Драме Амди Мустафа

С.б: 1032189097

Группа: НФИмд-01-19

Создание бота

В этом проекте Python мы построим chatbot с использованием методов глубокого обучения. Chatbot будет обучен набору данных, который содержит категории (intents. json), модель и ответы. Мы используем специальную повторяющуюся нейронную сеть (LSTM) для классификации категории, к которой относится сообщение пользователя, а затем дадим случайный ответ из списка ответов.

Давайте создадим чатбот на основе восстановления с использованием NLTK, Keras, Python и т. д.

В этом проекте я использую командную строку anaconda с виртуальной средой (tf-gpu), чтобы не иметь проблемы с версиями библиотек и модулей.

Начнем с установки с помощью следующих команд

```
conda create -n tf-gpu tensorflow-gpu
```

эта строка позволяет нам установить tensorflow в виртуальной среде (tf-gpu), которую нам нужно активировать с помощью следующей команды :

```
conda activate
```

1. Импорт и загрузка файла данных

Сначала мы создаем файл с именем *'intents.json'*, который содержит набор данных, который мы будем использовать. Это файл JSON, который содержит шаблоны, которые нам нужно найти, и ответы, которые мы хотим вернуть пользователю.

После этого мы затем создаем имя файла python, например *train_bot21.py*. Мы импортируем необходимые пакеты для нашего chatbot и инициализируем переменные, которые мы будем использовать в нашем проекте Python. Поэтому мы использовали пакет json для разбора файла JSON в python.

```
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import json
import pickle

import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import SGD
import random
```

Файл *intents.json*, который мы будем использовать, выглядит следующим образом :

```
{"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",
        "patterns": ["How you could help me?", "What you can do?", "What help you provide?", "How you can be helpful?", "What support is
        "responses": ["I can guide you through Adverse drug reaction list, Blood pressure tracking, Hospitals and Pharmacies", "Offering
        "context": [""]
       },
       {"tag": "adverse_drug",
        "patterns": ["How to check Adverse drug reaction?", "Open adverse drugs module", "Give me a list of drugs causing adverse behavi
        "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 p
```


2. Данные предварительной обработки

В нашем файле python train_bot21.py, мы сделаем притворство.

```
words=[]
classes = []
documents = []
ignore_words = ['?', '!']
data_file = open('intents.json').read()
intents = json.loads(data_file)
```

Когда мы работаем с текстовыми данными, нам нужно выполнить различные предварительные обработки данных, прежде чем создавать модель машинного обучения или глубокого обучения. Токенизация - это первое, что мы можем сделать с текстовыми данными. Tokenizing-это процесс разбиения всего текста на небольшие части, такие как слова.

Здесь мы просматриваем шаблоны и бросаем предложение с помощью функции nltk.word_tokenize ()и добавляем каждое слово в список слов. Мы также создаем список классов для наших тегов.

```
for intent in intents['intents']:
    for pattern in intent['patterns']:

        #tokenize each word
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        #add documents in the corpus
        documents.append((w, intent['tag']))

        # add to our classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])

```

Теперь давайте лемматизировать каждое слово и удалить дубликаты слов из списка. Лемматизация-это процесс преобразования слова в его форму леммы, а затем создания файла рассола для хранения объектов Python, которые мы будем использовать при прогнозировании.

```
# lemmaztize and lower each word and remove duplicates
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))
# sort classes
classes = sorted(list(set(classes)))
# documents = combination between patterns and intents
print (len(documents), "documents")
# classes = intents
print (len(classes), "classes", classes)
# words = all words, vocabulary
print (len(words), "unique lemmatized words", words)


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

```

3. Создание данных обучения и тестирования

Теперь мы создадим обучающие данные, в которых мы будем предоставлять вход и выход. Наш вклад будет модель и выход будет класс нашей модели входа принадлежит. Но компьютер не понимает текст, поэтому мы преобразуем текст в числа.

```
53
54  # create our training data
55  training = []
56  # create an empty array for our output
57  output_empty = [0] * len(classes)
58  # training set, bag of words for each sentence
59  for doc in documents:
60      # initialize our bag of words
61      bag = []
62      # list of tokenized words for the pattern
63      pattern_words = doc[0]
64      # lemmatize each word - create base word, in attempt to represent related words
65      pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]
66      # create our bag of words array with 1, if word match found in current pattern
67      for w in words:
68          bag.append(1) if w in pattern_words else bag.append(0)
69
70      # output is a '0' for each tag and '1' for current tag (for each pattern)
71      output_row = list(output_empty)
72      output_row[classes.index(doc[1])] = 1
73
74      training.append([bag, output_row])
75  # shuffle our features and turn into np.array
76  random.shuffle(training)
77  training = np.array(training)
78  # create train and test lists. X - patterns, Y - intents
79  train_x = list(training[:,0])
80  train_y = list(training[:,1])
81  print("Training data created")
82
```

4. Построить модель

У нас есть наши тренировочные данные, теперь мы построим глубокую нейронную сеть, которая имеет 3 слоя. Для этого мы

используем последовательный API Keras. Обучив модель в течение 200 эпох, мы достигли 100% точности на нашей модели. Сохраните модель как ' Amdy_chatbot_model.h5’.

```
83
84  # Create model - 3 layers. First layer 128 neurons, second layer 64 neuro
85  # equal to number of intents to predict output intent with softmax
86  model = Sequential()
87  model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
88  model.add(Dropout(0.5))
89  model.add(Dense(64, activation='relu'))
90  model.add(Dropout(0.5))
91  model.add(Dense(len(train_y[0]), activation='softmax'))
92
93  # Compile model. Stochastic gradient descent with Nesterov accelerated gr
94  sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
95  model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['a
96
97  #fitting and saving the model
98  hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_
99  model.save('Amdy_chatbot_model.h5', hist)
100
101 print("Your model has been created successfully !")
102
```

5. Предсказать ответ (графический интерфейс пользователя)

Теперь, чтобы предсказать фразы и получить ответ от пользователя, чтобы мы могли создать новый файл ‘bot21.py".

Мы загрузим обученную модель, а затем используем графический пользовательский интерфейс, который будет предсказывать ответ бота. Модель скажет нам только класс, к которому он принадлежит, поэтому мы будем реализовывать некоторые функции, которые

будут определять класс, а затем мы получим случайный ответ из списка ответов.

Опять же, мы импортируем необходимые пакеты и загружаем файлы pickle ' words. pkl’ and ‘classes. pkl’, которые мы создали, когда мы сформировали нашу модель:

bot21.py

```
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('Amdy_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'))
14
```

Чтобы предсказать класс, нам нужно будет предоставить обратную связь так же, как мы это делали во время обучения. Поэтому мы создадим функции, которые будут выполнять предварительную обработку текста, а затем предсказать класс.

```python
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
```

После предсказания класса мы получим случайный ответ из списка намерений.

```python

def getResponse(ints, intents_json):
    tag = ints[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

def chatbot_response(msg):
    ints = predict_class(msg, model)
    res = getResponse(ints, intents)
    return res

```

Теперь давайте закодировать графический пользовательский интерфейс. Для этого мы используем библиотеку Tkinter, которая уже поставляется в python. Мы возьмем сообщение ввода пользователя, а затем использовать вспомогательные функции, которые мы создали, чтобы получить ответ от бота и отобразить его на графическом интерфейсе.

```
#Creating GUI with tkinter
import tkinter
from tkinter import *


def send():
    msg = EntryBox.get("1.0",'end-1c').strip()
    EntryBox.delete("0.0",END)

    if msg != '':
        ChatLog.config(state=NORMAL)
        ChatLog.insert(END, "You: " + msg + '\n\n')
        ChatLog.config(foreground="#442265", font=("Verdana", 12 ))

        res = chatbot_response(msg)
        ChatLog.insert(END, "Bot: " + res + '\n\n')

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


base = Tk()
base.title("Hello")
base.geometry("400x500")
base.resizable(width=FALSE, height=FALSE)

#Create Chat window
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
```

```
101
102  #Create Button to send message
103  SendButton = Button(base, font=("Verdana",12,'bold'), text="Send", width="12", height=5,
104                      bd=0, bg="#32de97", activebackground="#3c9d9b",fg='#ffffff',
105                      command= send )
106
107  #Create the box to enter message
108  EntryBox = Text(base, bd=0, bg="white",width="29", height="5", font="Arial")
109  #EntryBox.bind("<Return>", send)
110
111
112  #Place all components on the screen
113  scrollbar.place(x=376,y=6, height=386)
114  ChatLog.place(x=6,y=6, height=386, width=370)
115  EntryBox.place(x=128, y=401, height=90, width=265)
116  SendButton.place(x=6, y=401, height=90)
117
118  base.mainloop()
119
```

6. Выполняем chatbot

Для запуска chatbot у нас есть два основных файла; train_bot21.py и bot21.py.

Во-первых, мы тренируем модель с помощью команды в терминале :

```
Python train_bot21.py
```

Затем, чтобы запустить приложение, мы запускаем второй файл.

```
Python bot21.py
```

Давайте объясним файлы нашего проекта

- **Intents. json**-файл данных, который имеет предопределенные шаблоны и ответы.
- **train_bot21.py** – в этом файле Python мы написали скрипт для создания модели и обучения нашему чатботу.
- **Words.pkl** – это файл pickle, в котором мы храним слова объект Python, который содержит список нашего словаря.
- **classes.pkl** – файл классов рассол содержит список категорий.
- **Amdy_Chatbot_model.h5** – это тренированная модель, которая содержит информацию о модели и имеет веса нейронов.
- **bot21.py** -это скрипт Python, в котором мы реализовали графический интерфейс для нашего chatbot. Пользователи могут легко взаимодействовать с ботом.