Question Answering Benchmarking: State of the Union Address

Contents

Here we go over how to benchmark performance on a question answering task over a state of the union address.

It is highly reccomended that you do any evaluation/benchmarking with tracing enabled. See here for an explanation of what tracing is and how to set it up.

```
# Comment this out if you are NOT using tracing
import os
os.environ["LANGCHAIN_HANDLER"] = "langchain"
```

Loading the data

First, let's load the data.

```
from langchain.evaluation.loading import load_dataset
dataset = load_dataset("question-answering-state-of-the-union")
```

Skip to main content

```
Found cached dataset json
(/Users/harrisonchase/.cache/huggingface/datasets/LangChainDatasets___json/LangChainDatasets--question-answering-state-of-the-union-a7e5a3b2db4f440d/0.0.0/0f7e3662623656454fcd2b650f34e886a7db4b9104504885bd462096cc7a9f51)
```

Setting up a chain

Now we need to create some pipelines for doing question answering. Step one in that is creating an index over the data in question.

```
from langchain.document_loaders import TextLoader
loader = TextLoader("../../modules/state_of_the_union.txt")
```

```
from langchain.indexes import VectorstoreIndexCreator
```

```
vectorstore = VectorstoreIndexCreator().from_loaders([loader]).vectorstore
```

```
Running Chroma using direct local API.
Using DuckDB in-memory for database. Data will be transient.
```

Now we can create a question answering chain.

```
from langchain.chains import RetrievalQA
from langchain.llms import OpenAI
```

```
chain = RetrievalQA.from_chain_type(llm=OpenAI(), chain_type="stuff",
retriever=vectorstore.as_retriever(), input_key="question")
```

Make a prediction

First, we can make predictions one datapoint at a time. Doing it at this level of granularity allows use to explore the outputs in detail, and also is a lot cheaper than running over multiple

Skip to main content

```
chain(dataset[0])
```

```
{'question': 'What is the purpose of the NATO Alliance?',
 'answer': 'The purpose of the NATO Alliance is to secure peace and stability in
Europe after World War 2.',
 'result': ' The NATO Alliance was created to secure peace and stability in Europe
after World War 2.'}
```

Make many predictions

Now we can make predictions

```
predictions = chain.apply(dataset)
```

Evaluate performance

Now we can evaluate the predictions. The first thing we can do is look at them by eye.

```
predictions[0]
```

```
{'question': 'What is the purpose of the NATO Alliance?',
 'answer': 'The purpose of the NATO Alliance is to secure peace and stability in
Europe after World War 2.',
 'result': ' The purpose of the NATO Alliance is to secure peace and stability in
Europe after World War 2.'}
```

Next, we can use a language model to score them programatically

```
from langchain.evaluation.qa import QAEvalChain
```

```
llm = OpenAI(temperature=0)
eval_chain = QAEvalChain.from_llm(llm)
graded_outputs = eval_chain.evaluate(dataset, predictions,
question_key="question", prediction_key="result")
```

Skip to main content

We can add in the graded output to the `predictions` dict and then get a count of the grades.

```
for i, prediction in enumerate(predictions):
    prediction['grade'] = graded_outputs[i]['text']
```

```
from collections import Counter
Counter([pred['grade'] for pred in predictions])
```

```
Counter({' CORRECT': 7, ' INCORRECT': 4})
```

We can also filter the datapoints to the incorrect examples and look at them.

```
incorrect = [pred for pred in predictions if pred['grade'] == " INCORRECT"]
```

```
incorrect[0]
```

```
{'question': 'What is the U.S. Department of Justice doing to combat the crimes of
Russian oligarchs?',
 'answer': 'The U.S. Department of Justice is assembling a dedicated task force to
go after the crimes of Russian oligarchs.',
 'result': ' The U.S. Department of Justice is assembling a dedicated task force
to go after the crimes of Russian oligarchs and is naming a chief prosecutor for
pandemic fraud.',
 'grade': ' INCORRECT'}
```