Information Retrieval Chatbot - README

Overview

This project implements an Information Retrieval (IR) Chatbot designed to answer queries across multiple topics and handle both factual Q&A and general chit-chat interactions. It integrates:

- A topic classification pipeline (Naive Bayes, Logistic Regression, and an optional BERT-based classifier).
- A Wikipedia-based dataset scraped via a BFS approach.
- Semantic retrieval using embeddings (`clips/mfaq`) for improved relevance.
- An abstractive summarization pipeline for generating concise answers.
- A chit-chat model (Blender Bot) for off-topic and conversational queries.
- Optionally, a UI with analytics for visualizing query distribution and topic-wise statistics.

Team Details:

Lakshmi Bhavana Thalapaneni	(G01452929 - ltalapan@gmu.edu)
Aradhya Jain	(G01462086 - ajain30@gmu.edu)
Akhila Palempati	(G01448123 - apalempa@gmu.edu)

System Requirements

- **Operating System:** Linux, macOS, or Windows
- **Python Version:** Python 3.8 or above
- **Hardware:**
  - CPU: Modern multi-core CPU recommended
  - GPU: Recommended if training or fine-tuning large models (e.g., BERT, custom summarization). Otherwise, CPU inference may suffice, albeit slower.
- **Memory:** At least 16GB RAM recommended (depending on dataset size and models used).

Dependencies

- **Python Libraries:**
  - `transformers` (Hugging Face)
  - `datasets` or `evaluate` (for metrics)
  - `sentence-transformers` (for `clips/mfaq` embeddings)
  - `scikit-learn` (for classification)
  - `Flask` (for running microservices)
  - `requests`, `BeautifulSoup` (if re-running the scraper, ensure network access)
  - `pandas`, `numpy` (general data processing)

Install all Python dependencies pip command.

Dataset

- **Source:** Wikipedia, scraped via the MediaWiki API.
- **Topics:** Health, Environment, Technology, Economy, Entertainment, Sports, Politics, Education, Travel, Food.
- **Size:** ~50,000 documents total (5,000+ per topic).

The dataset is provided in `ir_data.json`:

- Keys: `title`, `revision_id`, `summary`, `description`, `url`, `topic`.
- This file should be placed in the project root directory.

Code Files

- **`web_crawler.py`**: Scrapes Wikipedia to build the dataset using BFS.
- **`ir_data.json`**: The resulting dataset with documents categorized into topics.
- **`classifier/`**:
  - `naive_bayes.py`, `logistic_regression.py`: Training and saving classic ML classifiers for topic classification.
  - `text_classifier.py`: Loads trained classifiers and returns topic predictions for queries.
- **`chitchat/chitchat.py`**: Flask service handling chit-chat queries with Blender Bot.
- **`retrieval/doc_retriever.py`**: Retrieves top-k documents based on semantic embeddings for a given topic.
- **`summarization/summary_generator.py`**: Summarizes retrieved text using a transformer-based summarization pipeline.
- **`utils/utils.json`**: Contains mappings of IPs/ports for microservices.
- **Custom Model Training Code (e.g., `train_transformer_classifier.py`)**: Scripts for fine-tuning a BERT-based model or custom summarizer over `ir_data.json`.
- **`index.py`**: The main orchestrator, receiving user queries, calling the classifier, routing to either chit-chat or retrieval+summarization, and returning final answers.

Running the Project

1. **Data Setup:**

   Ensure `ir_data.json` is in the project root. If you need a new dataset, run `web_crawler.py` (requires network access to Wikipedia). This may take time and resources. (~18 hours).

2. **Train or Load Classifiers:**

   Run `logistic_regression.py` and `naive_bayes.py` once to generate `.pkl` models. Then `text_classifier.py` loads these models at runtime.

3. **Start Microservices:**
    - Classifier:
    - `cd classifier`
    - `python text_classifier.py`
    - Chitchat:
    - `cd ../chitchat`
    - `python chitchat.py`
    - Retrieval (for each topic or a combined retriever):
    - `cd ../retrieval`
    - `python doc_retriever.py`
    - Summarization:
    - `cd ../summarization`
    - `python summary_generator.py`

    Ensure each service starts successfully and note their ports. Adjust `utils.json` or code to point to correct IP:port values if needed.

4. **Main Orchestrator:** Start `index.py` from the project root:

    - `cd ../`
    - `python index.py`

    This acts as the entry point. It receives queries (POST requests), calls classifier, then retrieval and summarization or chit-chat services, and returns the final answer.

5. **Optional: Frontend/UI:** If a web-based UI is implemented, follow its setup instructions. Typically:

    - `cd frontend`
    - `npm install`
    - `npm start`

    Access the UI via `http://localhost:3000` or the configured host/port. The UI interacts with `index.py` backend services.

Example Usage

- Using `POSTMAN` or `curl`, send a JSON POST request to `index.py`:
-
```
curl -X POST -H "Content-Type: application/json" \
      -d '{"input":"What is mental health?", "topics":[]}' \
      http://127.0.0.1:5000
```

  The system should respond with a JSON answer containing a summarized explanation.

- For a chit-chat query:
-
```
curl -X POST -H "Content-Type: application/json" \
     -d '{"input":"Tell me a joke", "topics":[]}' \
     http://127.0.0.1:5000
```

Expect a conversational response from the Blender Bot model.

Extending the Project

- **Change Thresholds:**

  Adjust classification threshold or summarization parameters in config files for quick tuning.

- **Retrain Models:**

  Run the custom training scripts (like `custom_model.py`) to improve classification accuracy if you have more resources.

- **Update Dataset:**

  Rerun `web_crawler.py` to expand coverage or refresh data. Consider incremental updates and integrating APIs for live data feeds.

Troubleshooting

- **Service Connection Errors:** Check `utils.json` and ensure correct IPs/ports for each microservice.
- **Slow Summarization:** Reduce `num_beams` or `max_length` in `summary_generator.py`.
- **Memory Issues:** Lower batch sizes for embedding computation or switch to CPU-only models if GPU memory is insufficient.

Conclusion

This README should guide you through setting up, running, and experimenting with the IR Chatbot. Adjust parameters, train models, and explore new topics to continue improving the system's accuracy, responsiveness, and user satisfaction.

Links:

Due to the size of the dataset and code files, here's the GMU SharePoint link.

https://github.com/AradhyaJain/Information-Retrieval-Chatbot

CS747 Project

Group 8 Wikipedia Information Retrieval Chatbot.pptx