Chatbot selbsterstellen in Python

khatere kazemi

Seminar:Anwendungen und Systeme2

Holger Holm Grumt Suarez

september 2019

Was ist ein Chatbot?

Ein Chatbot ist eine Software mit künstlicher Intelligenz in einem Gerät (Siri, Alexa, Google Assistant usw.), einer Anwendung, einer Website oder einem anderen Netzwerk, die versucht, die Bedürfnisse des Verbrauchers zu ermitteln.

Wie funktionieren Chatbots?

Es gibt 2 Arten der Chatbots:

1- Regelbasiert

2- Selbstlernend

1.Regelbasierte chatbots haben eine Anweisung mit einer Liste von Befehlen und Schlüsselwörtern, um Fragen zu beantworten.

2.Selbstlernende Chatbots funktionieren nach dem Prinzip der künstlichen Intelligenz und sind effektiver als regelbasierte Chatbots. Es gibt zwei Arten von selbstlernenden Chatbots:

- Suchbots,
- Generative Bots.

suchbots:verwenden heuristische Methoden, um eine Antwort aus einer Bibliothek von vordefinierten Replikaten auszuwählen.

Generative:können ihre eigenen Antworten erstellen und reagieren nicht immer auf eine der vordefinierten Optionen.

Bot bauen

Voraussetzungen Es werden praktische Kenntnisse der Scikit-Bibliothek und des NLTK vorausgesetzt.

1.NLP

Das Fachgebiet, das sich auf die Wechselwirkungen zwischen menschlicher Sprache und Computern konzentriert, heißt menschlicher Sprache und Computern konzentriert, heißt Natural Language Processing, kurz NLP.

2.NLTK

NLTK (Natural Language Toolkit) ist eine führende Plattform zum Erstellenvon Python-Programmen für die Arbeit mit menschlichen Sprachdaten.

3.Text Pre-Processing with NLTK

Die grundlegende Textvorverarbeitung umfasst:

- **uppercase or lowercase:Konvertieren Sie Buchstaben in Groß oder Klein buchstaben, damit der Algorithmus die gleichen Wörter nicht erneut verarbeitet**
- **Tokenization:der Prozess des Konvertierens der normalen Textzeichenfolgen in eine Liste von Tokenn**
- **RemovingNoise:d.h. alles, was keine Zahl oder kein Buchstabe ist**

  EntfernenvonStoppwörtern
- **Stemming: das Wort zur Wurzelbedeutung zwingen.**
- **Lemmatisierung:Eine leichte Variante der Stemmung ist die Lemmatisierung**

4.Bag ofWords

Nach deme rsten Schritt der Vorverarbeitung muss der Text in einen Vektor von Zahlen konvertieren werden.Word,Set besteht aus einem Wörterbuch mit berühmten Wörtern und aus den Häufigkeiten,auf die jedes Wort im Text trifft.

5.TF-IDF Approach

Das Problem mit dem "Word-Set" liegt darin, dass der Text von häufig vorkommenden Wörtern dominiert werden kann, die keine wertvollen Informationen enthalten. Auch "Word-Set" gibt den langen Texten mehr Bedeutung als den kurzen Texten.

Ein Ansatz zur Lösung dieser Probleme besteht darin,die Häufigkeit des Auftretens des Wortes nicht in einem Txet,sondern auf einmal zuberechnen.Dieser Ansatz wird als TF-IDF (Term Frequency-Inverse Document Frequency) bezeichnet und besteht aus zwei Schritten:;

- **TF–Berechnung der Wort häufigkeit in einem Text**

• IDF–Berechnung,wie selten ein Wort in allen Texten vorkommt.

6.Cosine Similarity

Cosin Similarity ist ein Maß für die Ähnlichkeit zwischen zwei Nicht-Null-Vektoren.Cosine similarity modul wird aus der scikit learn library impottiert.

Überblick über Chatbot

Dieser Chatbot kann ein Dokument mit Textinformationen analysieren und die Fragen des Benutzers beantworten. Der Chatbot verwendet das Natural Language Processing Toolkit (NLTK), um die Textinformationen zu verarbeiten

Zunächst werden wir NLTK- und String-Bibliotheken importieren und einige Daten herunterladen, die zum Verarbeiten von Text von nltk benötigt werden.

```
import nltk
import string
```

```
#Download only once
nltk.download('punkt')  # vorgeübter Tokenizer für Englisch
nltk.download('wordnet') # lexikalische Datenbank für die englische Sprache
```

Jetzt müssen wir einige Informationen in den Chatbot eingeben, damit er unsere Fragen beantworten kann. Kopieren wir einige Informationen aus dem Internet und speichern Sie wir in einer Variablen. Stellen wir sicher, dass Sie mindestens 3 Sätze haben, damit unser Chatbot ein bisschen weiser wird. wir können auch Daten von einer Datei oder Webseite laden, wie im Originalcode gezeigt.

```
text = """Ein Chatterbot, Chatbot oder kurz Bot ist ein textbasiertes Dialogsystem, welches das Chatten mit einem technischen System erlaubt. Er hat je einen Bereich zur Textein- und -ausgabe, über die sich in natürlicher Sprache mit dem dahinterstehenden System kommunizieren lässt. Chatbots können, müssen aber nicht in Verbindung mit einem Avatar benutzt werden. Technisch sind Bots näher mit einer Volltextsuchmaschine verwandt als mit künstlicher oder gar natürlicher Intelligenz. Mit der steigenden Computerleistung können Chatbot-Systeme allerdings immer schneller auf immer umfangreichere Datenbestände zugreifen und daher auch intelligente Dialoge für den Nutzer bieten. Solche Systeme werden auch als virtuelle persönliche Assistenten bezeichnet.
Es gibt auch Chatbots, die gar nicht erst versuchen, wie ein menschlicher Chatter zu wirken (daher keine Chatterbots), sondern ähnlich wie IRC-Dienste nur auf spezielle Befehle reagieren. Sie können als Schnittstelle zu Diensten außerhalb des Chats dienen, oder auch Funktionen nur innerhalb ihres Chatraums anbieten, z. B. neu hinzugekommene Chatter mit dem Witz des Tages begrüßen....."""
```

Wir werden jetzt alle Buchstaben im Text in Kleinbuchstaben umwandeln, um sicherzustellen, dass aufgrund der Groß- und Kleinschreibung nicht dasselbe Wort mehrmals gezählt wird.

```
text = text.lower()
```

Jetzt müssen wir die Wörter und Sätze mit einem Token versehen. In den folgenden Codezeilen werden zwei Listen erstellt, die erste besteht aus allen Sätzen im Text und die zweite aus allen Wörtern

im Text. Wir werden die Satz-Tokenizer- und Wort-Tokenizer-Methoden von nltk verwenden, wie unten gezeigt.

```
sentences = nltk.sent_tokenize(text)
tokens = nltk.word_tokenize(text)
```

Nach dem Tokenisieren werden die Token durch Lemmatisieren, Entfernen der Stoppwörter und Entfernen der Interpunktionen bereinigt. Lemmatisieren ist der Prozess der Umwandlung eines Wortes in seine Wurzelform. Beispielsweise haben Wörter wie "run", "ran" und "running" dieselbe Bedeutung und müssen daher nicht als unterschiedliche Wörter betrachtet werden. Durch die Lemmatisierung werden alle Wörter reduziert zu run. Stoppwörter stellen die in der natürlichen Sprache am häufigsten verwendeten Wörter wie "a", "is", "what" usw. dar, die keinen Mehrwert für die Funktionen des Textklassifikators bieten. Daher entfernen wir sie auch. WordNet ist ein semantisch orientiertes Wörterbuch des Englischen, das in NLTK enthalten ist.

```
#Initialisierung des WordNetLemmatizers
lemmer = nltk.stem.WordNetLemmatizer()

#Importing the stopwords
from nltk.corpus import stopwords

#Lemmatizing the words or tokens
def LemTokens(tokens):
  return [lemmer.lemmatize(token,'v') for token in tokens if token not in
set(stopwords.words('english')) ]
```

Der obige Codeblock reduziert ein Wort auf seine Grundform, während auch die Stoppwörter entfernt werden.

```
#Eine Wörterbuchreferenz zum Ersetzen von Sonderzeichen
remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
```

Im folgenden Codeblock definieren wir eine Methode namens LemNormalize, um die zuvor definierte Funktion LemTokens und das Wörterbuch remove_punct_dict zum Lemmatisieren sowie zum Bereinigen und Tokenisieren des Texts zu verwenden.

```
#Methode zum Aufräumen und Tokenisieren des Texts
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
```

Die translate-Methode konvertiert alle Interpunktionen, die in den Schlüsseln von remove_punct_dict definiert sind, in ihre jeweiligen Werte, die None sind.

Als nächstes wird eine Funktion für eine Begrüßung durch den Bot definieren, d. H. Wenn eine Benutzereingabe eine Begrüßung ist, gibt der Bot eine Begrüßungsantwort zurück.function split():Gibt eine Liste von Zeichenfolgen zurück

```
GREETING_INPUTS = ("hello", "hi", "greetings", "sup", "what's up","hey",)
GREETING_RESPONSES = ["hi", "hey", "*nods*", "hi there", "hello", "I am glad! You aretalking to me"]
def greeting(sentence):
"""If user's input is a greeting, return a greeting response"""
  for word in sentence.split():
    if word.lower() in GREETING_INPUTS:
     return random.choice(GREETING_RESPONSES)
```

Im folgenden Code konvertieren wir die Sätze mit der CountVectorizer-Methode in das Bag-of-Words-Modell. Anschließend prüfen wir, ob cosine similarity zwischen den Eingaben der Benutzer und den Sätzen in der Worttüte identisch ist.

```
#Importieren der Bibliotheken für cosine_similarity & CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer
```

Wir können nun eine Methode definieren, um die Benutzereingaben aufzunehmen und die Ähnlichkeit mit den Sätzen im Text zu überprüfen.

```
1 def response(user_response):
2     sentences.append(user_response)
3     cv = CountVectorizer(max_features = 50, tokenizer = LemNormalize, analyzer = 'word')
4     X = cv.fit_transform(sentences)
5     vals_cv = cosine_similarity(X[-1], X)
6     indx_of_most_similar_sentence = vals_cv.argsort()[0][-2] #sorting the indexes based on
increasing similarity
7     flat_vals_cv = vals_cv.flatten()
8     flat_vals_cv.sort()
9     highest_similarity = flat_vals_cv[-2] # required tfidf = most similar to

10    if(highest_similarity == 0):
          robo_response = "I am sorry! I don't understand you"
          return robo_response
11    else:
          robo_response = sentences[indx_of_most_similar_sentence]
          return robo_response
```

Wir vergleichen den obigen Code mit dem folgenden Pseudocode, um den Prozess in jeder Zeile zu verstehen.

1) Funktionsdefinition: - Wir definieren die Antwortfunktion.

2) wir hinzufügen der Benutzereingabe zum Satzkorpus.

3) wir initialisieren den CountVectorizer mit den Parametern max_features = 50, tokenizer = LemNormalize und analyzer = 'word'. Der CountVectorizer erstellt eine Matrix mit jedem Wort als Spalten, Zeilen, die Sätze darstellen, und den Werten, die die Anzahl jedes Wortes in jedem Satz darstellen.

Mit max_features = 50 werden 50 Wörter aus dem Satzkorpus als Spalten oder Merkmale und jeder Satz als Zeile ausgewählt. Wenn der Text beispielsweise 4 Sätze enthält, erstellt der CountVectorizer einen Vektor der Form 4 × 50.

4) Umwandlung des Satzkorpus in count_vectorizer X.

5) Berechnung der cosin similarity des letzten Satzes (Benutzereingabe) mit dem gesamten CountVerter X.

Die Cosinus-similarityt wird berechnet als das Verhältnis zwischen den Punktprodukten des Auftretens und dem Produkt der Größe des Auftretens von Termen. Dies ergibt ein Array der Länge 4 für einen Text, der 4 Sätze enthält (der 4. Satz ist die Benutzereingabe), wobei die cosin similarity als Elemente verwendet wird. Der letzte Satz weist immer die höchste cosin similarity auf, da es sich um die Benutzereingabe handelt.

6) wir Sortieren der Indizes des Arrays mit cosin similarity in aufsteigender Reihenfolge und Aufnehmen des vorletzten Elements. Dies gibt den Index des ähnlichsten Satzes an.

7) Abflachen des cosin similarity arrays in einen Zeilenvektor.

8) wir sortieren der Werte in aufsteigender Reihenfolge der cosin similarity.

9) wir speichern des zweithöchsten cosin similarity werts.

10) Wenn der zweithöchste cosin similarity wert Null ist, bedeutet dies, dass keine Übereinstimmung vorliegt und Chatbot eine Meldung ausgibt, die besagt, dass die Benutzerabfrage nicht verstanden werden kann.

11) Andernfalls zeigt chatbot dem Benutzer die übereinstimmende Zeile aus dem Text an.

Der folgende Codeblock führt eine Schleife aus, um die Chat-Sitzung fortzusetzen, bis der Benutzer mit einem der Beendigungscodes beendet wird oder die Frage, ob die Sitzung fortgesetzt werden soll, mit "Nein" beantwortet wird. Jedes Mal, wenn der Benutzer einen Satz oder ein Wort eingibt, wird es an die oben erläuterte Antwortmethode übergeben und gibt eine Übereinstimmung zurück, wenn ähnliche Sätze gefunden werden, andernfalls wird eine Standardnachricht angezeigt.

```
flag=True
print("ROBO: My name is Robo. I will answer your queries about Chatbots. If you want to exit, type Bye!")
while(flag==True):
    user_response = input()
```

```
    user_response=user_response.lower()
    if(user_response!='bye'):
        if(user_response=='thanks' or user_response=='thank you' ):
            flag=False
            print("ROBO: You are welcome..")
        else:
            if(greeting(user_response)!=None):
                print("ROBO: "+greeting(user_response))
            else:
                print("ROBO: ",end="")
                print(response(user_response))
                sent_tokens.remove(user_response)
    else:
        flag=False
        print("ROBO: Bye! take care..")
```

Hier ist eine Beispiel-Chat-Sitzung mit Chatty:

```
ROBO: My name is Robo. I will answer your queries about Chatbots. If you want to exit, type Bye!
hi
ROBO: hi there
```

```
# Import dernotwendigenBibliotheken
```

```
import random
import string
import
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import nltk
from nltk.stem import WordNetLemmatizer
nltk.download('punkt') #nur zum ersten mal
nltk.download('wordnet') #nur zum ersten mal

 #TOkenisation

sent_tokens = nltk.sent_tokenize(raw)
word_tokens = nltk.word_tokenize(raw)

 # Preprocessing

lemmer = nltk.stem.WordNetLemmatizer()
def LemTokens(tokens):
return [lemmer.lemmatize(token) for token in tokens]
remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))

 # K eyword-Auswahl
      GREETING_INPUTS = ("hello", "hi", "greetings", "sup", "what's up","hey",)
GREETING_RESPONSES = ["hi", "hey", "*nods*", "hi there", "hello", "I am glad! You aretalking to me"]
def greeting(sentence):
"""If user's input is a greeting, return a greeting response"""
  for word in sentence.split():
    if word.lower() in GREETING_INPUTS:
      return random.choice(GREETING_RESPONSES)

 # Antwortgenerierung
def response(user_response):
sentences.append(user_response)
cv = CountVectorizer(max_features=50,tokenizer=LemNormalize,analyzer='word')
 X= cv. fit_transform(sentences)
vals_cv = cosine_similarity(X[-1], X)
idx_of_most_similar_sentence=vals_cv.argsort()[0][-2]



      flat_vals_cv = vals_cv.flatten()
       flat_vals_cv.sort()
       highest_similarity = flat_vals_cv [-2]
       if(highest_similarity ==0):
       robo_response= "I am sorry! I don't understand you"
```

```
        return robo_response
        else:
        robo_response = sentences[indx_of_most_similar_sentence]
return robo_response

flag=True
print("ROBO: My name is Robo. I will answer your queries about Chatbots. If you want to exit, type Bye!")
while(flag==True):
  user_response = input()
  user_response=user_response.lower()
  if(user_response!='bye'):
    if(user_response=='thanks' or user_response=='thank you' ):
      flag=False
      print("ROBO: You are welcome..")
   else:
      if(greeting(user_response)!=None):
        print("ROBO: "+greeting(user_response))
      else:
        print("ROBO: ",end="")
        print(response(user_response))
        sent_tokens.remove(user_response)
  else:
    flag=False

print("ROBO: Bye! take care..")
```

Quellen

https://medium.com/analytics-vidhya/building-a-simple-chatbot-in-python-using-nltk-7c8c8215ac6e

https://github.com/parulnith/Building-a-Simple-Chatbot-in-Python-using-NLTK/blob/master/Chatbot.ipynb

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html