

Intro

Services used:

- Vertex AI (instance in workbench, where to open the notebook): have to activate the instance before using
- Cloud Storage (store the data file)
- BigQuery (connection to dashboard)

```
In [12]: # Libraries
import logging
from google.cloud import aiplatform
from IPython.display import display, HTML, Markdown
import plotly.graph_objects as go
import nest_asyncio
import warnings
import vertexai
from vertexai.preview.evaluation import (
    EvalTask,
    CustomMetric,
    make_metric,
)
import langid
from google.cloud import storage
from google.cloud import bigquery
import pandas as pd
import io
import numpy as np
from ast import literal_eval
import pandas_gbq
from google.auth import default
```

```
In [2]: # Initialize Vertex AI and BigQuery
vertexai.init(project='[REDACTED]', location='[REDACTED]')
client = bigquery.Client()

# Setup logging and warnings
logging.getLogger("urllib3.connectionpool").setLevel(logging.ERROR)
nest_asyncio.apply()
warnings.filterwarnings("ignore")
```

Data cleaning

1. Read the file from Cloud Storage using the pre-defined bucket name and file name
2. Reformat the "Conversation" column to a standard "question-answer" pair for future evaluation
3. Separate the "Conversation" column to two columns, "question" and "answer" each
4. Since Vertex AI doesn't provide high accuracy with non English text, we store the English conversation in a separate data frame and only evaluate those only

```
In [3]: # Load data from GCS
bucket_name = '████████████████████'
file_name = '██████████████████'
client = storage.Client()
bucket = client.get_bucket(bucket_name)
blob = bucket.blob(file_name)
data = blob.download_as_bytes()
df = pd.read_excel(io.BytesIO(data))
df.head()
```

```
Out[3]:
```

	Unnamed: 0	ID	Conversation
0			
1			
2			
3			
4			

```
In [4]: # Preprocess the data
def safe_convert(value):
    try:
        return literal_eval(value)
    except (SyntaxError, ValueError):
        return []
df["Conversation"] = df["Conversation"].apply(safe_convert)

def reformat_conversation(conversation):
    formatted_conversation = [
        {"question": msg["content"]} if msg.get("role") == "user" else {"answer": msg["content"]}
        for msg in conversation
    ]
    return formatted_conversation
df["Conversation"] = df["Conversation"].apply(reformat_conversation)

def extract_values(row, key, index):
    try:
        return row[index][key]
    except (IndexError, KeyError):
        return None
df['question'] = df['Conversation'].apply(lambda x: extract_values(x, 'question', 0))
df['answer'] = df['Conversation'].apply(lambda x: extract_values(x, 'answer', 1))
```

```
df = df.dropna()
df.reset_index(drop=True, inplace=True)
df
```

Out[4]:

Unnamed: 0	ID	Conversation	question	answer
0				
1				
2				
3				
4				
...				
434				
435				
436				
437				
438				

439 rows x 5 columns

```
In [5]: # Language detection
def detect_language(text):
    return langid.classify(text)[0] == 'en' # Returns True if language is Eng
```

```
df['is_english'] = df['question'].apply(detect_language)

df_en = df[df['is_english'] == True] # Keep rows where is_english is True
df_other = df[df['is_english'] == False]

df_en.drop(columns=['is_english'], inplace=True)
df_other.drop(columns=['is_english'], inplace=True) # Drop the temporary column
df_en
```

Out [5]:

Unnamed:
0

ID Conversation question answer

0	<div></div>			
1				
2				
4				
5				
...				
434				
435				
436				
437				
438				

421 rows x 5 columns

Evaluation

- 1. Two helper functions to display the evaluation report and example explanation

2. Self identified metrics (for now, only display those that can be done with only user questions and AI responses)

This part has a reference notebook from Google Vertex AI website:

https://colab.research.google.com/github/GoogleCloudPlatform/generative-ai/blob/main/gemini/evaluation/evaluate_rag_rapid_evaluation_sdk.ipynb#scrollTo=tdMpJnLKXv

```
In [6]: # Helper functions
def display_eval_report(eval_result, metrics=None):
    """Display the evaluation results."""

    title, summary_metrics, report_df = eval_result
    metrics_df = pd.DataFrame.from_dict(summary_metrics, orient="index").T
    if metrics:
        metrics_df = metrics_df.filter(
            [
                metric
                for metric in metrics_df.columns
                if any(selected_metric in metric for selected_metric in metrics)
            ]
        )
        report_df = report_df.filter(
            [
                metric
                for metric in report_df.columns
                if any(selected_metric in metric for selected_metric in metrics)
            ]
        )

    # Display the title with Markdown for emphasis
    display(Markdown(f"## {title}"))

    # Display the metrics DataFrame
    display(Markdown("### Summary Metrics"))
    display(metrics_df)

    # Display the detailed report DataFrame
    display(Markdown(f"### Report Metrics"))
    display(report_df)

def display_explanations(df, metrics=None, n=1):
    style = "white-space: pre-wrap; width: 800px; overflow-x: auto;"
    df = df.sample(n=n)
    if metrics:
        df = df.filter(
            [
                "instruction", "context", "reference", "completed_prompt", "response"
            ] + [
                metric
                for metric in df.columns
                if any(selected_metric in metric for selected_metric in metrics)
            ]
        )

    for index, row in df.iterrows():
        for col in df.columns:
```

```
display(HTML(f"<h2>{col}:</h2> <div style='{style}'>{row[col]}</div>"))
display(HTML("<hr>"))
```

Note that to run the current full data file with those valid metrics takes around 6 mins, so for testing purpose, this part can be changed to reduce the testing time:

```
eval_dataset = pd.DataFrame( { "instruction": df_en["question"], "response":
df_en["answer"], } )
```

to

```
eval_dataset = pd.DataFrame( { "instruction": df_en["question"][:10], "response":
df_en["answer"][:10], } )
```

```
In [7]: # Prepare evaluation dataset
eval_dataset = pd.DataFrame(
    {
        "instruction": df_en["question"],
        "response": df_en["answer"],
    }
)

# Evaluate
eval_task = EvalTask(
    dataset=eval_dataset,
    metrics=[
        'question_answering_quality',
        'question_answering_relevance',
        'question_answering_helpfulness',
        '"summarization_quality"',
        '"summarization_verbosity"',
        '"summarization_helpfulness"',
        'fulfillment',
        'groundedness',
    ],
    experiment='legal-ai-evaluation'
)

eval_result = eval_task.evaluate()
```

 [VIEW EXPERIMENT](#)

Associating projects/197532339649/locations/us-central1/metadataStores/default/contexts/legal-ai-evaluation-0c86f285-62a3-42dc-89fe-dc420b7dc3f7 to Experiment: legal-ai-evaluation

 [VIEW EXPERIMENT RUN](#)

Computing metrics with a total of 1263 Vertex online evaluation service requests.

100% | ██████████ | 1263/1263 [06:28<00:00, 3.25it/s]

3 errors encountered during evaluation. Continue to compute summary metrics for the rest of the dataset.

Error encountered for metric question_answering_relevance at dataset index 238: Error: 400 List of found errors: 1.Field: question_answering_relevance_input.instance.instruction; Message: Required field is not set. [field_violations {
 field: "question_answering_relevance_input.instance.instruction"
 description: "Required field is not set."
}]

Error encountered for metric question_answering_helpfulness at dataset index 238: Error: 400 List of found errors: 1.Field: question_answering_helpfulness_input.instance.instruction; Message: Required field is not set. [field_violations {
 field: "question_answering_helpfulness_input.instance.instruction"
 description: "Required field is not set."
}]

Error encountered for metric fulfillment at dataset index 238: Error: 400 List of found errors: 1.Field: fulfillment_input.instance.instruction; Message: Required field is not set. [field_violations {
 field: "fulfillment_input.instance.instruction"
 description: "Required field is not set."
}]

Evaluation Took:388.3905657760006 seconds

```
In [8]: # Display evaluation results
display_eval_report(("Eval Result", eval_result.summary_metrics, eval_result.r
```

Eval Result

Summary Metrics

row_count	question_answering_relevance/mean	question_answering_relevance/std	question_
0			

Report Metrics

	instruction	response	question_answering_relevance/explanation	question_answering_
0	<div></div>			
1				
2				
3				
4				
...				
416	<div></div>			
417				
418				
419				
420				

```
In [9]: # display_explanations(eval_result.metrics_table, n=2)
```

```
In [10]: # display_explanations(eval_result.metrics_table, metrics=["summarization_help
```

Things to be done

1. **Ground truth** : for legal questions, we need the manually verified ground truth to evaluate the accuracy of the content that legal AI generated in order to make it fine tuned
2. **Context** : text to reference while evaluating
3. **Categorize the user input** : if they should be evaluated as "Summarization" or "Question answering" since they will follow the different rules of inputting

Summarization inputs:

- instruction: provided at inference time. Instructions can include information such as tone and formatting. For example, Summarize the text from the point of view of the computer, including all references to AI.
- context: the text to be summarized.
- prediction: the LLM response from the instruction and context parameters.

So, the following metrics cannot be evaluated yet: [summarization_quality](#), [summarization_helpfulness](#), [summarization_verbosity](#).

Question answering inputs (doesn't apply to all metrics):

- instruction: the question to be answered and the answering instructions are provided at inference time. Instructions can include information such as tone and formatting. For example, How long does it take to bake the apple pie? Give an overestimate and an underestimate in your response.
- context: the text to reference when answering the question. In our example for inference_instruction, this might include the text on a page of a cooking website.
- prediction: the LLM response from the instruction and context parameters.
- baseline_prediction (ground truth): the baseline LLM response to be compared against prediction. Both responses share the same instruction and context.

So, the following metrics cannot be evaluated yet: [question_answering_quality](#).

Connect to BigQuery for dashboard

1. More metrics name can be added into "schema" later when they are evaluated

result

Data loaded to