Assignment: LLM-Based Semantic Re-Ranking

Overview

Search engines and question-answering systems often retrieve many potentially relevant documents for a user's query. Traditional ranking uses simple lexical overlap or embedding similarity, but **semantic re-ranking** (using a **Large Language Model (LLM)** to judge relevance) can dramatically improve precision.

In this assignment, you'll build a small **LLM-powered re-ranking system**.
You'll receive a dataset of *queries* and *candidate passages* that were retrieved using a baseline embedding model.
Your task is to use an LLM API (OpenAI, Anthropic, Hugging Face, etc.) to assign *relevance scores* and produce a new ranked order.
Then you'll evaluate how well your ranking matches the provided "gold" relevance labels.

Learning Outcomes

By the end of this assignment, you will be able to:

- **Integrate** an external API into a Python program and handle rate limits or errors.
- **Design effective prompts** to elicit structured numeric judgments from an LLM.
- **Compute ranking metrics** (Precision@k, Recall@k, nDCG) to evaluate retrieval quality.
- **Analyze** how LLM-based judgments differ from baseline lexical or embedding-based rankings.

Provided Materials

You will receive:

File	Description
`queries_candidates.csv`	Dataset containing queries and candidate passages.
`rerank_starter.py`	Template code showing data loading and metric setup.

Each row of the CSV looks like:

query_id	query_text	candidate_id	candidate_text	baseline_rank	baseline_score	gold_label
1	What is reinforcement learning?	a	Reinforcement learning involves an agent...	1	0.72	1
1	What is reinforcement learning?	b	Unsupervised learning groups unlabeled data...	2	0.65	0
1	What is reinforcement learning?	c	Reward-based feedback mechanisms...	3	0.60	1

- `baseline_rank` and `baseline_score` come from an embedding model.
- `gold_label` = 1 means relevant; 0 means not relevant.
- Each query has about 5–10 candidate passages.

Your Tasks

1. Load and Explore the Data

Inspect the structure of the dataset:

```
import pandas as pd
df = pd.read_csv("queries_candidates.csv")
df.head()
```

Confirm how many unique queries and candidates there are.

2. Query the LLM for Relevance Scores

For each `(query_text, candidate_text)` pair, ask the model to rate relevance.

Example prompt

```
You are evaluating search results.

Query: "What is reinforcement learning?"
Candidate passage:
"Reinforcement learning involves an agent that interacts with an environment to maximize rew
```

`Rate how relevant the candidate passage is to the query on a scale from 0 (not relevant) to 5 (highly relevant).`

Parse the numeric response into a float `llm_score`.
Store all results in a new column in your DataFrame.

Tips

- Be explicit: ask the model to "Respond only with a number."
- Include retries or sleep to handle rate limits.
- Log costs and latency (tokens or time) if your API provides them.

3. Re-Rank the Candidates

Sort candidates by the LLM-assigned score within each query.

```
reranked = (
    df.sort_values(["query_id", "llm_score"], ascending=[True, False])
)
```

4. Evaluate the Rankings

Compute ranking metrics using `gold_label` as ground truth.

Example starter functions:

```
from sklearn.metrics import ndcg_score
import numpy as np

def precision_at_k(y_true, k=3):
    return np.sum(y_true[:k]) / k

def recall_at_k(y_true, k=3):
    return np.sum(y_true[:k]) / np.sum(y_true)

for qid, group in df.groupby("query_id"):
    y_true = group.sort_values("baseline_rank")["gold_label"].to_numpy()
    y_pred = group.sort_values("baseline_rank")["baseline_score"].to_numpy()
    baseline_ndcg = ndcg_score([y_true], [y_pred])

    y_pred_llm = group.sort_values("llm_score", ascending=False)["llm_score"].to_numpy()
    ndcg_llm = ndcg_score([y_true], [y_pred_llm])

    print(f"Query {qid}: baseline nDCG={baseline_ndcg:.3f}, LLM nDCG={ndcg_llm:.3f}")
```

Compare your LLM-based ranking to the baseline.

5. Analyze and Reflect

In your written report, discuss: - What prompt you used and why. - Any variations you tested (e.g., "rate 1–10," "choose top-3," etc.). - Where the LLM improved or failed. - How cost, latency, or API constraints affected your design.

Deliverables

File	Description
`rerank.py`	Code that loads data, calls the API, and outputs LLM scores.
`results.csv`	Contains `query_id, candidate_id, llm_score, llm_rank`.
`report.pdf`	1-2 pages describing prompt design, evaluation, and results.

Evaluation Rubric

Criterion	Excellent (A)	Proficient (B)	Developing (C)	Incomplete (D/F)
API Integration	Robust, modular API use with retry and error handling	Functional API use	Basic calls, limited error handling	Fails or missing
Prompt Design	Structured, clear, and tested variations	Functional, consistent	Poorly structured prompt or inconsistent parsing	Missing or unusable
Ranking Evaluation	Implements multiple metrics (Precision@k, Recall@k, nDCG) and compares baselines	Implements at least one metric correctly	Partial or incorrect metrics	No evaluation
Analysis & Insight	Clear, thoughtful discussion of results and limitations	Some interpretation of results	Minimal analysis	No analysis

Criterion	Excellent (A)	Proficient (B)	Developing (C)	Incomplete (D/F)
Code Quality	Clean, reproducible, and well-documented	Readable, minor issues	Hard to follow or unstructured	Not runnable

Optional Extensions

- **Prompt Variants:** try adding explanations ("give a reason and a score") and see if that changes consistency.
- **Majority Vote:** query the model multiple times and average the scores.
- **Open vs. Closed Models:** compare results between two APIs.
- **Few-Shot Example Prompting:** include one or two sample ratings inside the prompt to reduce variance.

Submission

Submit your Python code, results CSV, and report as a compressed `.zip` or via your class GitHub repository.
Your report should be concise and visually clear: figures, tables, or short excerpts are welcome.