به نام خدا

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

درس پردازش زبان طبیعی

پاسخ تمرین ۵

نام و نام خانودگی: سامان اسلامی نظری

شماره دانشجویی: ۸۱۰۱۹۹۳۷۵

تیر ماه ۱۴۰۳

فهرست

سوال اول

بخش اول – دریافت و آماده‌سازی دادگان

ابتدا با استفاده از کتابخانه‌های Requests و Beautiful Soup صفحه مورد نظر را دانلود و لینک پی‌دی‌اف‌های هر فصل را استخراج کردیم. سپس با استفاده از `OnlinePDFLoader` در کتابخانه `LangChain` تمام لینک‌ها را دانلود و به فرمت مناسب ذخیره کردیم. در نهایت نیز لیست این داکیومنت‌ها را با استفاده از `RecursiveCharacterTextSplitter` به تکه‌های کوچک‌تر تبدیل کردیم.

Recursive Text Splitter چگونه عمل می‌کند؟

مدل‌های زبانی که ما برای پاسخ دادن به کاربر استفاده می‌کنیم Context Window با اندازه محدودی دارند. بنابراین هنگامی که کاربر از ما سوالی می‌پرسد، نمی توانیم کل یک داکیومنت را به عنوان Context به مدل زبانی بدهیم. ابتدا داکیومنت‌ها را به بخش‌های کوچک‌تر معنایی تبدیل می‌کنیم و سپس از این بخش‌های کوچک‌تر برای پاسخ دادن به کاربر استفاده می‌کنیم.

فلسفه Recursive Text Splitter این است که داکیومنت‌ها را ابتدا بر اساس پاراگراف‌ها، سپس جمله‌ها و در نهایت بر اساس بخش‌های کوچک‌تر تقسیم کند؛ این‌ها اجزایی از داکیومنت هستند که انسجام معنایی بیش‌تر دارند.

برای این کار، Recursive Text Splitter لیستی از کاراکترها دارد که بر اساس آن‌ها داکیومنت‌ها را تقسیم می‌کند. لیست پیشفرض آن `["\n\n", "\n", " ", ""]` است. ابتدا سعی می‌کند تا بر اساس کاراکتر اول داکیومنت را تقسیم کند؛ در صورتی که سایز داکیومنت همچنان بزرگ‌تر از chunk size مورد نظر بود،بر اساس کاراکتر بعدی در لیست مذکور داکیومنت را تقسیم می‌کند. Chunk size برابر تعداد کاراکترهای موجود در هر تقسیم می‌باشد.

تاثیر Chunk Size و Chunk Overlap

به طور کلی Chunk Size به تعداد کاراکترهای موجود در هر تکه اشاره می‌کند. Chunk Overlap نیز به تعداد کاراکترهایی که تکه‌های کنار هم می‌توانند به اشتراک بگذارند را نشان می‌دهد. در کل Chunk Size بزرگ‌تر به ما دید کلی‌تری از متن می‌دهد؛ برای تسک‌هایی که نیازمند فهم معنای کلی متن هستند، مثل ترجمه ماشینی یا خلاصه‌سازی متن، مقدار بزرگ‌تر Chunk Size می‌تواند تاثیر مثبتی داشته باشد؛ همچنین در مواردی که باید ارتباط معنایی بین بخش‌های مختلف متن را بدست آوریم، مثل تسک‌های پرسش و پاسخ یا در مواردی که می‌خواهیم یک متن جدید تولید کنیم نیز می‌تواند تاثیر مثبتی داشته باشد. در مواردی مثل پیدا کردن کلمات کلیدی، تشخیص

احساسات متن یا چک کردن غلط‌های املایی که نیازمند تمرکز بیش‌تر روی کلمات و بخش‌های کوچک‌تر متن هستند، مقدار کوچک‌تر Chunk Size کمک بیش‌تری می‌کند.

Chunk Overlap به ما کمک می‌کند تا ارتباط معنایی بین بخش‌های مختلف داکیومنت‌ها بیش‌تر حفظ شود؛ این موضوع باعث می‌شود که تقسیم‌سازی متن موجب از بین رفتن جریان معنایی نشود. علاوه بر آن این موضوع به ما کمک می‌کند تا بخش‌هایی که معانی نزدیک دارند در کنار هم باقی بمانند.

بخش دوم – تولید بازنمایی و پایگاه داده برداری

بخش سوم – پیاده‌سازی بازیاب ترکیبی (امتیازی)

`EnsembleRetriever` وظیفه اصلی در ترکیب کردن بازیاب‌های مختلف را دارد. لیستی از بازیاب‌های مختلف را به عنوان ورودی دریافت می‌کند؛ سپس نتیجه بازیاب‌ها را ترکیب کرده و یک پاسخ نهایی به کاربر ارائه می‌کند. این روش از هر نوع بازیاب تکی دیگری بهتر عمل می‌کند!

متداول‌ترین روش این است که یک بازیاب Dense را با یک بازیاب Sparse ترکیب کنیم. بازیاب‌های Dense تمرکز بیش‌تری روی معنای داکیومنت‌ها دارند در حالی که بازیاب‌های Sparse بر اساس کلمات کلیدی جستجو کرده و داکیومنت‌های مرتبط را برمی‌گردانند. در نهایت `EnsembleRetriever` بر اساس الگوریتم Reciprocal Rank Fusion پاسخ‌های هر بازیاب را رتبه‌بندی کرده و به کاربر خروجی می‌دهد.

تفاوت بین BM25 و FAISS

الگوریتم BM25 نوعی الگوریتم رتبه‌بندی است که به سه مورد اهمیت می‌دهد: اول اینکه چقدر کلمات کوئری در داکیومنت مورد هدف ظاهر می‌شوند؛ دوم طول داکیومنت و سوم میانگین طول داکیومنت‌ها در مجموعه داکیومنت‌هایی که در حال جستجو هستیم. این الگوریتم به نحوی با استفاده از TF-IDF و روش‌های دیگر سعی می‌کند تا داکیومنت‌ها را رتبه‌بندی کند.

به طور خلاصه FAISS از وکتورهای معنایی استفاده می‌کند. برای مثال در بخش‌های قبل از `HuggingFaceEmbeddings` استفاده کردیم تا داکیومنت‌ها را تبدیل به وکتور کنیم. سپس الگوریتم FAISS کوئری کاربر را دریافت کرده و با استفاده از روش‌هایی مثل L2 یا ضرب ماتریسی یا Cosine Similarity میزان ارتباط هر داکیومنت با کوئری دریافتی را محاسبه می‌کند. در نهایت داکیومنت‌هایی که بیش‌ترین ارتباط معنایی دارند را برمی‌گرداند.

الگوریتم BM25 نوعی بازیاب Lexical می‌باشد، در حالی که الگوریتم FAISS نوعی بازیاب Semantic است. در نوع اول داکیومنت‌ها بدون در نظر گرفتن معنای آن‌ها جستجو می‌شوند؛ به صورت Bag of Words و صرفا بر اساس جستجوی کلمات مشابه. با این حال در نوع دوم بازیابی، معنای کلمات و در کل ارتباط معنایی بین کلمات نیز در نظر گرفته می‌شوند. برای مثال کلمات مترادف یا نزدیک به هم و در کل ارتباط بین کلمات نیز در این روش در نظر گرفته می‌شوند.

نتیجه آزمایش بازیاب روی چند پرس و جوی مختلف

نتیجه سه پس و جوی متفاوت در زمینه‌های پردازش زبان طبیعی، علوم کامپیوتر و موضوعی کاملا نامرتبط در بخش Implementing Retrievers در نوتبوک قابل مشاهده است.

بخش چهارم – پیاده‌سازی Router Chain

در این جا زنجیری را پیاده‌سازی کردیم که وظیفه آن مسیریابی و انتخاب ابزار مناسب برای چت بات است. پرامپت زیر برای این منظور استفاده شد:

```
You must route user queries to one of three classes: VectorStore,
SearchEngine, or None.
If the user query is about Natural Language Processing and Speech Processing,
choose VectorStore.
If the query is something about computer science but it's not related to NLP,
SearchEngine.
If it's nothing about NLP or Computer Science, choose None.
Output only the chosen class. Do not output anything more than that.
{output_instruction}
query: {query}
```

در نهایت سه نوع پرس و جو در زمینه‌های متفاوت به این زنجیر داده شد که نتیجه آن به صورت زیر می‌باشد:

```python
router_chain = router_prompt | router_llm | router_parser

nlp_test_result = router_chain.invoke({
    "query": "How should I implement an LSTM model?",
    "output_instruction": router_parser.get_format_instructions()
})

cs_test_result = router_chain.invoke({
    "query": "What's a redd-black tree?",
    "output_instruction": router_parser.get_format_instructions()
})

other_test_result = router_chain.invoke({
    "query": "Who's the president of Congo?",
    "output_instruction": router_parser.get_format_instructions()
})

print(f"NLP: {nlp_test_result}, CS: {cs_test_result}, Other: {other_test_result}")
```

```
NLP: class_name='VectorStore', CS: class_name='SearchEngine', Other: class_name='None'
```

عکس 1 نتیجه اجرای پرس و جوهای متفاوت در زنجیر Router

علت مقدار صفر برای Temperature

این پارامتر در حین نمونه‌برداری از توزیع خروجی مدل روی ووکب استفاده می‌شود. مقادیر بیش‌تر از یک باعث نزدیک‌تر شدن احتمال هر کلمه به دیگری شده و در نهایت توزیع خروجی smooth-تر می‌شود. در این صورت کلماتی که انتخاب می‌شوند رندوم‌تر خواهند بود. در صورتی که از مقادیر کوچک‌تر از یک استفاده کنیم، احتمال هر کلمه در توزیع خروجی تفاوت بیش‌تری می‌کند، به طوری که محتمل‌ترین توزیع احتمال بیش‌تری از قبل به خود می‌گیرد. در صورتی که این مقدار را صفر انتخاب کنیم از روش greedy استفاده می‌کنیم. در این جا نیز از آنجایی که قصد تولید متن نداریم، randomness فاکتور مهمی نبوده و مهم‌ترین موضوع ممکن این است که درست‌ترین حالت را انتخاب کنیم تا قابل پارس کردن توسط `PydanticParser` باشد. بنابراین انتخاب روش greedy معقولانه و درست می‌باشد.

بخش پنجم – پیاده‌سازی Search Engine Chain

این زنجیر صرفا با استفاده از پلتفرم Tavily پرس و جوی کاربر را جستجو کرده و پنج سند را برمی‌گرداند. برای اینکه بتوانیم اسناد برگشتی توسط این پلتفرم را به نوع Document تغییر دهیم از تابع زیر در زنجیر استفاده کردیم:

```
@chain
def parse_search_engine(documents: list[dict[str, str]]) -> list[Document]:
    result_documents = [Document(
        page_content=doc['content'],
        metadata={'url': doc['url']}
    ) for doc in documents]

    return result_documents
```

بخش ششم – پیاده‌سازی Relevancy Check Chain

برای این بخش از پرامپت زیر استفاده کردیم:

```
You are provided with q user question and a document. If the given document
is relevant to the user question and can be used to answer it, output
'Relevant', and if not, output 'Irrelevant'. Only output the words Relevant
and Irrelevant in a JSON format as described in the output instructions.
User question: {user_query}
Document: {retrieved_document}
Output instruction: {output_instruction}
```

دلیل نیاز به این زنجیر عمدتا به این خاطر است که زنجیر retriever به هر حال سندهایی را باز می‌گرداند حتی اگر مرتبط نباشند. بنابراین برای اینکه بتوانیم این موضوع را کنترل کنیم و مثلا در پاسخ «چه کسی رئیس جمهور کنگو است؟» اسناد مرتبط با پردازش زبان طبیعی را به مدل زبانی ندهیم، نیاز به چنین زنجیری داریم.

بخش هفتم – پیاده‌سازی Fallback Chain

برای پیاده‌سازی این بخش از پرامپت زیر استفاده کردیم:

```
You are a friendly and kind teaching assistant. Your job is to provide
educational material related to NLP and Speech Recognition to the human user.
Do not respond to the queries that are outside the context of NLP and Speech
Recognition. If a query is not related acknowledge your limitations.

Current conversation:

{chat_history}


Human: {query}
```

باقی مراحل در نوتبوک قابل مشاهده می‌باشند.

بخش هشتم – پیاده‌سازی Generate with Context Chain

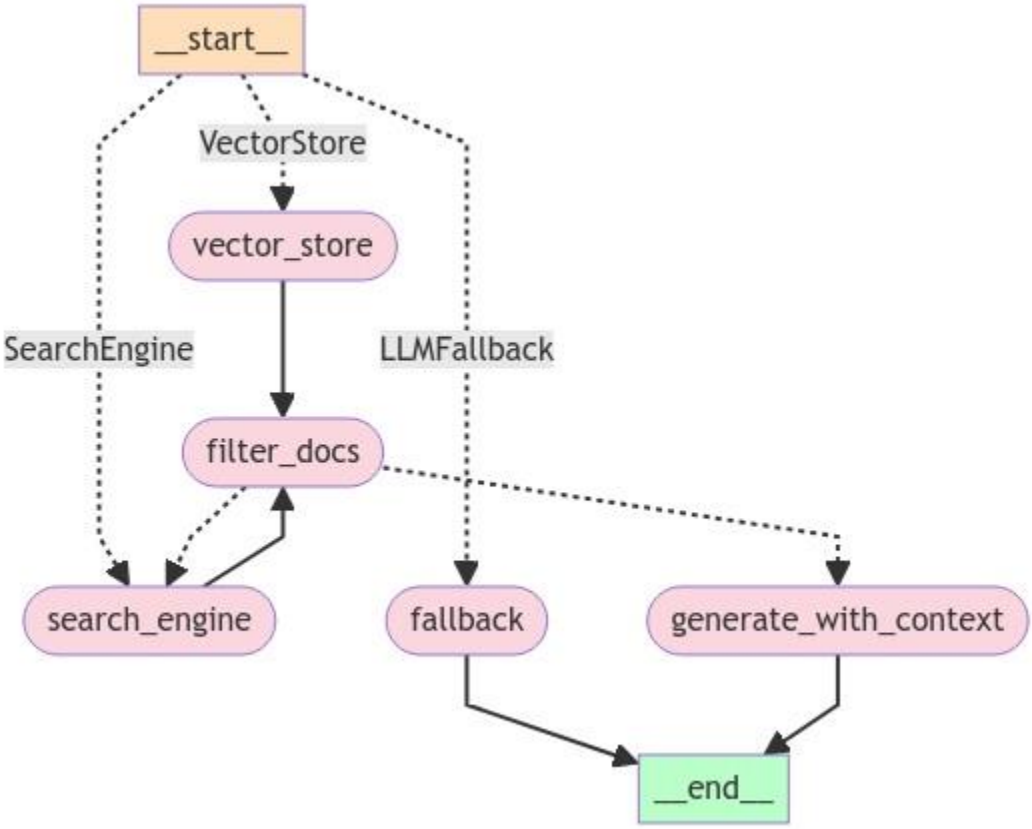
پرامپت استفاده‌شده در این بخش به صورت زیر می‌باشد:

```
You are a helpful assistant. Answer the query below based only on the
provided context. If the given context is not relevant, DO NOT answer based
on your own knowledge.


Context: {documents}


Query: {query}
```

باقی مراحل در نوتبوک قابل مشاهده می‌باشند.

بخش نهم – آماده‌سازی گراف با استفاده از LangGraph

گراف نهایی به صورت زیر می‌باشد:



عکس 2 گراف نهایی چت بات

گره‌ها و یال‌هایی که در آن به کار برده شدند به صورت زیر می‌باشند:

```
workflow = StateGraph(BotState)

workflow.add_node('vector_store', vector_store_node)
workflow.add_node('search_engine', search_engine_node)
workflow.add_node('fallback', fallback_node)
workflow.add_node('generate_with_context', generate_with_context_node)
workflow.add_node('filter_docs', filter_docs_node)

workflow.set_conditional_entry_point(
    router_node,
    {
        'VectorStore': 'vector_store',
        'SearchEngine': 'search_engine',
        'LLMFallback': 'fallback'
    }
)
workflow.add_edge('vector_store', 'filter_docs')
workflow.add_edge('search_engine', 'filter_docs')
workflow.add_conditional_edges(
    'filter_docs',
    lambda docs: 'search_engine' if len(docs) == 0 else
'generate_with_context',
    {
        'search_engine': 'search_engine',
        'generate_with_context': 'generate_with_context'
    }
)
workflow.add_edge('fallback', END)
workflow.add_edge('generate_with_context', END)
```