SQL Question Answering Benchmarking: Chinook

Contents

Here we go over how to benchmark performance on a question answering task over a SQL database.

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("sql-qa-chinook")
```

```
Downloading and preparing dataset json/LangChainDatasets--sql-qa-chinook to
/Users/harrisonchase/.cache/huggingface/datasets/LangChainDatasets   json/LangChain
```

Skip to main content

```
7528565d2d992b47/0.0.0/0f7e3662623656454fcd2b650f34e886a7db4b9104504885bd462096cc7a
9f51...
```

```
Dataset json downloaded and prepared to
/Users/harrisonchase/.cache/huggingface/datasets/LangChainDatasets___json/LangChain
Datasets--sql-qa-chinook-
7528565d2d992b47/0.0.0/0f7e3662623656454fcd2b650f34e886a7db4b9104504885bd462096cc7a
9f51. Subsequent calls will reuse this data.
```

```
dataset[0]
```

```
{'question': 'How many employees are there?', 'answer': '8'}
```

Setting up a chain

This uses the example Chinook database. To set it up follow the instructions on https://database.guide/2-sample-databases-sqlite/, placing the `.db` file in a notebooks folder at the root of this repository.

Note that here we load a simple chain. If you want to experiment with more complex chains, or an agent, just create the `chain` object in a different way.

```
from langchain import OpenAI, SQLDatabase, SQLDatabaseChain
```

```
db = SQLDatabase.from_uri("sqlite:///../../../notebooks/Chinook.db")
llm = OpenAI(temperature=0)
```

Now we can create a SQL database chain.

```
chain = SQLDatabaseChain(llm=llm, database=db, input_key="question")
```

Skip to main content

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 datapoints

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

```
{'question': 'How many employees are there?',
 'answer': '8',
 'result': ' There are 8 employees.'}
```

Make many predictions

Now we can make predictions. Note that we add a try-except because this chain can sometimes error (if SQL is written incorrectly, etc)

```
predictions = []
predicted_dataset = []
error_dataset = []
for data in dataset:
    try:
        predictions.append(chain(data))
        predicted_dataset.append(data)
    except:
        error_dataset.append(data)
```

Evaluate performance

Now we can evaluate the predictions. We can use a language model to score them programatically

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

Skip to main content

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

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': 3, ' 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': 'How many employees are also customers?',
 'answer': 'None',
 'result': ' 59 employees are also customers.',
 'grade': ' INCORRECT'}
```