Automated Essay Grading System

Python Implementation Documentation

Analysis of answer.py

November 28, 2025

Abstract

This document provides comprehensive documentation for an automated essay grading system implemented in Python. The system uses natural language processing techniques to compare student answers against teacher-provided reference answers, utilizing TF-IDF vectorization, cosine similarity, and semantic expansion through WordNet.

Contents

1 Introduction

The automated essay grading system analyzes textual student responses and grades them by comparing semantic similarity to reference answers. The implementation leverages multiple NLP techniques including tokenization, part-of-speech tagging, lemmatization, and semantic similarity measures.

1.1 Core Dependencies

- `nltk` - Natural Language Toolkit for text processing
- `wordnet` - Semantic lexical database for synonym/hypernym matching
- `django` - Configuration management
- Standard libraries: `re`, `math`, `logging`

2 Main Classes

2.1 Answer Class

The `Answer` class is the primary component handling student answer evaluation.

2.1.1 Initialization Parameters

The constructor initializes various thresholds and configuration settings:

Key Thresholds

- `dist_threshold = 0.25` - Minimum cosine distance for sentence matching
- `sen_threshold = 0.33` - Single sentence matching threshold
- `multisen_threshold = 0.4` - Multiple sentence matching threshold
- `multisen_matchrate = 0.3` - Rate for multiple sentence matches

2.1.2 Optional Features

- **Synonym Expansion**: Matches semantically similar words (e.g., ”lawyer” $\approx$ ”attorney”)
- **Ancestor Expansion**: Matches hypernyms (e.g., ”professional” for ”lawyer”)
- **Part-of-Speech Tagging**: Improves matching accuracy
- **Grammar Checking**: Provides qualitative feedback
- **Single-Match Prevention**: Prevents token reuse

2.2 ImageAnswer Class

A simplified class for grading image-based questions by comparing selected image points against reference standards.

3 Core Methods

3.1 SentenceAnalysis()

Purpose: Preprocesses text into analyzable sentence structures.

Process:

1. Splits text into sentences using regex patterns
2. Tokenizes each sentence into words
3. Applies POS tagging
4. Lemmatizes words using WordNet
5. Computes TF-IDF weighted term vectors

Example:

```
ans.SentenceAnalysis(
    "The lawyer chased the vehicle.",
    freq_dist
)
# Returns: List of sentence dictionaries with vectors
```

3.2 CalCosDist()

Purpose: Calculates cosine similarity between student and standard answer vectors.

Mathematical Foundation:

The cosine similarity is computed as:

$$\cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| \times |\vec{b}|}$$

where θ is the angle between vectors $\vec{a}$ and $\vec{b}$.

Components: q $= \sum_i std\_freq_i^2$
$s = \sum_i stu\_freq_i^2$
$qs = \sum_i std\_freq_i \times stu\_freq_i$
$\cos(\theta) = \frac{qs}{\sqrt{q \times s}}$

3.3 Mark()

Purpose: Evaluates student answers against teacher key sentences.

Algorithm:

1. For each key sentence from the teacher
2. Calculate maximum cosine distance to any student sentence
3. Check if distance exceeds threshold
4. Verify keyword matches (if specified)
5. Record matching details for grading

Returns: List of correctly answered point identifiers.

3.4 Analysis()

Purpose: Main pipeline orchestrating the entire grading process.
Workflow:

Student Text Input
↓
SentenceAnalysis()
↓
Mark()
↓
Comparison()
↓
Omitted()
↓
Final Grade + Feedback

4 Grading Algorithm

4.1 Step-by-Step Process

1. **Text Preprocessing**
   - Sentence segmentation
   - Word tokenization
   - POS tagging with NLTK
   - Lemmatization via WordNet
2. **Vectorization**
   - TF-IDF style weighting
   - Term frequency normalization
   - Inverse document frequency computation
3. **Similarity Calculation**
   - Cosine distance computation
   - Optional term expansion
   - Keyword verification
4. **Scoring**
   - Threshold comparison
   - Rule-based point allocation
   - Final grade determination

4.2 Term Expansion Strategy

4.2.1 Synonym Expansion

When a standard word doesn't appear in the student answer, the system searches for synonyms:

Example:
Standard: ”attorney”
Student: ”lawyer”
Action: Match found via WordNet synonyms
Scale Factor: 0.90

4.2.2 Hypernym Expansion

If no synonym matches, the system checks for ancestor terms (hypernyms):

Example:
Standard: ”lawyer”
Student: ”professional”
Action: Match found via WordNet hypernym chain
Scale Factor: 0.45

5 Configuration Options

5.1 Feature Toggles

Setting	**Default**	**Description**
APPLY_SYNONYM_EXPANSION	False	Enable synonym matching
APPLY_ANCESTOR_EXPANSION	False	Enable hypernym matching
MAX_ANCESTOR_LINKS	5	Maximum hypernym distance
USE_PART_OF_SPEECH	False	Include POS in matching
ONLY_MATCH_SENTENCE_ONCE	False	Prevent sentence reuse
USE_TRUE_TF_IDF	False	Use standard TF-IDF formula
APPLY_GRAMMAR_CHECKING	False	Enable grammar critique

Table 1: Configuration Settings

5.2 Scale Factors

- `synonym_scale_factor`: 0.90 (high confidence)
- `ancestor_scale_factor`: 0.45 (moderate confidence)

6 Example Usage

6.1 Complete Example

```
# Initialize the answer grader
ans = Answer()

# Teacher's key answer
key_text = "The attorney chased the ambulance downtown."

# Student's answer
student_text = "The lawyer chased the vehicle downtown."

```

```python
# Analyze and grade
mark, marklist, omitted = ans.Analysis(
    student_text,
    freq_dist,
    key_sentences,
    point_list,
    grading_rules
)

print(f"Grade: {mark}")
print(f"Correct Points: {marklist}")
print(f"Feedback: {omitted}")
```

6.2 Expected Flow

1. System recognizes "lawyer" ≈ "attorney" (synonym)
2. System recognizes "vehicle" ≈ "ambulance" (synonym)
3. Calculates high cosine similarity
4. Awards points if similarity exceeds threshold

7 Advanced Features

7.1 Grammar Checking Integration

When enabled, the system provides qualitative critique:

- Grammatical error detection
- Sentence structure analysis
- Writing quality metrics

7.2 Closeness Evaluation

The `EvaluateCloseness()` method provides document-level similarity:

- Merges all sentences into single document vectors
- Computes recall and precision over matching tokens
- Calculates F-score weighted by cosine similarity

$$F - score = 2 \times \frac{precision \times recall}{precision + recall}$$

$$Closeness = \cos(\theta) \times F - score$$

8 Utility Functions

8.1 find_most_freq_term()

Finds the most frequent term from a list within a frequency distribution.

8.2 list_difference()

Returns items present in the first list but not in the second.

9 Best Practices

1. **Calibrate Thresholds**: Adjust based on question difficulty
2. **Use Synonym Expansion**: Improves vocabulary flexibility
3. **Enable Grammar Checking**: Provides comprehensive feedback
4. **Test with Sample Data**: Validate grading accuracy
5. **Review Detailed Marks**: Use for system refinement

10 Limitations and Considerations

- Requires well-formed reference answers
- May struggle with highly creative or unconventional responses
- Depends on WordNet coverage for semantic expansion
- Threshold tuning needed for different question types
- Cannot assess reasoning depth or originality

11 Conclusion

This automated essay grading system provides a sophisticated NLP-based approach to evaluating short-answer questions in educational settings. By combining TF-IDF vectorization, cosine similarity, and semantic expansion through WordNet, it offers flexible and reasonably accurate automated grading capabilities.

11.1 Key Strengths

- Semantic understanding via WordNet
- Configurable grading parameters
- Multiple matching strategies
- Detailed feedback generation
- Grammar and style checking integration

11.2 Future Enhancements

Potential improvements could include:

- Deep learning-based semantic similarity
- Context-aware word embeddings (Word2Vec, BERT)
- Multi-language support
- Improved handling of complex reasoning
- Real-time adaptive threshold adjustment