure"]
  },
  {"tag": "location",
  "patterns": ["Where are you located?", "What is your location?", "Where are you based?"],
  "responses": ["I am a virtual assistant, so I am located in the cloud.", "My location is virtual."]
  },

  {"tag": "time",
  "patterns": ["What time is it?", "Can you tell me the time?", "What's the current time?"],
  "responses": ["I'm sorry, I am not able to know the current time. But you can check your device for the current time."]
  },

  {"tag": "age",
  "patterns": ["How old are you?", "What's your age?", "When were you created?"],
  "responses": ["I am a recent creation, I was created in 2021.", "I am a young model, I am not able to know my age."]
  },

  {"tag": "weather",
  "patterns": ["What's the weather like?", "How's the weather?", "Is it raining?"],
  "responses": ["I'm sorry, I am not able to know the current weather. But you can check your device for the current weather information."]
  },

  {"tag": "help",
  "patterns": ["Can you help me?", "I need assistance.", "Can you assist me?"],
  "responses": ["Of course! What do you need help with?", "Sure, how can I assist you?", "What do you need help with?"]
  },
```

Działanie programu

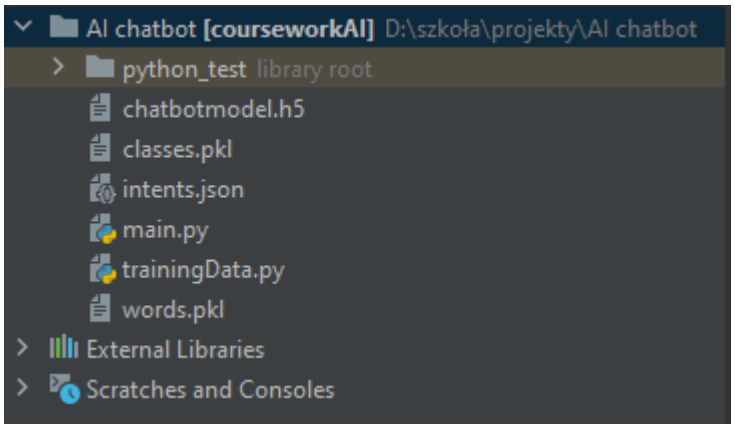
Pliki przed uruchomieniem kodu traningData.py



Tworzenie modelu na podstawie danych z pliku Json

```
Epoch 184/200
15/15 [==============================] - 0s 2ms/step - loss: 0.1070 - accuracy: 0.9467
Epoch 185/200
15/15 [==============================] - 0s 2ms/step - loss: 0.1674 - accuracy: 0.9200
Epoch 186/200
15/15 [==============================] - 0s 2ms/step - loss: 0.1410 - accuracy: 0.9600
Epoch 187/200
15/15 [==============================] - 0s 1ms/step - loss: 0.2035 - accuracy: 0.9067
Epoch 188/200
15/15 [==============================] - 0s 1ms/step - loss: 0.1876 - accuracy: 0.9067
Epoch 189/200
15/15 [==============================] - 0s 1ms/step - loss: 0.0949 - accuracy: 0.9600
Epoch 190/200
15/15 [==============================] - 0s 1ms/step - loss: 0.1248 - accuracy: 0.9600
Epoch 191/200
15/15 [==============================] - 0s 1ms/step - loss: 0.1418 - accuracy: 0.9467
Epoch 192/200
15/15 [==============================] - 0s 2ms/step - loss: 0.1496 - accuracy: 0.9200
Epoch 193/200
15/15 [==============================] - 0s 2ms/step - loss: 0.2120 - accuracy: 0.9200
Epoch 194/200
15/15 [==============================] - 0s 2ms/step - loss: 0.1573 - accuracy: 0.9333
Epoch 195/200
15/15 [==============================] - 0s 2ms/step - loss: 0.1846 - accuracy: 0.9333
Epoch 196/200
15/15 [==============================] - 0s 2ms/step - loss: 0.2357 - accuracy: 0.9467
Epoch 197/200
15/15 [==============================] - 0s 1ms/step - loss: 0.1566 - accuracy: 0.9467
Epoch 198/200
15/15 [==============================] - 0s 1ms/step - loss: 0.1602 - accuracy: 0.9333
Epoch 199/200
15/15 [==============================] - 0s 1ms/step - loss: 0.1047 - accuracy: 0.9600
Epoch 200/200
15/15 [==============================] - 0s 1ms/step - loss: 0.1343 - accuracy: 0.9467
Done
```

Pliki po uruchomieniem kodu traningData.py



Rozmowa z botem

```
| You: Hi
1/1 [==============================] - 0s 76ms/step
| Bot: Goodbye
| You: What's your name?
1/1 [==============================] - 0s 26ms/step
| Bot: I'm bobot your virtual assistant
| You: How old are you?
1/1 [==============================] - 0s 16ms/step
| Bot: I am a young model, I am not able to know my age.
| You: Can you help me?
1/1 [==============================] - 0s 17ms/step
| Bot: Of course! What do you need help with?
```

Jak widzimy bot jest w stanie się z nami komunikować w zrozumiały sposób .Jeżeli chcemy, aby bot był w stanie rozmawiać, na wiele tematów koniecznym było by zwiększenie danych, na których trenujemy model.

Podsumowanie

W projekcie, który przeprowadziliśmy, skupiliśmy się na stworzeniu czatbota w języku Python. W celu osiągnięcia tego celu skorzystaliśmy z biblioteki NLTK, która umożliwiła nam przetwarzanie języka naturalnego, a także z biblioteki random, która pozwoliła na losowanie odpowiedzi bota.

Kod źródłowy, który stworzyliśmy składa się z kilku funkcji, które pozwalają na wprowadzanie pytań przez użytkownika, a następnie na generowanie odpowiedzi przez bota. Funkcja ta wykorzystuje algorytm klasyfikacji, aby określić, jakie pytanie zostało zadane i jakiej odpowiedzi bot powinien udzielić.