6.1_LLM_Testing_1a_CG

November 12, 2024

Testing of 1a CodeGemma with RAG and Schema

Loading packages, libraries and secrets into notebook

[20]:
```
# Importing the required libraries
from langchain_openai import OpenAIEmbeddings
from langchain_openai import OpenAI
from langchain_openai import ChatOpenAI
from langchain_core.output_parsers import StrOutputParser
from langchain.prompts import ChatPromptTemplate
from langchain_core.runnables import RunnableParallel, RunnablePassthrough
import os
from dotenv import load_dotenv
from datasets import load_dataset
import pandas as pd
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_anthropic import ChatAnthropic
```

[21]:
```
# Accessing the secrets from the environment variables
load_dotenv()
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
GOOGLE_API_KEY = os.getenv("GOOGLE_API_KEY")
ANTHROPIC_API_KEY = os.getenv("ANTHROPIC_API_KEY")
```

Loading data into dataframe for testing

[ ]:
```
# Upload the dataset and transform to dataframe
# Define the dataset path
dataset_path = "../8_Testing_Input_and_Output/App_Output_1a_CG.csv"
print("Dataset Path:", dataset_path)

# Check if the file exists at the specified path
if not os.path.isfile(dataset_path):
    raise FileNotFoundError(f"Unable to find the file at {dataset_path}")

# Load the dataset
testing_output_1a = load_dataset('csv', data_files=dataset_path)

# Convert the dataset to a pandas dataframe
```

```
df_1a_testing_output = testing_output_1a['train'].to_pandas()

# Print a few rows to verify
print(df_1a_testing_output.head())
```

Testing Template

Explanation Assessment Template

[23]:
```
# Chain setup explanation
testing_template_explanation = """
"How well does the following Explanation explain the SQL Query? Please assess␣
 ↪it critically then assign and output one of the following scores where 4 is␣
 ↪the highest and 1 is the lowest: Acceptable (4), Minor errors (3), Major␣
 ↪errors (2), or Unacceptable (1). To determine the score, go through the␣
 ↪assessment step by step and consider the accuracy and understandability of␣
 ↪the explanation assigning one score for accuracy and understandability, and␣
 ↪a combined overall score for the explanation."

SQL Query: {query}

Explanation: {explanation}

Question: {question}
"""

prompt_testing_explanation = ChatPromptTemplate.
 ↪from_template(testing_template_explanation)
```

Translation Assessment Template

[24]:
```
# Chain setup translation
testing_template_translation = """
"How well does the following Translation translate the SQL Query? Please assess␣
 ↪it critically then assign and output one of the following scores where 4 is␣
 ↪the highest and 1 is the lowest: Acceptable (4), Minor errors (3), Major␣
 ↪errors (2), or Unacceptable (1). To determine the score, go through the␣
 ↪assessment step by step and consider the accuracy and understandability of␣
 ↪the translation assigning one score for accuracy and understandability, and␣
 ↪a combined overall score for the translation."

SQL Query: {query}

Translation: {translation}

Question: {question}
"""
```

```
prompt_testing_translation = ChatPromptTemplate.
↪from_template(testing_template_translation)
```

OpenAI Assessment

Explanation Assessment

[25]:
```
# Model and parsing setup
model = ChatOpenAI(api_key=OPENAI_API_KEY, model="gpt-4o-mini")
parser = StrOutputParser()

chain_testing_OAI_explanation = (
    {"query": RunnablePassthrough(), "explanation": RunnablePassthrough(),␣
↪"question": RunnablePassthrough()}
    | prompt_testing_explanation
    | model
    | parser
)

# Function to compare each question and result using the chain
def Explanation_testing_OAI(df_1a_testing_output):
    assessment_OAI_explanation = []

    for i, row in df_1a_testing_output.iterrows():
        # Get the question and result from the dataframe
        query = row["Query"]
        question = row["Question"]
        explanation = row["Explanation"]

        # Create a dictionary with query and result to pass to the chain
        inputs = {"query": query, "explanation": explanation, "question" :␣
↪question}

        # Run the chain and catch any potential errors
        try:
            test_output_OAI_explanation = chain_testing_OAI_explanation.
↪invoke(inputs)
        except Exception as e:
            test_output_OAI_explanation = f"Error in row {i}: {str(e)}"

        # Store the comparison output
        assessment_OAI_explanation.append( test_output_OAI_explanation)

    # Add the comparison results to a new column
    df_1a_testing_output["Assessment OAI Explanation"] =␣
↪assessment_OAI_explanation

    return df_1a_testing_output
```

```
# Call the function and process the dataframe
df_explanation_assessment_OAI_ = Explanation_testing_OAI(df_1a_testing_output)
```

Translation Assessment

```
[26]: chain_testing_OAI_translation = (
    {"query": RunnablePassthrough(), "translation": RunnablePassthrough(),␣
 ↪"question": RunnablePassthrough()}
    | prompt_testing_translation
    | model
    | parser
)

# Function to compare each question and result using the chain
def Translation_testing_OAI(df_1a_testing_output):
    assessment_OAI_translation = []

    for i, row in df_1a_testing_output.iterrows():
        # Get the question and result from the dataframe
        query = row["Query"]
        translation = row["Translation"]
        question = row["Question"]

        # Create a dictionary with query and result to pass to the chain
        inputs = {"query": query, "translation": translation, "question" :␣
 ↪question}

        # Run the chain and catch any potential errors
        try:
            test_output_OAI_translation = chain_testing_OAI_translation.
 ↪invoke(inputs)
        except Exception as e:
            test_output_OAI_translation = f"Error in row {i}: {str(e)}"

        # Store the comparison output
        assessment_OAI_translation.append( test_output_OAI_translation)

    # Add the comparison results to a new column
    df_1a_testing_output["Assessment OAI Translation"] =␣
 ↪assessment_OAI_translation

    return df_1a_testing_output

# Call the function and process the dataframe
df_translation_assessment_OAI = Translation_testing_OAI(df_1a_testing_output)
```

Gemini Assessment

Explanation Assessment

```
[27]: Gemini_model = ChatGoogleGenerativeAI(model="gemini-pro",␣
       ↪api_key=GOOGLE_API_KEY)


      chain_testing_Gemi_explanation = (
          {"query": RunnablePassthrough(), "explanation": RunnablePassthrough(),␣
       ↪"question": RunnablePassthrough()}
          | prompt_testing_explanation
          | Gemini_model
          | parser
      )

      # Function to compare each question and result using the chain
      def Explanation_testing_Gemi(df_1a_testing_output):
          assessment_Gemi_explanation = []

          for i, row in df_1a_testing_output.iterrows():
              # Get the question and result from the dataframe
              query = row["Query"]
              question = row["Question"]
              explanation = row["Explanation"]

              # Create a dictionary with query and result to pass to the chain
              inputs = {"query": query, "explanation": explanation, "question" :␣
       ↪question}

              # Run the chain and catch any potential errors
              try:
                  test_output_Gemi_explanation = chain_testing_Gemi_explanation.
       ↪invoke(inputs)
              except Exception as e:
                  test_output_Gemi_explanation = f"Error in row {i}: {str(e)}"

              # Store the comparison output
              assessment_Gemi_explanation.append(test_output_Gemi_explanation)

          # Add the comparison results to a new column
          df_1a_testing_output["Assessment Gemini Explanation"] =␣
       ↪assessment_Gemi_explanation

          return df_1a_testing_output

      # Call the function and process the dataframe
      df_explanation_assessment_Gemi = Explanation_testing_Gemi(df_1a_testing_output)
```

Translation Assessment

[28]:
```
chain_testing_Gemi_translation = (
    {"query": RunnablePassthrough(), "translation": RunnablePassthrough(),␣
 ↪"question": RunnablePassthrough()}
    | prompt_testing_translation
    | Gemini_model
    | parser
)

# Function to compare each question and result using the chain
def Translation_testing_Gemi(df_1a_testing_output):
    assessment_Gemi_translation = []

    for i, row in df_1a_testing_output.iterrows():
        # Get the question and result from the dataframe
        query = row["Query"]
        question = row["Question"]
        translation = row["Translation"]

        # Create a dictionary with query and result to pass to the chain
        inputs = {"query": query, "translation": translation, "question" :␣
 ↪question}

        # Run the chain and catch any potential errors
        try:
            test_output_Gemi_translation = chain_testing_Gemi_translation.
 ↪invoke(inputs)
        except Exception as e:
            test_output_Gemi_translation = f"Error in row {i}: {str(e)}"

        # Store the comparison output
        assessment_Gemi_translation.append(test_output_Gemi_translation)

    # Add the comparison results to a new column
    df_1a_testing_output["Assessment Gemini Translation"] =␣
 ↪assessment_Gemi_translation

    return df_1a_testing_output

# Call the function and process the dataframe
df_translation_assessment_Gemi = Translation_testing_Gemi(df_1a_testing_output)
```

Claude Assessment

Explanation Assessment

[29]:
```
Claude_model = ChatAnthropic(model="claude-3-5-sonnet-20240620",␣
 ↪api_key=ANTHROPIC_API_KEY)
```

```
chain_testing_Claude_explanation = (
    {"query": RunnablePassthrough(), "explanation": RunnablePassthrough(),␣
 ↪"question": RunnablePassthrough()}
    | prompt_testing_explanation
    | Claude_model
    | parser
)

# Function to compare each question and result using the chain
def Explanation_testing_Claude(df_1a_testing_output):
    assessment_Claude_explanation = []

    for i, row in df_1a_testing_output.iterrows():
        # Get the question and result from the dataframe
        query = row["Query"]
        question = row["Question"]
        explanation = row["Explanation"]

        # Create a dictionary with query and result to pass to the chain
        inputs = {"query": query, "explanation": explanation, "question" :␣
 ↪question}

        # Run the chain and catch any potential errors
        try:
            test_output_Claude_explanation = chain_testing_Claude_explanation.
 ↪invoke(inputs)
        except Exception as e:
            test_output_Claude_explanation = f"Error in row {i}: {str(e)}"

        # Store the comparison output
        assessment_Claude_explanation.append(test_output_Claude_explanation)

    # Add the comparison results to a new column
    df_1a_testing_output["Assessment Claude Explanation"] =␣
 ↪assessment_Claude_explanation

    return df_1a_testing_output

# Call the function and process the dataframe
df_explanation_assessment_Claude =␣
 ↪Explanation_testing_Claude(df_1a_testing_output)
```

Translation Assessment

[19]:
```
chain_testing_Claude_translation = (
    {"query": RunnablePassthrough(), "translation": RunnablePassthrough(),␣
↪"question": RunnablePassthrough()}
    | prompt_testing_translation
    | Claude_model
    | parser
)

# Function to compare each question and result using the chain
def Translation_testing_Claude(df_1a_testing_output):
    assessment_Claude_translation = []

    for i, row in df_1a_testing_output.iterrows():
        # Get the question and result from the dataframe
        query = row["Query"]
        question = row["Question"]
        translation = row["Translation"]

        # Create a dictionary with query and result to pass to the chain
        inputs = {"query": query, "translation": translation, "question" :␣
↪question}

        # Run the chain and catch any potential errors
        try:
            test_output_Claude_translation = chain_testing_Claude_translation.
↪invoke(inputs)
        except Exception as e:
            test_output_Claude_translation = f"Error in row {i}: {str(e)}"

        # Store the comparison output
        assessment_Claude_translation.append(test_output_Claude_translation)

    # Add the comparison results to a new column
    df_1a_testing_output["Assessment Claude Translation"] =␣
↪assessment_Claude_translation

    return df_1a_testing_output

# Call the function and process the dataframe
df_translation_assessment_Claude =␣
↪Translation_testing_Claude(df_1a_testing_output)

# Save the dataframe, including the comparison, to a CSV file
df_translation_assessment_Claude.to_csv("../8_Testing_Input_and_Output/
↪LLM_assessment_1a_CG.csv", index=False)
```