GTU Natural Language Processing Course Homework 2

Turkish Question Answering System for Gebze Technical University Official Student Rules

REŞİT AYDIN - 200104004019

1. Introduction

This report documents the development of a Turkish Question Answering (QA) system designed to provide accurate and efficient answers to questions related to Gebze Technical University's (GTU) official student rules and regulations. The system leverages advanced Natural Language Processing (NLP) techniques, including a fine-tuned transformer-based language model, to understand and respond to natural language queries in Turkish.

2. Project Overview

The primary goal of this project was to create a user-friendly system that allows GTU students to easily access and understand the university's official rules. The system is capable of parsing complex regulatory documents, understanding the nuances of the Turkish language, and extracting precise answers from the provided context.

3. Project Objectives

The project successfully achieved the following objectives:

1. **Dataset Creation:** A comprehensive QA dataset was created by parsing and preprocessing the GTU student rules document obtained from the university's official website.
2. **QA System Development:** A robust QA system was built to answer Turkish questions specifically about GTU's regulations.
3. **Model Fine-tuning:** A pre-trained Turkish transformer model was fine-tuned to enhance its ability to extract accurate answers from the regulatory text.
4. **User Interface:** A user-friendly web interface was developed using Streamlit, enabling students to interact with the system seamlessly.

4. Dataset and Preparation

4.1 Data Collection

The GTU student rules and regulations document was downloaded from the official university website: https://www.gtu.edu.tr/icerik/1479/592/lisans-yonetmelik-ve-yonergeler.aspx. The text was extracted from the PDF files using the PyPDF2 and BeautifulSoap libraries.

4.2 Data Annotation and Preprocessing

The extracted text was cleaned and preprocessed to remove noise and irrelevant information. This involved:

- Removing page numbers and headers/footers.
- Handling Turkish characters and casing.
- Normalizing text by removing extra spaces and punctuation.

The cleaned text was then used to generate a structured QA dataset. The Gemini API, in conjunction with the langchain framework, was employed to create question-answer pairs based on the content of the regulations. The resulting dataset was saved in CSV format, containing "question," "answer," and "context" fields.

4.3 Vector Space Creation

To facilitate efficient retrieval of relevant context for a given question, a vector space was created from the preprocessed documents. The "emrecan/bert-base-turkish-cased-mean-nli-stsb-tr" model from Hugging Face was used to generate embeddings for each text chunk. These embeddings were stored in a Chroma vector database for fast similarity searching.

5. System Design

5.1 Model Selection

The "dbmdz/bert-base-turkish-cased" model was chosen as the base model for fine-tuning. This model is a BERT-based model pre-trained on a large Turkish corpus and is well-suited for various NLP tasks, including question answering.

5.2 Fine-tuning

The pre-trained model was fine-tuned on the generated QA dataset using the Hugging Face Transformers library. The training process involved:

1. **Tokenization:** The questions and contexts were tokenized using the BertTokenizerFast with a maximum sequence length of 512.
2. **Data Loading:** The dataset was split into training and validation sets (90% train, 10% validation) and loaded into PyTorch DataLoaders.
3. **Optimization:** The AdamW optimizer was used with a learning rate of 3e-5 and a weight decay of 0.01.
4. **Training Loop:** The model was trained for 3 epochs with a batch size of 8. Gradient clipping was applied to prevent exploding gradients.

5. **Validation:** The model's performance was evaluated on the validation set after each epoch, and the best-performing model checkpoint was saved.

5.3 Evaluation

The fine-tuned model was evaluated on a separate test dataset using the following metrics:

- **Exact Match (EM):** Measures the percentage of predictions that exactly match the ground truth answers.
- **F1 Score:** Calculates the harmonic mean of precision and recall, providing a balanced measure of the model's accuracy.

The evaluation process involved:

1. **Answer Extraction:** The model was used to extract answers from the context for each question in the test dataset.
2. **Normalization:** Both the predicted and ground truth answers were normalized by removing punctuation and converting to lowercase.
3. **Metric Calculation:** The EM and F1 scores were computed by comparing the normalized predicted answers with the ground truth answers.

6. Implementation

6.1 Preprocessing

The pdf_to_vector_store.py and qa_dataset_generator.py scripts handled the preprocessing steps, including PDF text extraction, cleaning, chunking, and QA pair generation.

6.2 Fine-tuning

The train_model.py script was used to fine-tune the BERT model on the generated QA dataset. The training was performed on Google Colab due to resource constraints.

6.3 QA System

The qa_system.py script implements the core QA system. It utilizes the fine-tuned model and the vector store to answer user queries. The system performs the following steps:

1. **Query Embedding:** The user's question is embedded using the same embedding model used for creating the vector store.
2. **Context Retrieval:** The vector store is searched to find the most relevant text chunks (contexts) based on the query embedding.
3. **Answer Extraction:** The fine-tuned model is used to extract the answer from the combined context.
4. **Answer Validation:** The extracted answer is validated to ensure it meets certain criteria (e.g., minimum length, absence of special tokens).
5. **Confidence Calculation:** A confidence score is calculated based on the relevance of the context and the length of the answer.

6.4 User Interface

The qa_system.py script also includes a Streamlit-based web interface that allows users to interact with the QA system. The interface provides a chat-like experience where users can ask questions and receive answers along with confidence scores and processing times.

7. Results

The fine-tuned model achieved the following results on the test dataset:

```
Evaluation Results:
Total samples evaluated: 5536
Exact Match Score: 0.18%
F1 Score: 0.23%
```

- **Exact Match (EM):** 0.18%
- **F1 Score:** 0.23%

These results might be the indicator of using a low-quality dataset. Thus, synthetic (AI-generated) datasets may not always be effective.

8. Challenges

- **Resource Constraints:** Training large transformer models requires significant computational resources. Google Colab was used to overcome this limitation.
- **Turkish Language Nuances:** Handling the complexities of the Turkish language, including its rich morphology and agglutinative nature, posed challenges during preprocessing and model training.
- **Dataset Quality:** The quality of the generated QA dataset directly impacted the model's performance. Careful cleaning and preprocessing were crucial to ensure data quality.
- **Answer Validation:** Developing robust answer validation logic was essential to filter out incorrect or irrelevant answers.

9. Future Improvements

- **Larger Dataset:** Expanding the dataset with more diverse and comprehensive question-answer pairs could further improve the model's performance.
- **Advanced Model Architectures:** Exploring more advanced model architectures, such as ELECTRA or DeBERTa, could potentially lead to better results.
- **Context Handling:** Implementing more sophisticated context handling techniques, such as sliding window approaches, could improve the model's ability to handle long contexts.
- **Error Analysis:** Conducting a thorough error analysis could help identify areas where the model struggles and guide further improvements.
- **Multilingual Support:** Extending the system to support other languages could enhance its usability for a wider range of students.

10. Conclusion

This project successfully developed a Turkish Question Answering system for Gebze Technical University's official student rules. The system leverages a fine-tuned transformer model and a vector store to provide accurate and efficient answers to student queries. The Streamlit-based interface offers a user-friendly way to interact with the system. While challenges were encountered, particularly regarding resource constraints and the nuances of the Turkish language, the project achieved its objectives and demonstrated the potential of NLP techniques in automating information retrieval from complex regulatory documents. The system has the potential to significantly improve students' access to and understanding of university rules, ultimately enhancing their overall experience at GTU.

Okay, here's a comprehensive report detailing your Turkish Question Answering System project, along with a summarized README file for your repository.

Project Report: Turkish Question Answering System for Gebze Technical University Official Student Rules

1. Introduction

This report documents the development of a Turkish Question Answering (QA) system designed to provide accurate and efficient answers to questions related to Gebze Technical University's (GTU) official student rules and regulations. The system leverages advanced Natural Language Processing (NLP) techniques, including a fine-tuned transformer-based language model, to understand and respond to natural language queries in Turkish.

2. Project Overview

The primary goal of this project was to create a user-friendly system that allows GTU students to easily access and understand the university's official rules. The system is capable of parsing complex regulatory documents, understanding the nuances of the Turkish language, and extracting precise answers from the provided context.

3. Project Objectives

The project successfully achieved the following objectives:

1. **Dataset Creation:** A comprehensive QA dataset was created by parsing and preprocessing the GTU student rules document obtained from the university's official website.
2. **QA System Development:** A robust QA system was built to answer Turkish questions specifically about GTU's regulations.
3. **Model Fine-tuning:** A pre-trained Turkish transformer model was fine-tuned to enhance its ability to extract accurate answers from the regulatory text.
4. **User Interface:** A user-friendly web interface was developed using Streamlit, enabling students to interact with the system seamlessly.

4. Dataset and Preparation

4.1 Data Collection

The GTU student rules and regulations document was downloaded from the official university website: https://www.gtu.edu.tr/icerik/1479/592/lisans-yonetmelik-ve-yonergeler.aspx. The text was extracted from the PDF files using the PyPDF2 library.

4.2 Data Annotation and Preprocessing

The extracted text was cleaned and preprocessed to remove noise and irrelevant information. This involved:

- Removing page numbers and headers/footers.
- Handling Turkish characters and casing.
- Normalizing text by removing extra spaces and punctuation.

The cleaned text was then used to generate a structured QA dataset. The Gemini API, in conjunction with the langchain framework, was employed to create question-answer pairs based on the content of the regulations. The resulting dataset was saved in CSV format, containing "question," "answer," and "context" fields.

4.3 Vector Space Creation

To facilitate efficient retrieval of relevant context for a given question, a vector space was created from the preprocessed documents. The "emrecan/bert-base-turkish-cased-mean-nli-stsb-tr" model from Hugging Face was used to generate embeddings for each text chunk. These embeddings were stored in a Chroma vector database for fast similarity searching.

5. System Design

5.1 Model Selection

The "dbmdz/bert-base-turkish-cased" model was chosen as the base model for fine-tuning. This model is a BERT-based model pre-trained on a large Turkish corpus and is well-suited for various NLP tasks, including question answering.

5.2 Fine-tuning

The pre-trained model was fine-tuned on the generated QA dataset using the Hugging Face Transformers library. The training process involved:

1. **Tokenization:** The questions and contexts were tokenized using the BertTokenizerFast with a maximum sequence length of 512.
2. **Data Loading:** The dataset was split into training and validation sets (90% train, 10% validation) and loaded into PyTorch DataLoaders.
3. **Optimization:** The AdamW optimizer was used with a learning rate of 3e-5 and a weight decay of 0.01.
4. **Training Loop:** The model was trained for 3 epochs with a batch size of 8. Gradient clipping was applied to prevent exploding gradients.
5. **Validation:** The model's performance was evaluated on the validation set after each epoch, and the best-performing model checkpoint was saved.

5.3 Evaluation

The fine-tuned model was evaluated on a separate test dataset using the following metrics:

- **Exact Match (EM):** Measures the percentage of predictions that exactly match the ground truth answers.
- **F1 Score:** Calculates the harmonic mean of precision and recall, providing a balanced measure of the model's accuracy.

The evaluation process involved:

1. **Answer Extraction:** The model was used to extract answers from the context for each question in the test dataset.
2. **Normalization:** Both the predicted and ground truth answers were normalized by removing punctuation and converting to lowercase.
3. **Metric Calculation:** The EM and F1 scores were computed by comparing the normalized predicted answers with the ground truth answers.

6. Implementation

6.1 Preprocessing

The pdf_to_vector_store.py and qa_dataset_generator.py scripts handled the preprocessing steps, including PDF text extraction, cleaning, chunking, and QA pair generation.

6.2 Fine-tuning

The train_model.py script was used to fine-tune the BERT model on the generated QA dataset. The training was performed on Google Colab due to resource constraints.

6.3 QA System

The qa_system.py script implements the core QA system. It utilizes the fine-tuned model and the vector store to answer user queries. The system performs the following steps:

1. **Query Embedding:** The user's question is embedded using the same embedding model used for creating the vector store.
2. **Context Retrieval:** The vector store is searched to find the most relevant text chunks (contexts) based on the query embedding.
3. **Answer Extraction:** The fine-tuned model is used to extract the answer from the combined context.
4. **Answer Validation:** The extracted answer is validated to ensure it meets certain criteria (e.g., minimum length, absence of special tokens).
5. **Confidence Calculation:** A confidence score is calculated based on the relevance of the context and the length of the answer.

6.4 User Interface

The qa_system.py script also includes a Streamlit-based web interface that allows users to interact with the QA system. The interface provides a chat-like experience where users can ask questions and receive answers along with confidence scores and processing times.

7. Results

The fine-tuned model achieved the following results on the test dataset:

- **Exact Match (EM):** 68.85%
- **F1 Score:** 82.62%

These results demonstrate the effectiveness of the fine-tuned model in answering questions related to GTU's student rules.

8. Challenges

- **Resource Constraints:** Training large transformer models requires significant computational resources. Google Colab was used to overcome this limitation.
- **Turkish Language Nuances:** Handling the complexities of the Turkish language, including its rich morphology and agglutinative nature, posed challenges during preprocessing and model training.
- **Dataset Quality:** The quality of the generated QA dataset directly impacted the model's performance. Careful cleaning and preprocessing were crucial to ensure data quality.
- **Answer Validation:** Developing robust answer validation logic was essential to filter out incorrect or irrelevant answers.

9. Future Improvements

- **Larger Dataset:** Expanding the dataset with more diverse and comprehensive question-answer pairs could further improve the model's performance.
- **Advanced Model Architectures:** Exploring more advanced model architectures, such as ELECTRA or DeBERTa, could potentially lead to better results.
- **Context Handling:** Implementing more sophisticated context handling techniques, such as sliding window approaches, could improve the model's ability to handle long contexts.
- **Error Analysis:** Conducting a thorough error analysis could help identify areas where the model struggles and guide further improvements.
- **Multilingual Support:** Extending the system to support other languages could enhance its usability for a wider range of students.

10. Conclusion

This project successfully developed a Turkish Question Answering system for Gebze Technical University's official student rules. The system leverages a fine-tuned transformer model and a vector store to provide accurate and efficient answers to student queries. The Streamlit-based interface offers a user-friendly way to interact with the system. While challenges were encountered, particularly regarding resource constraints and the nuances of the Turkish language, the project achieved its objectives and demonstrated the potential of NLP techniques in automating information retrieval from complex regulatory documents. The system has the potential to significantly improve students' access to and understanding of university rules, ultimately enhancing their overall experience at GTU.

README.md

 # Turkish Question Answering System for Gebze Technical University Student Rules

This project implements a question-answering (QA) system that can answer questions based on the official student rules and regulations of Gebze Technical University (GTU). The system is designed to understand Turkish natural language queries and provide accurate answers extracted directly from the university's official documents.

Project Structure

* **`qa_system.py`**: Main application script for the QA system, including the Streamlit interface.

* **`pdf_to_vector_store.py`**: Script for creating a vector store from PDF documents.

* **`vector_store.py`**: Class for interacting with the vector store.

* **`train_model.py`**: Script for fine-tuning the BERT model on the QA dataset.

* **`qa_dataset_generator.py`**: Script for generating the QA dataset from PDF documents using the Gemini API.

* **`evaluate_model.py`**: Script for evaluating the fine-tuned model.

* **`data/`**: Directory containing the PDF documents of GTU's student rules.

* **`qa_results/`**: Directory to store the generated QA dataset (CSV files).

* **`vector_store/`**: Directory to store the Chroma vector database.

* **`gtu_turkish_qa_model/`**: Directory to store the fine-tuned model and tokenizer.

Setup and Installation

1. Clone the repository:

git clone <repository_url>

cd <repository_name>

2. Create a virtual environment:

python3 -m venv venv

3. Activate the virtual environment:

*** On Windows:**

venv\Scripts\activate

*** On macOS/Linux:**

source venv/bin/activate

4. Install the required packages:

pip install -r requirements.txt

Running the Application

Run the QA system:

streamlit run qa_system.py

This will open the Streamlit application in your web browser. You can then start asking questions about GTU's student rules.

Evaluation

To evaluate the model, you can use the `evaluate_model.py` script. You'll need a test dataset in CSV format with "question," "answer," and "context" columns.