chatbot

June 19, 2024

[ ]:

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

# Load and preprocess data
text = "Today is a good day. We are in the lab"
tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])
total_words = len(tokenizer.word_index) + 1

input_sequences = []
for line in text.split('.'):
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)

# Pad sequences and create predictors and label
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = pad_sequences(input_sequences, maxlen=max_sequence_len,␣
  ↪padding='pre')
predictors, label = input_sequences[:,:-1], input_sequences[:,-1]
label = tf.keras.utils.to_categorical(label, num_classes=total_words)

# Define the model
model = Sequential([
    Embedding(total_words, 100, input_length=max_sequence_len-1),
    LSTM(150),
    Dense(total_words, activation='softmax')
])

# Compile and train the model
model.compile(loss='categorical_crossentropy', optimizer='adam',␣
  ↪metrics=['accuracy'])
model.fit(predictors, label, epochs=100, verbose=1)
```

```
Epoch 1/100
1/1 [==============================] - 3s 3s/step - loss: 2.3964 - accuracy:
0.1250
Epoch 2/100
1/1 [==============================] - 0s 18ms/step - loss: 2.3861 - accuracy:
0.5000
Epoch 3/100
1/1 [==============================] - 0s 17ms/step - loss: 2.3757 - accuracy:
0.5000
Epoch 4/100
1/1 [==============================] - 0s 18ms/step - loss: 2.3650 - accuracy:
0.6250
Epoch 5/100
1/1 [==============================] - 0s 18ms/step - loss: 2.3538 - accuracy:
0.6250
Epoch 6/100
1/1 [==============================] - 0s 16ms/step - loss: 2.3421 - accuracy:
0.6250
Epoch 7/100
1/1 [==============================] - 0s 18ms/step - loss: 2.3296 - accuracy:
0.6250
Epoch 8/100
1/1 [==============================] - 0s 19ms/step - loss: 2.3162 - accuracy:
0.6250
Epoch 9/100
1/1 [==============================] - 0s 16ms/step - loss: 2.3018 - accuracy:
0.7500
Epoch 10/100
1/1 [==============================] - 0s 17ms/step - loss: 2.2860 - accuracy:
0.7500
Epoch 11/100
1/1 [==============================] - 0s 16ms/step - loss: 2.2688 - accuracy:
0.7500
Epoch 12/100
1/1 [==============================] - 0s 22ms/step - loss: 2.2498 - accuracy:
0.7500
Epoch 13/100
1/1 [==============================] - 0s 23ms/step - loss: 2.2289 - accuracy:
0.7500
Epoch 14/100
1/1 [==============================] - 0s 18ms/step - loss: 2.2059 - accuracy:
0.7500
Epoch 15/100
1/1 [==============================] - 0s 19ms/step - loss: 2.1803 - accuracy:
0.7500
Epoch 16/100
1/1 [==============================] - 0s 18ms/step - loss: 2.1521 - accuracy:
0.7500
```

```
Epoch 17/100
1/1 [==============================] - 0s 18ms/step - loss: 2.1207 - accuracy:
0.7500
Epoch 18/100
1/1 [==============================] - 0s 18ms/step - loss: 2.0861 - accuracy:
0.7500
Epoch 19/100
1/1 [==============================] - 0s 18ms/step - loss: 2.0479 - accuracy:
0.7500
Epoch 20/100
1/1 [==============================] - 0s 19ms/step - loss: 2.0059 - accuracy:
0.7500
Epoch 21/100
1/1 [==============================] - 0s 19ms/step - loss: 1.9600 - accuracy:
0.7500
Epoch 22/100
1/1 [==============================] - 0s 19ms/step - loss: 1.9103 - accuracy:
0.6250
Epoch 23/100
1/1 [==============================] - 0s 17ms/step - loss: 1.8570 - accuracy:
0.6250
Epoch 24/100
1/1 [==============================] - 0s 20ms/step - loss: 1.8006 - accuracy:
0.5000
Epoch 25/100
1/1 [==============================] - 0s 19ms/step - loss: 1.7416 - accuracy:
0.5000
Epoch 26/100
1/1 [==============================] - 0s 22ms/step - loss: 1.6804 - accuracy:
0.5000
Epoch 27/100
1/1 [==============================] - 0s 20ms/step - loss: 1.6174 - accuracy:
0.5000
Epoch 28/100
1/1 [==============================] - 0s 20ms/step - loss: 1.5520 - accuracy:
0.6250
Epoch 29/100
1/1 [==============================] - 0s 17ms/step - loss: 1.4835 - accuracy:
0.6250
Epoch 30/100
1/1 [==============================] - 0s 24ms/step - loss: 1.4112 - accuracy:
0.6250
Epoch 31/100
1/1 [==============================] - 0s 18ms/step - loss: 1.3351 - accuracy:
0.7500
Epoch 32/100
1/1 [==============================] - 0s 19ms/step - loss: 1.2561 - accuracy:
0.8750
```

```
Epoch 33/100
1/1 [==============================] - 0s 18ms/step - loss: 1.1760 - accuracy:
0.8750
Epoch 34/100
1/1 [==============================] - 0s 18ms/step - loss: 1.0967 - accuracy:
1.0000
Epoch 35/100
1/1 [==============================] - 0s 18ms/step - loss: 1.0196 - accuracy:
1.0000
Epoch 36/100
1/1 [==============================] - 0s 17ms/step - loss: 0.9451 - accuracy:
1.0000
Epoch 37/100
1/1 [==============================] - 0s 18ms/step - loss: 0.8733 - accuracy:
1.0000
Epoch 38/100
1/1 [==============================] - 0s 18ms/step - loss: 0.8042 - accuracy:
1.0000
Epoch 39/100
1/1 [==============================] - 0s 19ms/step - loss: 0.7382 - accuracy:
1.0000
Epoch 40/100
1/1 [==============================] - 0s 19ms/step - loss: 0.6759 - accuracy:
1.0000
Epoch 41/100
1/1 [==============================] - 0s 19ms/step - loss: 0.6185 - accuracy:
1.0000
Epoch 42/100
1/1 [==============================] - 0s 20ms/step - loss: 0.5663 - accuracy:
1.0000
Epoch 43/100
1/1 [==============================] - 0s 20ms/step - loss: 0.5181 - accuracy:
1.0000
Epoch 44/100
1/1 [==============================] - 0s 19ms/step - loss: 0.4721 - accuracy:
1.0000
Epoch 45/100
1/1 [==============================] - 0s 20ms/step - loss: 0.4280 - accuracy:
1.0000
Epoch 46/100
1/1 [==============================] - 0s 18ms/step - loss: 0.3871 - accuracy:
1.0000
Epoch 47/100
1/1 [==============================] - 0s 17ms/step - loss: 0.3506 - accuracy:
1.0000
Epoch 48/100
1/1 [==============================] - 0s 18ms/step - loss: 0.3183 - accuracy:
1.0000
```

```
Epoch 49/100
1/1 [==============================] - 0s 19ms/step - loss: 0.2884 - accuracy:
1.0000
Epoch 50/100
1/1 [==============================] - 0s 20ms/step - loss: 0.2599 - accuracy:
1.0000
Epoch 51/100
1/1 [==============================] - 0s 18ms/step - loss: 0.2337 - accuracy:
1.0000
Epoch 52/100
1/1 [==============================] - 0s 17ms/step - loss: 0.2109 - accuracy:
1.0000
Epoch 53/100
1/1 [==============================] - 0s 16ms/step - loss: 0.1908 - accuracy:
1.0000
Epoch 54/100
1/1 [==============================] - 0s 19ms/step - loss: 0.1725 - accuracy:
1.0000
Epoch 55/100
1/1 [==============================] - 0s 20ms/step - loss: 0.1562 - accuracy:
1.0000
Epoch 56/100
1/1 [==============================] - 0s 22ms/step - loss: 0.1421 - accuracy:
1.0000
Epoch 57/100
1/1 [==============================] - 0s 23ms/step - loss: 0.1301 - accuracy:
1.0000
Epoch 58/100
1/1 [==============================] - 0s 17ms/step - loss: 0.1197 - accuracy:
1.0000
Epoch 59/100
1/1 [==============================] - 0s 17ms/step - loss: 0.1100 - accuracy:
1.0000
Epoch 60/100
1/1 [==============================] - 0s 19ms/step - loss: 0.1006 - accuracy:
1.0000
Epoch 61/100
1/1 [==============================] - 0s 16ms/step - loss: 0.0922 - accuracy:
1.0000
Epoch 62/100
1/1 [==============================] - 0s 17ms/step - loss: 0.0846 - accuracy:
1.0000
Epoch 63/100
1/1 [==============================] - 0s 19ms/step - loss: 0.0774 - accuracy:
1.0000
Epoch 64/100
1/1 [==============================] - 0s 18ms/step - loss: 0.0706 - accuracy:
1.0000
```

```
Epoch 65/100
1/1 [==============================] - 0s 19ms/step - loss: 0.0644 - accuracy:
1.0000
Epoch 66/100
1/1 [==============================] - 0s 17ms/step - loss: 0.0588 - accuracy:
1.0000
Epoch 67/100
1/1 [==============================] - 0s 17ms/step - loss: 0.0540 - accuracy:
1.0000
Epoch 68/100
1/1 [==============================] - 0s 22ms/step - loss: 0.0498 - accuracy:
1.0000
Epoch 69/100
1/1 [==============================] - 0s 19ms/step - loss: 0.0462 - accuracy:
1.0000
Epoch 70/100
1/1 [==============================] - 0s 17ms/step - loss: 0.0429 - accuracy:
1.0000
Epoch 71/100
1/1 [==============================] - 0s 20ms/step - loss: 0.0400 - accuracy:
1.0000
Epoch 72/100
1/1 [==============================] - 0s 18ms/step - loss: 0.0375 - accuracy:
1.0000
Epoch 73/100
1/1 [==============================] - 0s 18ms/step - loss: 0.0352 - accuracy:
1.0000
Epoch 74/100
1/1 [==============================] - 0s 16ms/step - loss: 0.0331 - accuracy:
1.0000
Epoch 75/100
1/1 [==============================] - 0s 18ms/step - loss: 0.0311 - accuracy:
1.0000
Epoch 76/100
1/1 [==============================] - 0s 16ms/step - loss: 0.0293 - accuracy:
1.0000
Epoch 77/100
1/1 [==============================] - 0s 17ms/step - loss: 0.0276 - accuracy:
1.0000
Epoch 78/100
1/1 [==============================] - 0s 19ms/step - loss: 0.0260 - accuracy:
1.0000
Epoch 79/100
1/1 [==============================] - 0s 17ms/step - loss: 0.0245 - accuracy:
1.0000
Epoch 80/100
1/1 [==============================] - 0s 18ms/step - loss: 0.0232 - accuracy:
1.0000
```

```
Epoch 81/100
1/1 [==============================] - 0s 17ms/step - loss: 0.0220 - accuracy:
1.0000
Epoch 82/100
1/1 [==============================] - 0s 18ms/step - loss: 0.0209 - accuracy:
1.0000
Epoch 83/100
1/1 [==============================] - 0s 20ms/step - loss: 0.0199 - accuracy:
1.0000
Epoch 84/100
1/1 [==============================] - 0s 17ms/step - loss: 0.0190 - accuracy:
1.0000
Epoch 85/100
1/1 [==============================] - 0s 18ms/step - loss: 0.0181 - accuracy:
1.0000
Epoch 86/100
1/1 [==============================] - 0s 18ms/step - loss: 0.0173 - accuracy:
1.0000
Epoch 87/100
1/1 [==============================] - 0s 19ms/step - loss: 0.0166 - accuracy:
1.0000
Epoch 88/100
1/1 [==============================] - 0s 17ms/step - loss: 0.0160 - accuracy:
1.0000
Epoch 89/100
1/1 [==============================] - 0s 17ms/step - loss: 0.0154 - accuracy:
1.0000
Epoch 90/100
1/1 [==============================] - 0s 17ms/step - loss: 0.0148 - accuracy:
1.0000
Epoch 91/100
1/1 [==============================] - 0s 18ms/step - loss: 0.0143 - accuracy:
1.0000
Epoch 92/100
1/1 [==============================] - 0s 17ms/step - loss: 0.0139 - accuracy:
1.0000
Epoch 93/100
1/1 [==============================] - 0s 20ms/step - loss: 0.0134 - accuracy:
1.0000
Epoch 94/100
1/1 [==============================] - 0s 20ms/step - loss: 0.0130 - accuracy:
1.0000
Epoch 95/100
1/1 [==============================] - 0s 25ms/step - loss: 0.0126 - accuracy:
1.0000
Epoch 96/100
1/1 [==============================] - 0s 19ms/step - loss: 0.0122 - accuracy:
1.0000
```

```
Epoch 97/100
1/1 [==============================] - 0s 20ms/step - loss: 0.0119 - accuracy:
1.0000
Epoch 98/100
1/1 [==============================] - 0s 17ms/step - loss: 0.0115 - accuracy:
1.0000
Epoch 99/100
1/1 [==============================] - 0s 16ms/step - loss: 0.0112 - accuracy:
1.0000
Epoch 100/100
1/1 [==============================] - 0s 16ms/step - loss: 0.0109 - accuracy:
1.0000
```

[ ]:

```
<keras.src.callbacks.History at 0x79751ff17940>
```

[ ]:

```python
import numpy as np
# Text generation
def generate_text(seed_text, next_words):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_sequence_len-1,␣
  ↪padding='pre')
        predicted = model.predict(token_list, verbose=0)
        predicted_word = tokenizer.index_word[np.argmax(predicted)]
        seed_text += " " + predicted_word
    return seed_text

print(generate_text("Your seed text", 10))
```

```
Your seed text is is a good day day day day lab lab
```

[ ]:

```python
link_to_dataset = "https://www.kaggle.com/datasets/narendrageek/
  ↪mental-health-faq-for-chatbot"
```

install dependencies

[ ]:

```
%pip install -r '/content/requirements.txt'
```

```
ERROR: Could not open requirements file: [Errno 2] No such file or
directory: '/content/requirements.txt'
```

[ ]:

```python
import nltk

nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

[ ]: True

[ ]:
```python
import pandas as pd
import nltk
import numpy as np
import re

from nltk.stem import wordnet                                # to perform␣
 ↪lemmitization
from sklearn.feature_extraction.text import CountVectorizer    # to perform bow
from sklearn.feature_extraction.text import TfidfVectorizer    # to perform␣
 ↪tfidf
from nltk import pos_tag                                     # for parts of␣
 ↪speech
from sklearn.metrics import pairwise_distances               # to perfrom␣
 ↪cosine similarity
from nltk import word_tokenize                               # to create␣
 ↪tokens
from nltk.corpus import stopwords                            # for stop words
```

[ ]:
```
!unzip /content/mental-health-faq.zip -d /content/
```

```
Archive:  /content/mental-health-faq.zip
  inflating: /content/Mental_Health_FAQ.csv
```

[ ]:
```python
path = '/content/Mental_Health_FAQ.csv'
```

[ ]:
```python
df = pd.read_csv(path)
df.head()
```

[ ]:
```
   Question_ID                                          Questions  \
0      1590140        What does it mean to have a mental illness?
1      2110618                       Who does mental illness affect?
2      6361820                           What causes mental illness?
3      9434130  What are some of the warning signs of mental i…
4      7657263             Can people with mental illness recover?

                                             Answers
0  Mental illnesses are health conditions that di…
1  It is estimated that mental illness affects 1 …
2  It is estimated that mental illness affects 1 …
```

```
3  Symptoms of mental health disorders vary depen…
4  When healing from mental illness, early identi…
```

[ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df[['Questions']],␣
 ↪df[['Answers']], random_state=42)
```

[ ]:
```
X_train.shape
```

[ ]:
```
(73, 1)
```

[ ]:
```
X_test.shape
```

[ ]:
```
(25, 1)
```

[ ]:
```
nltk.download('wordnet')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[ ]:
```
True
```

[ ]:
```
s = 'I feel sad a lot. Is that normal?'
```

[ ]:
```
lemma = wordnet.WordNetLemmatizer()
lemma.lemmatize("absorbed", pos = 'v')
```

[ ]:
```
'absorb'
```

[ ]:
```
pos_tag(nltk.word_tokenize(s),tagset = None)
```

[ ]:
```
stop = stopwords.words('english')
stop
```

[ ]:
```
def text_normalization(text):
    text = str(text).lower()

    spl_char_text = re.sub(r'[^ a-z]','',text)

    tokens = nltk.word_tokenize(spl_char_text)

    lema = wordnet.WordNetLemmatizer()

    tags_list = pos_tag(tokens,tagset=None)
    lema_words = []

    for token,pos_token in tags_list:
        if pos_token.startswith('V'):                   # verb
            pos_val = 'v'
        elif pos_token.startswith('J'):                 # adjective
            pos_val = 'a'
        elif pos_token.startswith('R'):                 # adverb
            pos_val = 'r'
        else:
            pos_val = 'n'                               # noun
        lema_token = lema.lemmatize(token,pos_val)

        if lema_token in stop:
          lema_words.append(lema_token)                 # appending the lemmatized␣
↪token into a list

    return " ".join(lema_words)
```

[ ]:

```
text_normalization("There is something i want to tell you")
```

[ ]:

```
df['lemmatized_text'] = df['Questions'].apply(text_normalization)
```

[ ]:

```
df.head()
```

[ ]:

```
cv = CountVectorizer()
X = cv.fit_transform(df['lemmatized_text']).toarray()
```

[ ]:

```
features = cv.get_feature_names_out()
df_bow = pd.DataFrame(X, columns = features)
df_bow.head()
```

[ ]:

```
Question = 'What treatment options are available?'
Question_lemma = text_normalization(Question)
Question_bow = cv.transform([Question_lemma]).toarray()
```

[ ]:

```
Question_lemma
```

[ ]:

```
Question_bow
```

[ ]:

[ ]:

[ ]:

```
cosine_value = 1- pairwise_distances(df_bow, Question_bow, metric = 'cosine' )
(cosine_value)
```

[ ]:
```
df['similarity_bow'] = cosine_value
```

[ ]:
```
df.head()
```

[ ]:
```
df.sort_values(by = 'similarity_bow', ascending=False).head()  # sorting the␣
 ↪values
```

[ ]:
```
df.sort_values(by = 'similarity_bow', ascending=False).tail()
```

[ ]:
```
threshold = 0.1 #considering the value of smiliarity to be greater than 0.1
df[df['similarity_bow'] > threshold]
```

[ ]:
```

```

[ ]:
```

```

[ ]:
```
Question1 = 'What treatment options are available'
```

[ ]:
```
tfidf = TfidfVectorizer()
x_tfidf = tfidf.fit_transform(df['lemmatized_text']).toarray()
```

[ ]:
```
Question_lemma1 = text_normalization(Question1)
Question_tfidf = tfidf.transform([Question_lemma1]).toarray()
```

[ ]:
```
Question_lemma1
```

[ ]:
```
Question_tfidf
```

[ ]:
```
df_tfidf = pd.DataFrame(x_tfidf,columns = tfidf.get_feature_names_out())
df_tfidf.head()
```

[ ]:
```
cos = 1-pairwise_distances(df_tfidf,Question_tfidf,metric='cosine')
```

[ ]:
```
df['similarity_tfidf'] = cos                                                ␣
 ↪# creating a new column
df_simi_tfidf = pd.DataFrame(df, columns=['Answers','similarity_tfidf'])    ␣
 ↪# taking similarity value of responses for the question we took
df_simi_tfidf
```

[ ]:
```
df_simi_tfidf.sort_values(by='similarity_tfidf', ascending=False).head(10)
```

[ ]:
```
threshold = 0.1
df_simi_tfidf[df_simi_tfidf['similarity_tfidf'] > threshold]
```

[ ]:
```
df['Answers'].loc[6]
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

```python
def chat_bow(text): #chatbot based on Bag-Of-Words Model
    lemma = text_normalization(text)
    bow = cv.transform([lemma]).toarray()
    cosine_value = 1- pairwise_distances(df_bow,bow, metric = 'cosine' )
    index_value = cosine_value.argmax()

    return df['Answers'].loc[index_value]

def chat_tfidf(text): #chatbot based on TF-IDF model
    lemma = text_normalization(text)
    tf = tfidf.transform([lemma]).toarray()
    cos = 1-pairwise_distances(df_tfidf,tf,metric='cosine')
    index_value = cos.argmax()
    return df['Answers'].loc[index_value]
```

[ ]:

```python
chat_bow('can you prevent mental health problems')
```

[ ]:

```
'We can all suffer from mental health challenges, but developing our wellbeing,
resilience, and seeking help early can help prevent challenges becoming
serious.'
```

[ ]:

```python
chat_bow('what is mental health')
```

[ ]:

```
'Just as there are different types of medications for physical illness,
different treatment options are available for individuals with mental illness.
Treatment works differently for different people. It is important to find what
works best for you or your child.'
```

[ ]:

```python
chat_tfidf('how do i see a counsellor')
```

[ ]:

```
'If your beliefs , thoughts , feelings or behaviours have a significant impact
on your ability to function in what might be considered a normal or ordinary
way, it would be important to seek help.'
```

[ ]:

```python
print(chat_tfidf('how to find a support group'))
```

```
Distraction is a very valid tool to help you cope when everything feels
overwhelming or when you feel lonely or isolated.
 If you don't have a lot of energy or focus right now, try low-effort
```

```
distractions like watching TV, browsing Youtube, listening to a podcast or
audiobook, playing a game on your phone, reading an easy book or magazine, or
working on a simple art project.
 If you have more energy and focus, give yourself a to-do list every day: you
can clean and take care of projects around your home, work on hobbies, connect
with family or friends, read a new book and catch up on your favourite TV shows.
You can find interesting opportunities to take online courses from universities
all over the world through MOOCs and other online learning platforms, you can
learn a new language online or through apps, and you can learn new hobbies and
activities. As more people have to practice social distancing or self-isolation,
people are finding creative ways to bring the world into their homes: you can
tour museums and art galleries, Skype with a scientist, watch animals at zoos
and nature preserves, and more.
 When normal schedules are disrupted, it's easy to fall into unhelpful habits.
Look for ways to keep yourself on track with healthier habits. You could set
yourself goals every day or turn activities into a fun competition with friends
or family-whoever takes the most language classes wins!
 Many communities are using social media platforms like Facebook to organize
support and help for neighbours. If you are healthy and it's safe to do so, you
can sign up to walk dogs, pick up groceries and household supplies, and help
others who can't go out at the moment. This can be a great way to make new
connections in your area, and helping others is good for your own mental health.
Just be sure to follow good hygiene practices and physical distancing-your own
health is important.
```

[ ]:

```python
chat_bow("what should i do if i am worried about a friend?")
```

[ ]:

```
'This may depend on your relationship with them. Gently encouraging someone to
seek appropriate support would be helpful to start with.'
```

[ ]:

```python
chat_bow("what are warning signs of depression mental health?")
```

[ ]:

```
'There are many types of mental health professionals. Finding the right one for
you may require some research.'
```

[ ]:

[ ]: