Exemplar Answer Generation Project

This project developed a Python application that leverages **OpenAI's GPT models** (GPT-4o mini) to automatically generate high-quality exemplar answers for educational assessment questions. This project uses the training data (`data/cura-llm-training-data.json`) and OpenAI API access provided by Cura Education.

Background

Cura Education hosts an extensive library of online courses designed for student engagement and learning. Within these courses, students complete various tasks and submit responses to assessment questions. Exemplar answers are crucial for teachers as they provide clear benchmarks for evaluating student submissions.

Project Overview

- Processes educational task content, questions, and assessment rubrics
- Generates aligned exemplar answers using OpenAI API
- Includes evaluation metrics for answer quality
- Contains automated testing suite
- Supports educational assessment workflows

Key Features

- 🤖 Leverages OpenAI's GPT-4o mini model for answer generation
- 📊 Comprehensive data analysis and preprocessing pipeline
- 🎯 Multiple evaluation metrics for quality assessment
- 🧪 Broad test coverage with pytest

Setup and Installation

Prerequisites

- **Python 3.9+**
- Conda environment manager (recommended)

Environment Setup

1. Clone the repository

```
git clone [https://github.com/WendaZhang08/Exemplar-Answer-Generation-with-OpenAI-API.git]
cd Exemplar-Answer-Generation-with-OpenAI-API
```

2. Create and activate conda environment

```
conda create -n exemplar-env python=3.9
conda activate exemplar-env
```

3. Install required packages

```
pip install -r requirements.txt
```

Running the Project

1. **Data Analysis**: Open and run `Section1_Data_Analysis.ipynb` to perform data analysis of the training dataset.
2. **OpenAI Integration**: Open and run `Section2_OpenAI_Integration.py` to perform data preprocessing, exemplar answer generation and performance evaluation.
3. **Testing**: Execute the test suite command below, which will run all test cases and display detailed results.

```
pytest test_openai_handler.py -v
```

Project Structure

1. Data Analysis Section (Section1_Data_Analysis.ipynb)

This notebook contains comprehensive analysis of the training dataset used for generating exemplar answers. The analysis is structured as follows:

Environment Setup

- Initial setup of required dependencies including **pandas**, **numpy**, **matplotlib**, and **seaborn**
- Configuration of visualization settings and data processing utilities

Data Loading and Processing

- Loading of training data from JSON format
- Basic data validation and structure examination
- Conversion to pandas DataFrame for analysis

Exploratory Data Analysis

- Statistical analysis of key characteristics:
  - Question length distribution (majority between 0-200 characters)
  - Answer length patterns (concentrated between 100-400 characters)
  - Task content analysis (ranging from 0 to 40,000 characters)
  - Rubric scoring distribution (focused on 2-3 point scales)

Visualization

- Distribution plots for:
  - Question lengths
  - Answer lengths
  - Task content lengths
  - Rubric total scores

Key Findings

- Questions tend to be concise with occasional longer descriptions
- Exemplar answers follow a balanced length distribution
- Task content varies significantly in length based on topic complexity
- Assessment rubrics show standardized scoring patterns

This analysis provides crucial insights for:

- Understanding data patterns and characteristics
- Identifying potential preprocessing requirements
- Informing the development of answer generation strategies
- Establishing baseline metrics for evaluation

The structured analysis aids in developing a solution for generating appropriate exemplar answers that align with the existing patterns in the training data.

---

2. OpenAI Integration Section (Section2_OpenAI_Integration.ipynb)

This section implements the core functionality for generating exemplar answers using OpenAI's API, featuring a class hierarchy for API integration and evaluation:

Environment Setup

- Installation of required dependencies:
  - Core ML libraries: **numpy**, **scikit-learn**
  - OpenAI API related: **openai**, **tiktoken**
  - NLP tools: **nltk**, **sentence-transformers**
  - Data processing: **pandas**
  - Visualization: **matplotlib**, **seaborn**

Data Processing

- Loading preprocessed training data
- Data cleaning and standardization
- HTML entity removal
- Text normalization
- Length truncation for efficient processing
- Training/validation data split preparation
- Data formatting for API consumption

Implementation Architecture

1. **Base OpenAI Handler**

- API initialization and token management
- Basic tokenization functionality
- Token usage tracking and statistics

2. **Prompt Handler**

- Multiple prompt template implementations
- Context formatting for API requests
- Template optimization logic

3. **Generation Handler**

- Answer generation functionality
- Token usage optimization

4. **Training Handler**

- Prompt template selection and optimization
- Best example selection for few-shot learning
- Training process management

5. **Evaluation Handler**

- Comprehensive evaluation metrics implementation
    - Content relevance scoring
    - Rubric alignment assessment
    - Semantic similarity calculation
    - Answer structure analysis
    - Reasoning depth evaluation
- K-fold cross-validation implementation
- Correlation analysis between metrics

Evaluation Metrics

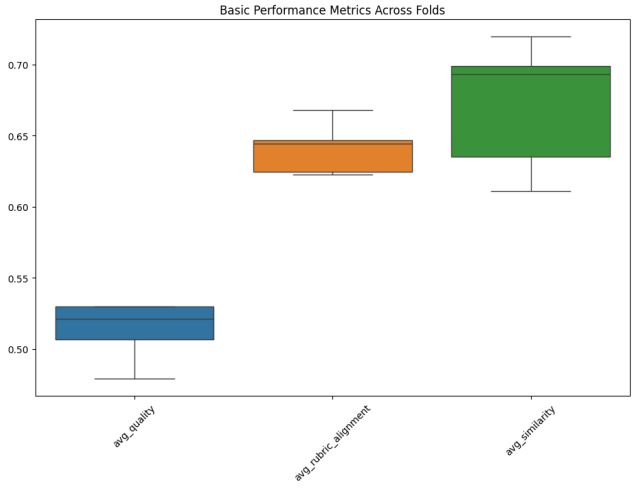
The evaluation system includes multiple dimensions:

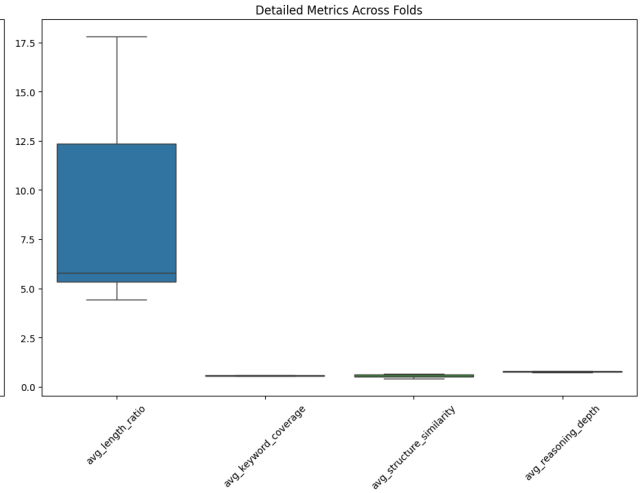
- **Basic Metrics**
    - Quality scoring (0-1 scale)
    - Rubric alignment (0-1 scale)
    - Semantic similarity with reference answers
- **Detailed Metrics**
    - Length ratio analysis
    - Keyword coverage
    - Structure similarity

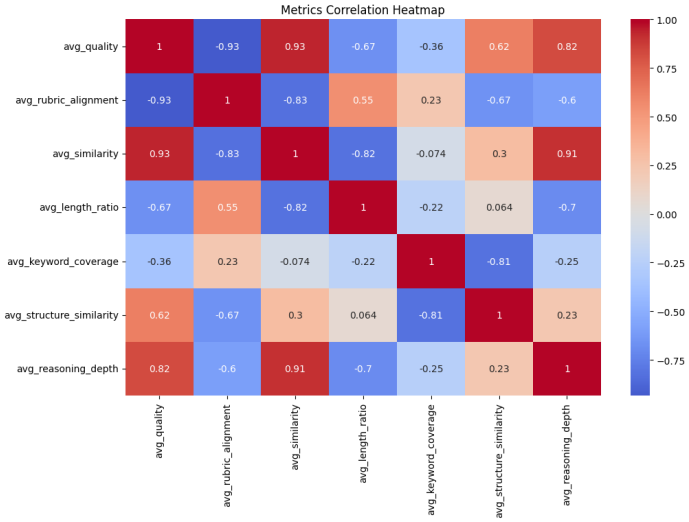
- Reasoning depth assessment

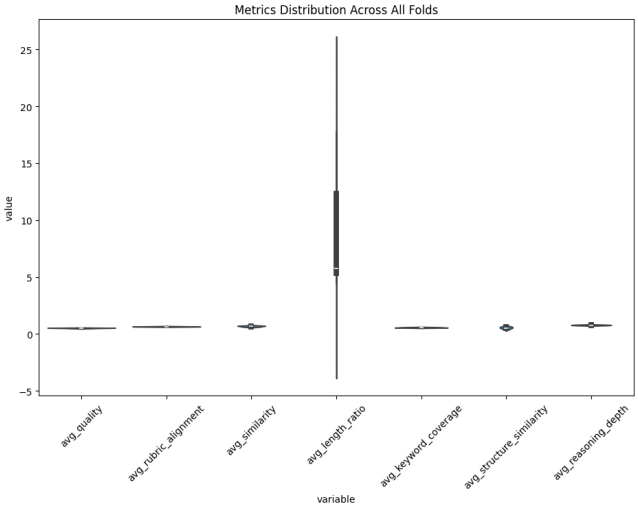
Visualization Components

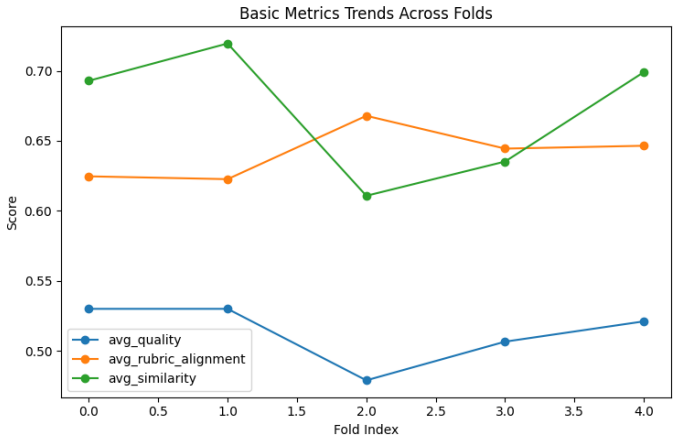
- Performance metrics across folds
- Correlation heatmaps
- Metric distribution analysis
- Cross-fold performance trends

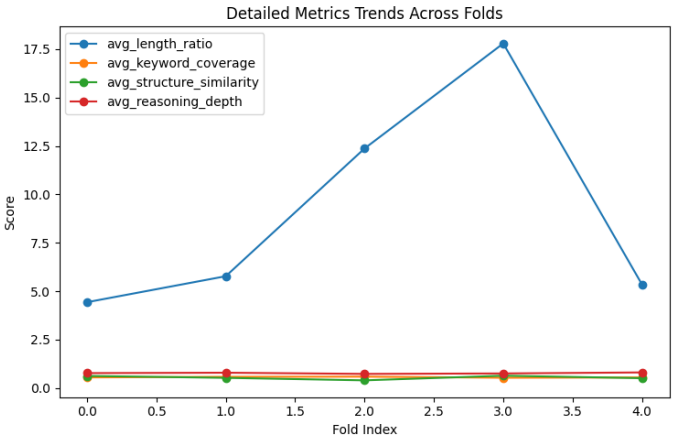












Analysis of Evaluation Results

> [!Note] Full analysis available in Section 3.3 of `Section2_OpenAI_Integration.ipynb`

The evaluation system demonstrates the model's performance with:

- Average Quality Score: **0.513 ± 0.019**
- Rubric Alignment: **0.641 ± 0.017**
- Semantic Similarity: **0.671 ± 0.041**

and:

- Length Ratio: **9.142 ± 5.161**
- Keyword Coverage: **0.559 ± 0.021**
- Structure Similarity: **0.539 ± 0.087**
- Reasoning Depth: **0.770 ± 0.027**

The evaluation reveals several key insights about the model's performance:

- **Robust Performance Metrics**:
    - Strong semantic alignment (**0.671 ± 0.041**)
    - Consistent rubric adherence (**0.641 ± 0.017**)
    - Stable quality scores (**0.513 ± 0.019**)
- **Generalization Capabilities**:
    - Consistent performance across k-fold validation
    - Stable metrics across different question types
    - Reliable pattern recognition in varied content areas
- **Areas for Optimization**:
    - Length ratio variability needs improvement
    - Trade-off between rubric alignment and natural language fluency
    - Potential for enhanced domain-specific terminology handling
- **Evaluation Framework Strengths**:
    - Multi-dimensional assessment approach
    - Rigorous scoring methodology
    - Comprehensive correlation analysis between metrics

For detailed analysis, refer to Section 3.3: The Analysis of the Evaluation Results.

---

3. Testing Implementation

Python Script Generation (Section2_OpenAI_Integration.py)

The implementation code has been converted from Jupyter notebook to a standalone Python script using `jupyter nbconvert`, making it suitable for automated testing.

Automated Testing (test_openai_handler.py)

Comprehensive test suite using `pytest` framework to validate the OpenAI handler implementation. The testing strategy focuses on four main areas:

1. **Basic Handler Testing**

- Validates proper initialization of the OpenAI handler with API keys
- Tests token counting and management functionality
- Verifies API interaction through mock responses
- Ensures correct token usage tracking and limits

2. **Quality Evaluation Testing**

- Tests the answer quality assessment algorithms through `test_evaluate_example_quality`
- Validates rubric alignment calculations
- Verifies semantic similarity measurements between generated and reference answers
- Tests keyword extraction functionality for content analysis

3. **Edge Case Handling**

- Tests behavior with empty inputs (`test_invalid_input_empty_string`)
- Validates handling of invalid input types
- Verifies response to token limit scenarios
- Tests API error handling mechanisms

4. **Integration Testing**

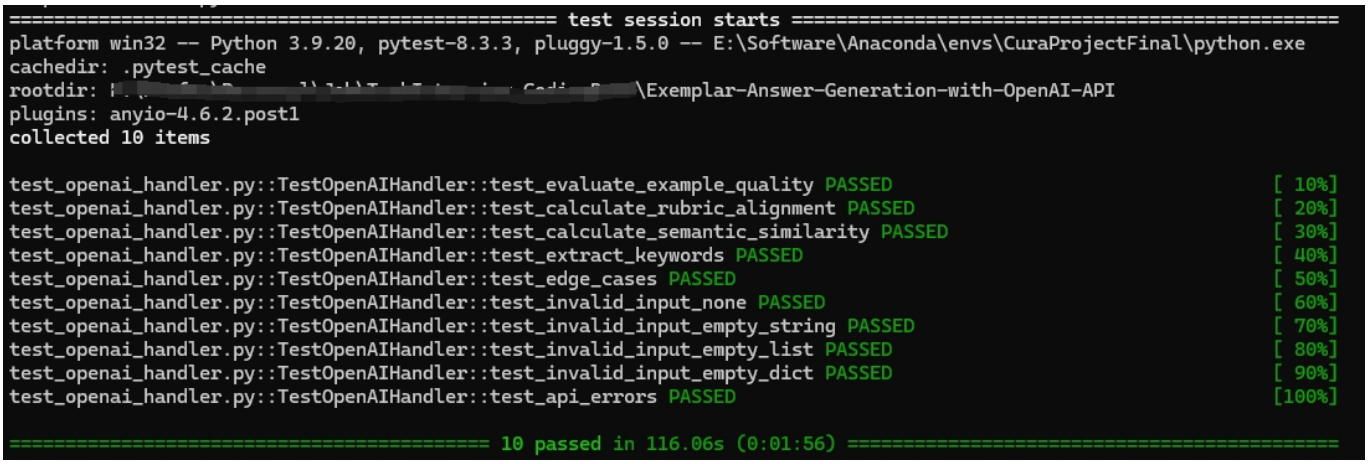
- Tests complete workflow from input processing to answer generation
- Validates prompt template selection logic
- Verifies training process outcomes
- Tests evaluation metrics calculation

Test Framework

The testing framework uses fixtures for test setup:

- `mock_openai_client`: Provides controlled API response simulation
- `sample_data`: Delivers standardized test cases for consistent evaluation
- `handler`: Creates preconfigured handler instances

Error handling validation is implemented through specific test cases that verify the system's response to various error conditions, including API failures and invalid inputs. The testing logic demonstrates the robustness of the implementation and its ability to handle various operational scenarios reliably.



License

This project is licensed under the MIT License - see the LICENSE file for details.

Acknowledgments

- Cura Education for providing the training dataset and OpenAI API access
- OpenAI for their GPT models and API

Contact Information

Author

Damon (Wenda) Zhang

- Master of Information Technology (Artificial Intelligence) @ The University of Melbourne
- Focus Areas: Machine Learning, Natural Language Processing, Website Development

Get in Touch

- ✉ Email: damonzhang0824@gmail.com
- LinkedIn: Wenda Zhang
- GitHub: WendaZhang08
- Location: Melbourne, Australia

Last updated: October 2024