7.1_Translation_Testing_1a_CG

November 12, 2024

Translation Testing of 1a CodeGemma with RAG and Schema

Loading packages, libraries and secrets into notebook

[ ]:
```
# Importing the required libraries
import os
from dotenv import load_dotenv
from datasets import load_dataset
import pandas as pd
from transformers import AutoTokenizer, AutoModel
from sentence_transformers import SentenceTransformer
from sentence_transformers.util import cos_sim
from sentence_transformers.quantization import quantize_embeddings
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
```

[ ]:
```
# Accessing the secrets from the environment variables
load_dotenv()
HF_Token = os.getenv("HF_TOKEN")
```

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())
```

Semantic Translation Testing

Embedding 1

[ ]:
```
# Load a pre-trained model for generating sentence embeddings
embedding_model_1 = AutoModel.from_pretrained('jinaai/jina-embeddings-v3',␣
↪trust_remote_code=True) # https://huggingface.co/jinaai/jina-embeddings-v3

# Function to compute embeddings and similarity
def Translation_assessment_1(df_1a_testing_output):
    df_1a_testing_output['Question'] = df_1a_testing_output['Question'].
↪fillna('').astype(str)
    df_1a_testing_output['Translation'] = df_1a_testing_output['Translation'].
↪fillna('').astype(str)

    # Generate embeddings for the "Question" and "Translation" columns
    question_embeddings = embedding_model_1.
↪encode(df_1a_testing_output['Question'].tolist(), task="text-matching",␣
↪convert_to_tensor=True)
    translation_embeddings = embedding_model_1.
↪encode(df_1a_testing_output['Translation'].tolist(), task="text-matching",␣
↪convert_to_tensor=True)

    # Calculate cosine similarity for each row
    similarities = cosine_similarity(question_embeddings,␣
↪translation_embeddings)

    # Since cosine_similarity returns a matrix, we extract the diagonal␣
↪(row-wise comparison)
    df_1a_testing_output['Similarity_1'] = np.diagonal(similarities)

    return df_1a_testing_output

# Call the function and process the dataframe
df_translation_assessment_1 = Translation_assessment_1(df_1a_testing_output)
```

Embedding 2

[ ]:
```
# 1. Specify preferred dimensions
dimensions = 512

# 2. Load the model
embedding_model_2 = SentenceTransformer("mixedbread-ai/mxbai-embed-large-v1",␣
↪truncate_dim=dimensions)  # https://huggingface.co/mixedbread-ai/
↪mxbai-embed-large-v1

# Function to generate a detailed instruction for the query
def get_detailed_instruct(task_description: str, query: str) -> str:
```

```
    return f'Instruct: {task_description}\nQuery: {query}'

# Function to compute embeddings and similarity
def Translation_assessment_2(df_1a_testing_output):
    # Define task instruction for the queries
    task = 'Compare the question and translation to assess the quality of the␣
 ↪translation.'

    # Add instruction to the "Question" column
    questions_with_instructions = [
        get_detailed_instruct(task, question) for question in␣
 ↪df_1a_testing_output['Question'].tolist()
    ]

    # Generate a list of documents to encode
    docs = questions_with_instructions + df_1a_testing_output['Translation'].
 ↪tolist()

    # 2. Encode
    embeddings = embedding_model_2.encode(docs)

    # Optional: Quantize the embeddings
    binary_embeddings = quantize_embeddings(embeddings, precision="ubinary")

    # Calculate cosine similarity between the first half (questions) and the␣
 ↪second half (translations)
    question_embeddings = embeddings[:len(questions_with_instructions)]
    translation_embeddings = embeddings[len(questions_with_instructions):]

    # Calculate cosine similarity
    similarities = cos_sim(question_embeddings, translation_embeddings)

    # Since cos_sim returns a matrix, we extract the diagonal (row-wise␣
 ↪comparison)
    df_1a_testing_output['Similarity_V2'] = np.diagonal(similarities.cpu().
 ↪numpy())

    return df_1a_testing_output

# Call the function and process the dataframe
df_translation_assessment_2 = Translation_assessment_2(df_1a_testing_output)
```

Embedding 3

[ ]:
```
# Load a pre-trained model for generating sentence embeddings
embedding_model = SentenceTransformer("thenlper/gte-large")
```

```
# Function to compute embeddings and similarity
def Translation_assessment_3(df_1a_testing_output):
    # Generate embeddings for the "Question" and "Translation" columns
    question_embeddings = embedding_model.
 ↪encode(df_1a_testing_output['Question'].tolist(), convert_to_tensor=True).
 ↪cpu()
    translation_embeddings = embedding_model.
 ↪encode(df_1a_testing_output['Translation'].tolist(), convert_to_tensor=True).
 ↪cpu()

    # Calculate cosine similarity for each row
    similarities = cosine_similarity(question_embeddings,␣
 ↪translation_embeddings)

    # Since cosine_similarity returns a matrix, we extract the diagonal␣
 ↪(row-wise comparison)
    df_1a_testing_output['Similarity_V3'] = np.diagonal(similarities)

    return df_1a_testing_output

# Call the function and process the dataframe
df_translation_assessment_3 = Translation_assessment_3(df_1a_testing_output)

# Saving to CSV with the similarity score
df_translation_assessment_3.to_csv('../8_Testing_Input_and_Output/
 ↪Translation_assessment_1a_CG.csv', index=False)
```