Performance using Langchain and OpenAI

Stan · Follow

4 min read · Jun 4, 2023

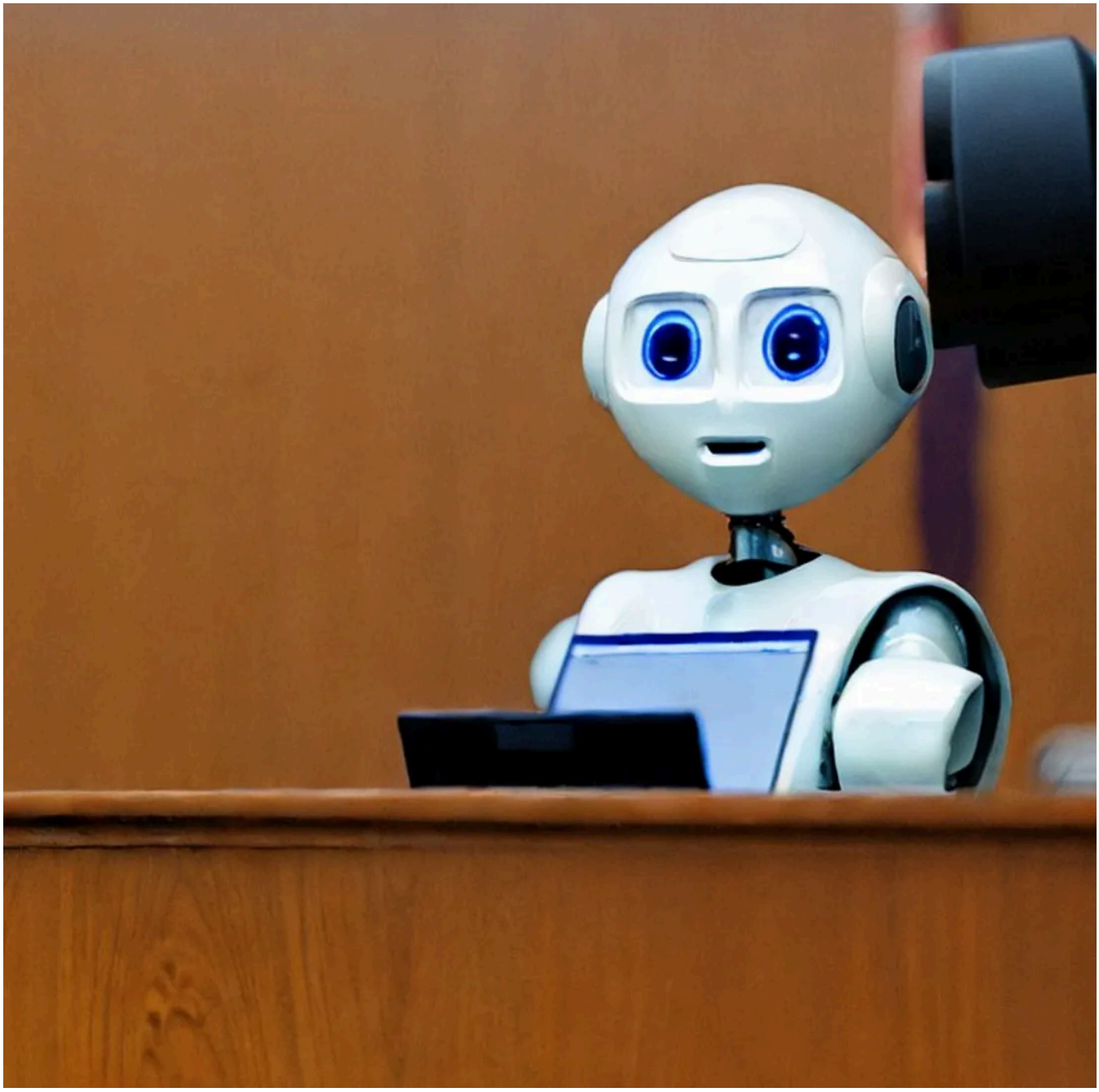


image is generated by author using stable diffusion2.1 with a prompt “ a robot as a judge in a court”

Motivation

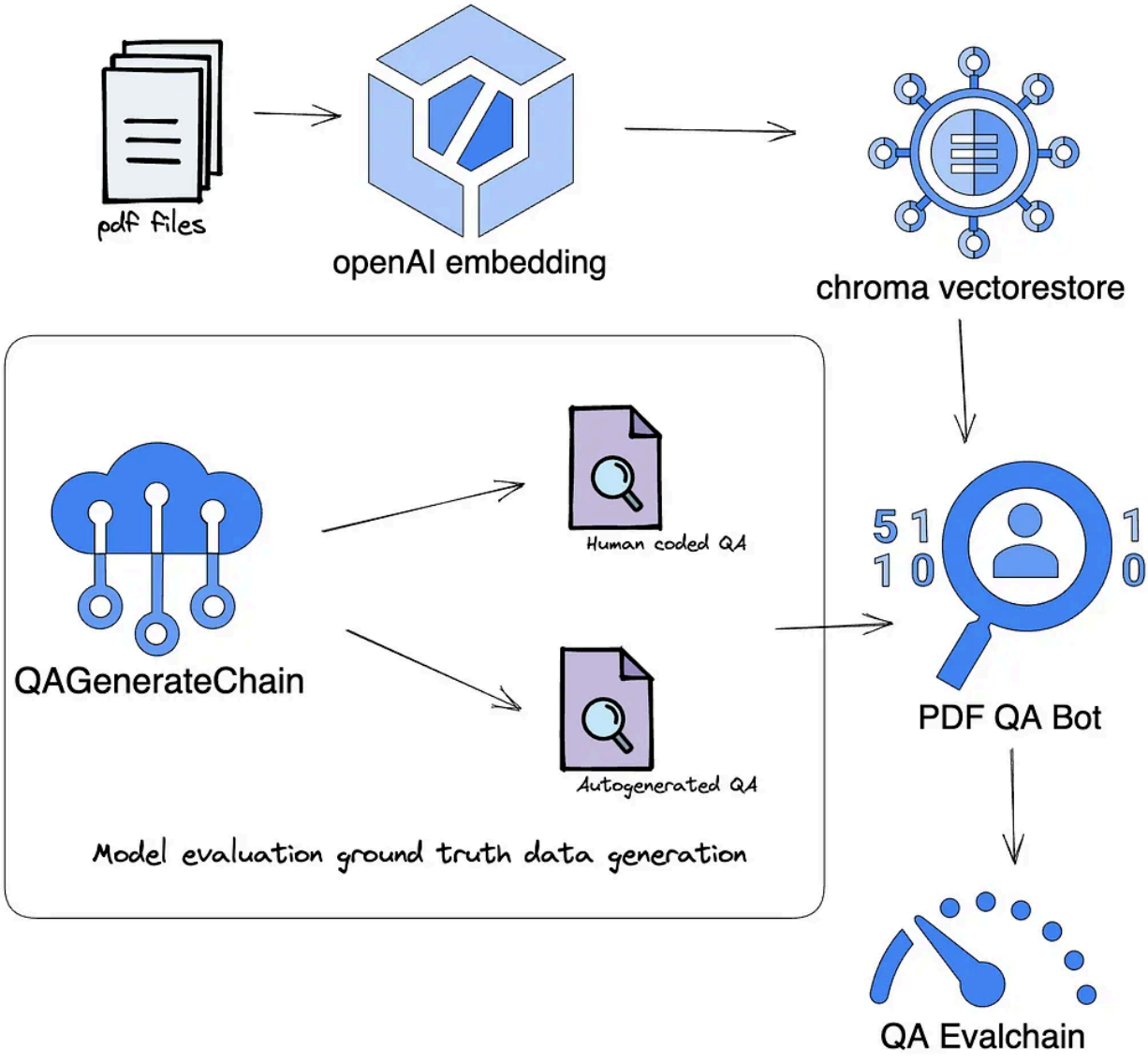
OpenAI and other large language models have gained significant momentum in the developer’s world. The OpenAI API and Langchain empower regular machine learning practitioners to use and develop LLMs without the need for supercomputer power.

To enable commercial applications of LLMs, we require clearly defined metrics to understand the performance of our models. However, unlike traditional machine learning algorithms, there is no universally accepted metric for developers to evaluate LLM performance.

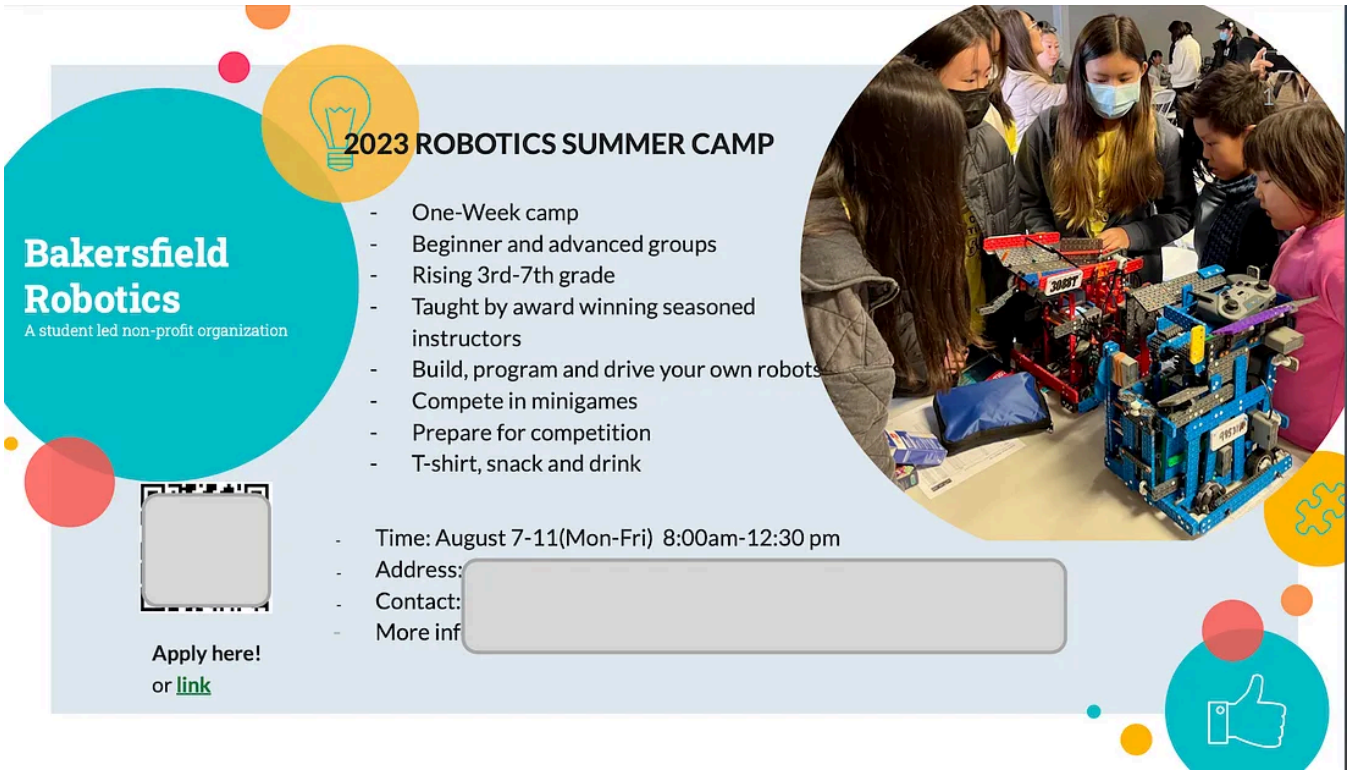
Fortunately, recently developed QA generation chains and QA evaluation chains provide developers with efficient means to evaluate model performance.

In this article we will cover following topics step by step

- Build a PDF QA Bot using Langchain retrievalQA chain
- Generate questions and answers based on QAgenerationChain
- Evaluate bot performance using QA Evaluation Chain



The workflow includes four interconnected parts: 1) The PDF is split, embedded, and stored in a vector store. 2) A PDF chatbot is built using the ChatGPT turbo model. 3) Ground truth data is generated by humans and/or the AutogenerateQA chain, which is then fed into the QA bot. 4) Finally, the QA evalchain is utilized to compare the model's predictions with the ground truth



A kids' summer camp flyer in PDF format is used to test the workflow because it is straightforward for humans to evaluate the chatbot's performance

Work Steps

```
from langchain.document_loaders import UnstructuredPDFLoader, OnlinePDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.vectorstores import Chroma
from langchain.embeddings.openai import OpenAIEmbeddings

from langchain.chains.question_answering import load_qa_chain
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI

from langchain.evaluation.qa import QAGenerateChain
from langchain.evaluation.qa import QAEvalChain

from dotenv import load_dotenv, find_dotenv
import os
```

load environmental variables

```
# Load environment variables
_ = load_dotenv(find_dotenv())
OPENAI_API_KEY = os.getenv('OPENAI_API_KEY')
```

Data preparation: load PDF, split documents

```
# Load PDF documents
def load_documents(file_path):
    loader = UnstructuredPDFLoader(file_path)
    return loader.load()

# Split documents into chunks
def chunk_documents(data):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overl
    return text_splitter.split_documents(data)

# Load PDF file
file_name = 'Robotic Summer Camp Flyer.pdf'
data = load_documents(file_name)


# Chunk documents
texts = chunk_documents(data)
print(f'Now you have {len(texts)} documents')
```

Set up OpenAI embedding, store in Chroma and set up QA chain

```
# Set up embeddings and vector store
embeddings = OpenAIEmbeddings(openai_api_key=OPENAI_API_KEY)
vectorstore = Chroma.from_documents(texts, embedding_function="gpt-turbo-3.5")

# Set up retrieval QA chain
llm = ChatOpenAI(temperature=0.0)
qa = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=vectorstore.as_retriever(),
    verbose=True,
    chain_type_kwargs={
        "document_separator": "<<<<>>>>>"
```

```
    }
)
```

Evaluate model performance by ground truth (hard coded answer and autogenerated answer)

- hard coded QA : QA query and answer are stored in a list of dictionary, which can be used by above QA retrievalQA chain

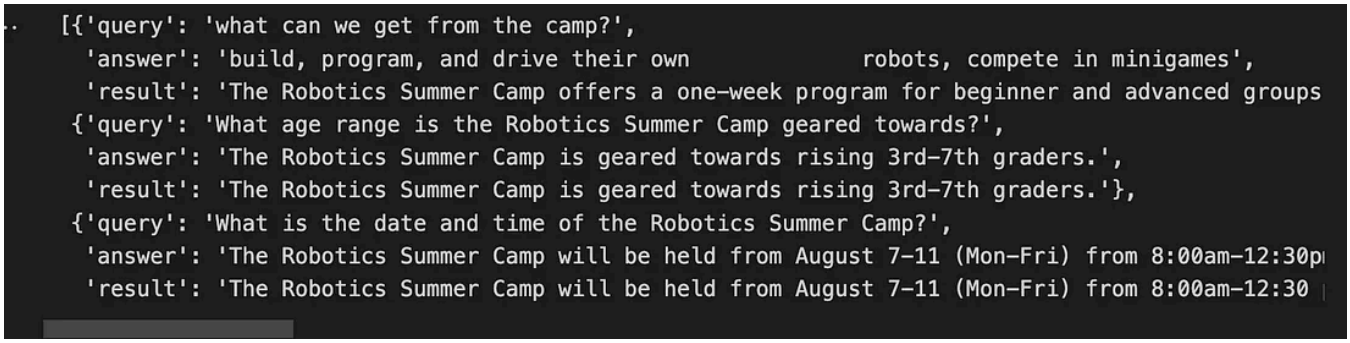
```
# hard coded answers
examples = [
    {
        "query": "what can we get from the camp?",
        "answer": "build, program, and drive their own \
            robots, compete in minigames"
    },
    {
        "query": "What age range is the Robotics Summer Camp geared towards?",
        "answer": "The Robotics Summer Camp is geared towards rising 3rd-7th gr
    }
]
```

- Another way to generate a QA dictionary is by using the QAGenerateChain, which provides a more efficient and scalable approach for generating answers. It is recommended to review these randomly generated QA pairs to ensure the accuracy of the answers.

```
# QA Generation

example_gen_chain = QAGenerateChain.from_llm(ChatOpenAI())

# auto generated Q/A
new_examples = example_gen_chain.apply_and_parse(
    [{"doc": t} for t in data]
)
```

Model prediction: predict answers using OpenAI+Langchain retreieve answer

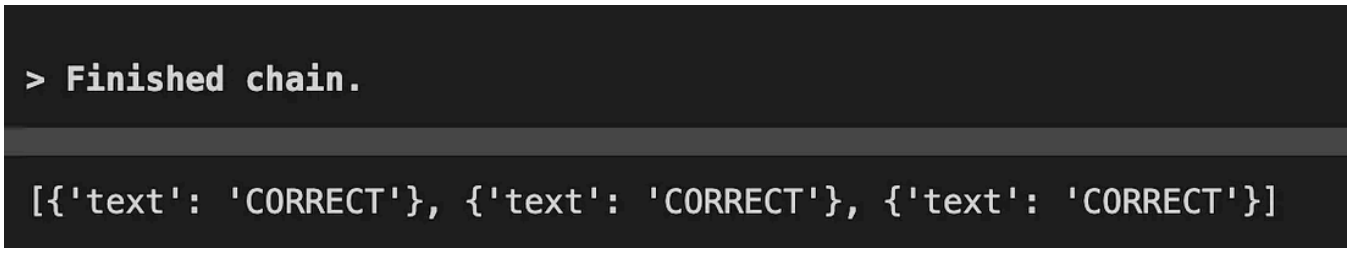
```
#combine hard QA with Autogenerated QA
examples += new_examples

# Predict answers
predictions = qa.apply(examples)
```



model prediction vs. ground truth. they matched pretty well. the first query and answer are quit different but they are essentially the same message with results are more verbose. That is why we use openAI for evaluation.

Model Evaluation: Evaluate answers by comparing model prediction and ground truth. The answer and result could be very different, which is why eval_chain is powered by ChatGPT as llm for evaluation.

```
# Evaluate the answers
llm = ChatOpenAI(temperature=0)
eval_chain = QAEvalChain.from_llm(llm)
graded_outputs = eval_chain.evaluate(examples, predictions)
graded_outputs
```



the chatbot did good job for this case. Since it is a simple pdf. More thorough evaluation should be carried out for more complicated tasks to build confidence for teams

In addition, you could use following helper function to calculate the accuracy of your model prediction and review the incorrect answers to improve your model.

```
total, correct, idx=0,0,0
for v in graded_outputs: # loop to compute accuracy
    total+=1; idx+=1
    if v['text'].strip()=='CORRECT':
        correct+=1
    else: # record index for incorrect predictions
        print(f"query: {predictions[idx]['query']}")
        print(f"Answer: {predictions[idx]['answer']}")
        print(f"Prediction: {predictions[idx]['result']}")

print(f'the model prediction accuracy is {correct/total*100} percent')
```

A sample output is illustrated below where the model prediction is incorrect.

```
query: What type of attention is used in the Transformer model?
Answer: The Transformer model uses multi-head attention in three different ways:
Prediction:  Self-attention.
query: What is the complexity per layer of self-attention (restricted)?
Answer: O(rnn d).
Prediction:  O(r*n*d)
query: What are three desiderata that motivated the use of self-attention in this
Answer: One desideratum was the total computational complexity per layer, another
Prediction:  The authors wanted self-attention to allow for more parallelization,
query: What is the training regime for the models described in the document?
Answer: The training regime for the models described in the document includes tra
Prediction:  The models were trained on the standard WMT 2014 English-German data
query: What is the best setting for single-head attention according to the docume
Answer: 0.9 BLEU worse than the best setting.
Prediction:  The document does not provide a best setting for single-head attenti
query: How many layers does the Transformer have according to the document?
Answer: 4 layers
Prediction:  4 layers
query: What were the parameters used in selecting the dropout, attention, and res
Answer: We performed only a small number of experiments to select the dropout, bo
Prediction:  The parameters used in selecting the dropout, attention, and residua
query: What paper is referenced in [11] of the document?
Answer: Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual lear
Prediction:  Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual
the model prediction accuracy is 83.33333333333334 percent
```

Model self evaluation result using above workflow on a larger pdf input “Attention is all you need”. The prediction accuracy is about 83% and the above list are incorrect answers for developers to debug.

Hope you found this article interesting. If you have any better tips or suggestions on using openAI or other LLM, please share them with us. Your clap and share are much appreciated. Have a great one and happy learning!

References

I highly recommend taking the short courses offered by deeplearning.ai for LLM developers and enthusiasts

https://www.deeplearning.ai/short-courses/

ChatGPT Langchain Model Evaluation Data Science

Follow

Written by Stan

1.3K Followers · 55 Following

A director data scientist working in a tech start-up who is passionate about making a positive impact on people around him

Responses (4)

Vijay Agrawal

What are your thoughts?

Drewonthemedia
Sep 16, 2023

In the QA EvaluationChain section, the apply_and_parse method is deprecated, instead pass an output parser directly to LLMChain.

1 Reply

Nicholas Kersting
Sep 1, 2023

Thanks this worked nicely, just a few tweaks were needed to your code however, possibly because things have gotten out of date. For example, the parameter to Chroma, vectorstore = Chroma.from_documents(texts, embedding=embeddings). But I'm sure... more

1 Reply

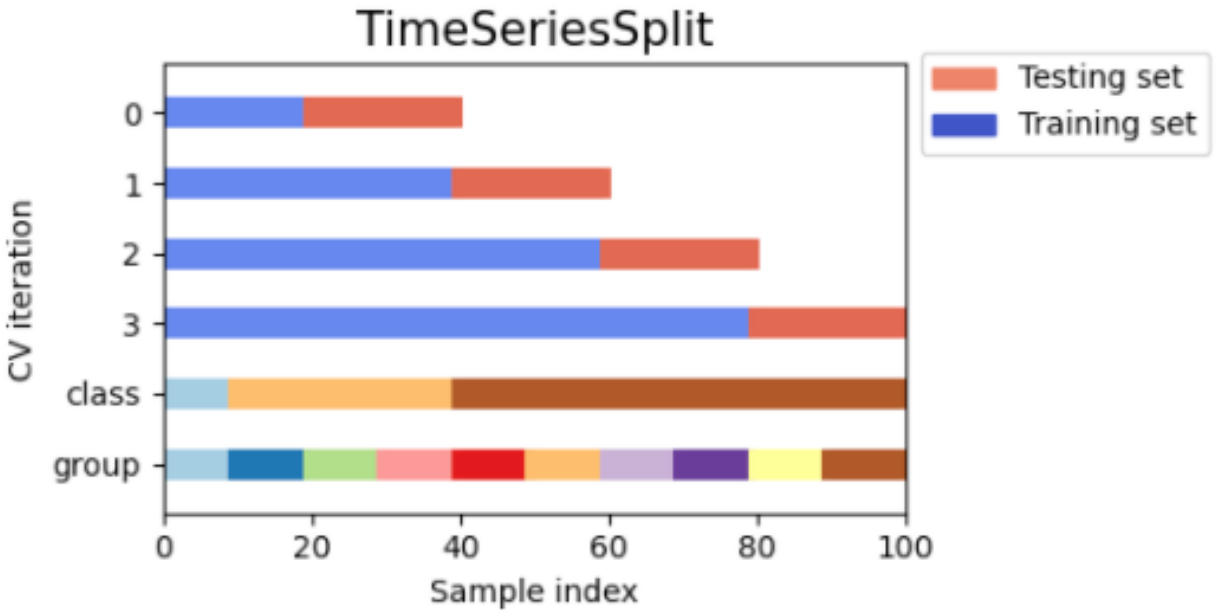
Matouš Eibich
Jul 25, 2023

Hello, thanks for a great article! Working on a similar thing and I was wondering if it's possible to create the baseline answers using the same Chroma vector store as is used during prediction. I would like that because my predicted answers are... more

1 Reply

See all responses

More from Stan



Stan

TimeSeries Split with Sklearn Tips

Nov 20, 2021 136

Stan

Creating a Retrieval-Augmented Generation (RAG) Model on Your Laptop for Free

Construct a Robust Knowledge Base Using Mistral-7B, LangChain, ChromaDB, and Streamlit

Feb 19, 2024 205

Stan

Build Free, Secure, and Traceable LLM Applications Using DeepSeek-R1

Install the Open-Source Deepseek R1 Locally and Chat with Your Own Data—The Results Are Impressive!

Feb 3 2

Stan

Improve Model Performance by stacking multiple models within sklearn pipeline

a case study using a highly imbalanced dataset

Jan 16, 2022 5

See all from Stan

Recommended from Medium

In Intel Tech by Intel

Tabular Data, RAG, & LLMs: Improve Results Through Data Table Prompting

How to ingest small tabular data when working with LLMs.

May 14, 2024 547 6

Siddhardhan S

Document Summarizer with LlamaIndex and Llama-3

Document summarization has become an essential task in today's fast-paced information world and it is an important use case in Generative...

Oct 11, 2024 5

Lists

ChatGPT prompts
51 stories · 2606 saves

The New Chatbots: ChatGPT, Bard, and Beyond
12 stories · 559 saves

ChatGPT
21 stories · 979 saves

What is ChatGPT?
9 stories · 514 saves

In Level Up Coding by Pavan Belagatti

What's the Best PDF Extractor for RAG? I Tried LlamaParse, Unstructured and Vectorize

If you're building retrieval augmented generation (RAG) applications, you will eventually need to work with documents that are in PDF form.

Feb 19 1K 13

Abonia Sojasingarayar

Introduction to spaCyLayout and PDF Extraction

Step by Step Tutorial

Feb 12 1 2

Batuhan Odabaş

ChatGPT commands to get real content

In the realm of content creation, AI has emerged as a game-changer. One AI tool that's making waves is ChatGPT.

Nov 3, 2024

Vipra Singh

AI Agents: Introduction (Part-1)

Discover AI agents, their design, and real-world applications.

Feb 3 847 20