Dataset Preprocessing and Model Training Documentation

This report provides a detailed overview of the dataset preprocessing and model training steps involved in the project. The goal is to prepare the dataset for summarization, detect context mismatches, and evaluate the quality of generated summaries. Below is a comprehensive breakdown of the process:

1. Dataset Preprocessing

1.1 Dataset Overview

- Source: The dataset is stored in a CSV file (`final\_labels.csv`).

- Columns:

- `entry\_id`: Unique identifier for each comment.

- `parent\_id`: Identifier of the parent comment (if any).

- `body`: Text content of the comment.

- Objective: Reconstruct discussion threads, generate summaries, and detect context mismatches.

1.2 Data Cleaning

- Handling Missing Data:

- Rows with missing `body` text are dropped using `df.dropna(subset=['body'])`.

- This ensures that only valid comments are processed.

- Handling Missing Parent IDs:

- Comments without a `parent\_id` are treated as root comments (starting points of threads).

1.3 Graph Construction

- Directed Graph:

- A directed graph (`networkx.DiGraph`) is constructed to represent the hierarchical structure of the threads.

- Nodes: Represent comments, with attributes like `body` and `parent\_id`.

- Edges: Represent parent-child relationships between comments.

- Graph Population:

- Iterate through the dataset and add nodes and edges to the graph.

- Example:

```python

for \_, row in df.iterrows():

G.add\_node(row['entry\_id'], body=row['body'], parent=row['parent\_id'])

if pd.notna(row['parent\_id']):

G.add\_edge(row['parent\_id'], row['entry\_id'])

1.4 Thread Reconstruction

- Depth-First Search (DFS):

- DFS is used to reconstruct full discussion threads starting from root comments.

- Example:

python

def get\_thread(root\_id):

thread = []

for node in nx.dfs\_preorder\_nodes(G, source=root\_id):

thread.append(G.nodes[node]['body'])

return " ".join(thread)

- Root Comments:

- Root comments are identified as those with no `parent\_id` or a missing `parent\_id`.

- Example:

python

root\_comments = df[df['parent\_id'].isna()]['entry\_id'].tolist()

1.5 Dataset Augmentation

- Reconstructed Threads:

- A new column, `reconstructed\_thread`, is added to the dataset to store the full text of each thread.

- Example:

python

df["reconstructed\_thread"] = df["entry\_id"].apply(lambda x: get\_thread(x) if x in root\_comments else None)

2. Model Training and Summarization

2.1 Summarization Model

- Model: A pre-trained T5-base model is used for summarization.

- Implementation:

- The Hugging Face `pipeline` API is used to load the model and generate summaries.

- Example:

python

summarizer = pipeline("summarization", model="t5-base", device=0)

2.2 Summarization Process

- Input Truncation:

- Threads are truncated to 512 tokens to fit the model's input size.

- Example:

python

text = " ".join(text.split()[:512])

- Dynamic Summary Lengths:

- Summary lengths are dynamically adjusted based on the input text's word count.

- Example:

python

word\_count = len(text.split())

max\_len = min(150, int(0.75 \* word\_count))

min\_len = min(5, int(0.3 \* word\_count))

- Summary Generation:

- Summaries are generated for each thread using the `summarizer` pipeline.

- Example:

python

summary = summarizer(text, max\_length=max\_len, min\_length=min\_len, do\_sample=False)[0]['summary\_text']

2.3 Error Handling

- Short Texts:

- Texts with fewer than 5 words are skipped to avoid generating low-quality summaries.

- Exception Handling:

- Errors during summarization (e.g., model failure) are caught to prevent runtime crashes.

3. Context Mismatch Detection

3.1 Semantic Similarity Model

- Model: A pre-trained SentenceTransformer model (`all-MiniLM-L6-v2`) is used to compute semantic similarity.

- Implementation:

- Example:

python

similarity\_model = SentenceTransformer("all-MiniLM-L6-v2")

3.2 Similarity Calculation

- Embedding Generation:

- Embeddings are generated for each comment and its parent comment.

- Example:

python

embeddings = similarity\_model.encode([comment, parent\_comment], convert\_to\_tensor=True)

- Cosine Similarity:

- Cosine similarity is computed between the embeddings.

- Example:

python

similarity = util.pytorch\_cos\_sim(embeddings[0], embeddings[1]).item()

3.3 Thresholding

- Context Mismatch:

- A similarity score below 0.5 is flagged as a context mismatch.

- Example:

python

return similarity < 0.5

4. Evaluation Metrics

4.1 BLEU Score

- Measures n-gram overlap between the summary and the original thread.

- Example:

python

bleu\_scores.append(sentence\_bleu([reference], candidate))

4.2 ROUGE Score

- Computes recall-oriented metrics (ROUGE-1, ROUGE-2, ROUGE-L) for summary quality.

- Example:

python

rouge = scorer.score(row['reconstructed\_thread'], row['summary'])

4.3 Perplexity

- Measures the fluency of the generated summaries using a GPT-2 model.

- Example:

python

perplexity = torch.exp(torch.tensor(loss)).item()

4.4 Semantic Similarity

- Measures the semantic alignment between summaries and original threads.

- Example:

python

similarity = util.pytorch\_cos\_sim(embeddings[0], embeddings[1]).item()