fhirfly /
notebooks

Code Issues Pull requests Actions Projects Security Insights

notebooks / PICO Pubmed Plus GPT3.5 Evidence Based Medicine.ipynb

fhirfly small delint 6 months ago

529 lines (529 loc) · 13.9 KB

Preview Code Blame Raw

PyPubMedGPT - Create Evidence Based Medicine with Pub Med using GPT and PICO Prompts

We start with a clinical question. We aren't too concerned with the format of the question, because we will convert it to PICO later

In [ ]:
```
simple_clincal_question = "What is the best treatment for a patient with a fractured tibia?"
```

Install Libraries

We need to install all of the Python Libraries that this Notebook needs to work.

In [ ]:
```
#First, install all of the requirements
!pip install requests
!pip install biopython
!pip install openai
!pip install transformers
!pip install numpy
```

Open AI API Key. You must get a key from [https://platform.openai.com/] and set it below

In [ ]:
```
import openai
openai.api_key = ""
```

Configure your email so that the NCBI service knows who you are

In [ ]:
```
from Bio import Entrez
from Bio import Entrez
Entrez.email = ""  # Always tell NCBI who you are
```

Define a chat function for ChatGPT completion, specfiy the 3.5 turbo mdel

In [ ]:

```
def chat(message):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "user", "content": f"{message}"},
        ],
        temperature=0.1
    )
    return response['choices'][0]['message']['content']
```

Rewrite the Clinical Question in PICO format

In [ ]:

```
pico_res = chat('rewrite the following clinical question according to the PICO model using (P), (I) , (C), (O) r
print(pico_res)
```

Parse the PICO question into its componements

In [ ]:

```
import re

# Regular expression to capture PICO components
pico_string = pico_res

# Refined regular expression to capture PICO components
pattern = r"In (?P<Patient>.*?) \(P\), (?P<Intervention>.*?) \(I\) (?P<Comparison>.*?) \(C\) (?P<Outcome>.*?) \

match = re.match(pattern, pico_string)

if match:
    pico_variables = match.groupdict()
else:
    pico_variables = "No match found!"
```

```
pico_variables
```

Search the Entrez Mesh Database for the meshed terms on our PICO Query

In [ ]:

```
query = ""
query_terms = ""
```

In [ ]:

```
idList = []
handle = Entrez.esearch(db="mesh", term=pico_variables['Patient'])
record = Entrez.read(handle)
handle.close()
mesh_terms = []
for translation in record['TranslationSet']:
    terms = translation['To'].split(' OR ')
    for term in terms:
        if '[MeSH Terms]' in term:
            mesh_terms.append(term.replace('[MeSH Terms]', '').replace('"', '').strip())
query_terms = [f"{term}" for term in mesh_terms]
query = " AND ".join(query_terms)
p_query = query
print(p_query)
```

In [ ]:

```
handle = Entrez.esearch(db="mesh", term=pico_variables['Intervention'])
record = Entrez.read(handle)
handle.close()
# Extract MeSH terms from the result
mesh_terms = []
for translation in record['TranslationSet']:
    terms = translation['To'].split(' OR ')
    for term in terms:
        if '[MeSH Terms]' in term:
            mesh_terms.append(term.replace('[MeSH Terms]', '').replace('"', '').strip())

query_terms = [f"{term}" for term in mesh_terms]
query = " OR ".join(query_terms)
i_query = query
print(i_query)
```

In [ ]:

```
handle = Entrez.esearch(db="mesh", term=pico_variables['Comparison'])
record = Entrez.read(handle)
handle.close()
mesh_terms = []
for translation in record['TranslationSet']:
    terms = translation['To'].split(' OR ')
    for term in terms:
        if '[MeSH Terms]' in term:
            mesh_terms.append(term.replace('[MeSH Terms]', '').replace('"', '').strip())
query_terms = [f"{term}" for term in mesh_terms]
query = " OR ".join(query_terms)
c_query = query
print(c_query)
```

In [ ]:

```
handle = Entrez.esearch(db="mesh", term=pico_variables['Outcome'])
record = Entrez.read(handle)
handle.close()
mesh_terms = []
for translation in record['TranslationSet']:
    terms = translation['To'].split(' OR ')
    for term in terms:
        if '[MeSH Terms]' in term:
            mesh_terms.append(term.replace('[MeSH Terms]', '').replace('"', '').strip())
query_terms = [f"{term}" for term in mesh_terms]
query = " OR ".join(query_terms)
o_query = query
print(o_query)
```

Construct the Final Query using the Mesh Terms

In [ ]:

```
final_query = f"({p_query}) AND ({i_query}) AND ({c_query}) AND ({o_query})"
print(final_query)
```

Query Pub Med using the Mesh Terms

In [ ]:

```
handle = Entrez.esearch(db="pubmed", term=final_query)
record = Entrez.read(handle)
handle.close()
handle.close()
idlist = record['IdList']
print(idlist)
print(record['Count'])
```

Fetch the Document Title and Abstract

In [ ]:

```
from Bio import Medline
handle = Entrez.efetch(db="pubmed", id=idlist, rettype="medline",retmode="text")
records = Medline.parse(handle)
records = list(records)
handle.close()
```

In [ ]:

```
articles = []

for record in records:
    title = record.get("TI", "?")
    author = record.get("AU", "?")
    journal = record.get("TA", "?")
    date_of_publication = record.get("DP", "?")
    abstract = record.get("AB", "?")
    keywords = record.get("OT", "?")
    mesh_terms =record.get("MH", "?")
    articles.append((title, abstract, journal, author, date_of_publication, keywords, mesh_terms))
```

Print a couple of the retrieved documents

In [ ]:

```
print(articles.__len__())
#print(articles)
```

Build a Vector Database with the Abstracts

In [ ]:

```
from transformers import BertTokenizer, BertModel
import torch

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained("bert-base-uncased")
model = BertModel.from_pretrained('bert-base-uncased')

def embed_text(text):
    if not text:
        return None  # or return a zero vector or another placeholder

    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True, max_length=512)
    with torch.no_grad():
        outputs = model(**inputs)
    return outputs['pooler_output'].numpy()


vectors = [embed_text(article[1]) for article in articles if article[1]]
vectors = [v for v in vectors if v is not None]
print(f"Number of vectors: {len(vectors)}")
```

In [ ]:

```
import faiss
import numpy as np

# Convert vectors list to a 2D numpy array
vectors_matrix = np.vstack(vectors)

# Build the index
index = faiss.IndexFlatL2(vectors_matrix.shape[1])
index.add(vectors_matrix)
```

Make a Vector of the PICO Query

In [ ]:

```
query_text = pico_res
query_vector = embed_text(query_text)
print(query_vector.shape)
print(pico_res)
```

Search the Vectorized Abstracts with the PICO Query

In [ ]:

```
# Define the number of nearest neighbors you want to retrieve
```