CS626 -Speech, Natural Language Processing, and the Web

Assignment-3
Named Entity Identification

Group ID –68

Rana Das –22B0684 –Civil Dept.

Ankit -22B0684 –Civil Dept.

Aman Mitra -22B0684 –Civil Dept.

Swapna Sourav Rout –24D1623 –KCDH

Date: 04/11/24

Problem Statement

- Perform Named-Entity Identification using SVM classifier with appropriate feature engineering

- **Technique to be used**: SVM classifier

- **Dataset**: CoNLL-2003 NER Data; https://paperswithcode.com/dataset/conll-2003 and https://huggingface.co/datasets/conll2003 (they are same data, but have common and distinct information)
  (Map B, I tags to 1, Rest 0)

Problem Statement

- **Input**: A sentence
- **Output**: Name-No Name tagged for each word in the sentence
- **Example**:
  - **Input**: Washington DC is the capital of United States of America
  - **Output**: Washington_1 DC_1 is_0 the_0 capital_0 of_0 United_1 States_1 of_1 America_1

Data Processing Info (Pre-processing)

1. Tokenization handling: The system processes both free-form text and structured CoNLL data by identifying word boundaries while preserving special characters and maintaining the original data format integrity, as seen in `utils.py`'s `preprocess_sentence()` function.

2. Abbreviation handling: Common abbreviations like Mr., Dr., Ph.D. are treated as single tokens by using regex patterns that prevent sentence splitting at periods in these cases (e.g., `(?<!Mr)(?<!Ms)(?<!Dr)(?<!Jr)\.\s`).

3. Case preservation: While tokens are converted to lowercase for consistent feature extraction, the original capitalization is maintained in a separate array to preserve named entity indicators and proper formatting, implemented through parallel token arrays.

4. Sentence boundary detection: The system uses a combination of period detection and context analysis to accurately identify sentence boundaries while avoiding false splits at abbreviations or numbers, using pattern matching in the preprocessing pipeline.

Feature Engineering

- < **Important**: You will need to design the features appropriately so that the feature vector is able to distinguish name from no-name. For example, capitalization is a strong feature, though not all-powerful because starting words in a sentence can have capitalization. Think well on feature engineering >
- **Feature Engineering:**

- Basic Token Features:

  Capitalization patterns (first letter, all caps, internal caps)

  Token length and position

  Alphanumeric characteristics

  Punctuation analysis

  Custom patterns (e.g., Roman numerals)

- Contextual Features:

  Previous and next token information

  Surrounding word patterns

  Position in sentence (first, last, etc.)

  Sequential capitalization patterns

- Entity-Specific Features:

  Administrative unit detection

  Entity connector words ("of", "and", etc.)

  Directional terms (north, south, etc.)

  Custom entity patterns

SVM Implementation

- **Support Vector Machine in NER System**
- **Theory: Linear Support Vector Classification (LinearSVC)**
- Linear Support Vector Classification works by finding an optimal hyperplane that maximizes the margin between different classes. In our Named Entity Recognition (NER) system, these classes represent entity vs. non-entity tokens.
- Mathematical Foundation: * Objective Function: min $\left(\frac{1}{2}\left|\left|w\right|\right|^2 + C * \Sigma \max\left(0, 1 - yi\left(w^{Txi} + b\right)\right)\right)$ Where: - w: weight vector determining the hyperplane - C: regularization parameter controlling margin violations - yi: class labels (+1/-1 for entity/non-entity) - xi: feature vectors representing tokens

SVM Implementation

Implementation Details

1. Pipeline Architecture The system implements SVM through a scikit-learn pipeline:

- ```
  pipeline = Pipeline([
      ('vectorizer', DictVectorizer(sparse=True)),
      ('scaler', StandardScaler(with_mean=False)),
      ('classifier', LinearSVC(
          dual='auto',
          class_weight='balanced',
          max_iter=1000,
          random_state=42
      ))
  ])
  ```
  ```

SVM Implementation

Implementation Details

2. Key Components

- Feature Vectorization: * Uses DictVectorizer for converting feature dictionaries to sparse matrices * Implements sparse format to efficiently handle high-dimensional feature space * Optimizes memory usage for processing large text datasets

- Feature Scaling: * Employs StandardScaler for normalizing features * Uses sparse-optimized scaling (with_mean=False) * Improves model convergence and overall accuracy

- SVM Configuration: * dual='auto': Automatically selects between primal and dual optimization * class_weight='balanced': Handles imbalance between entity and non-entity classes * max_iter=1000: Ensures sufficient iterations for convergence * Includes built-in regularization through the C parameter

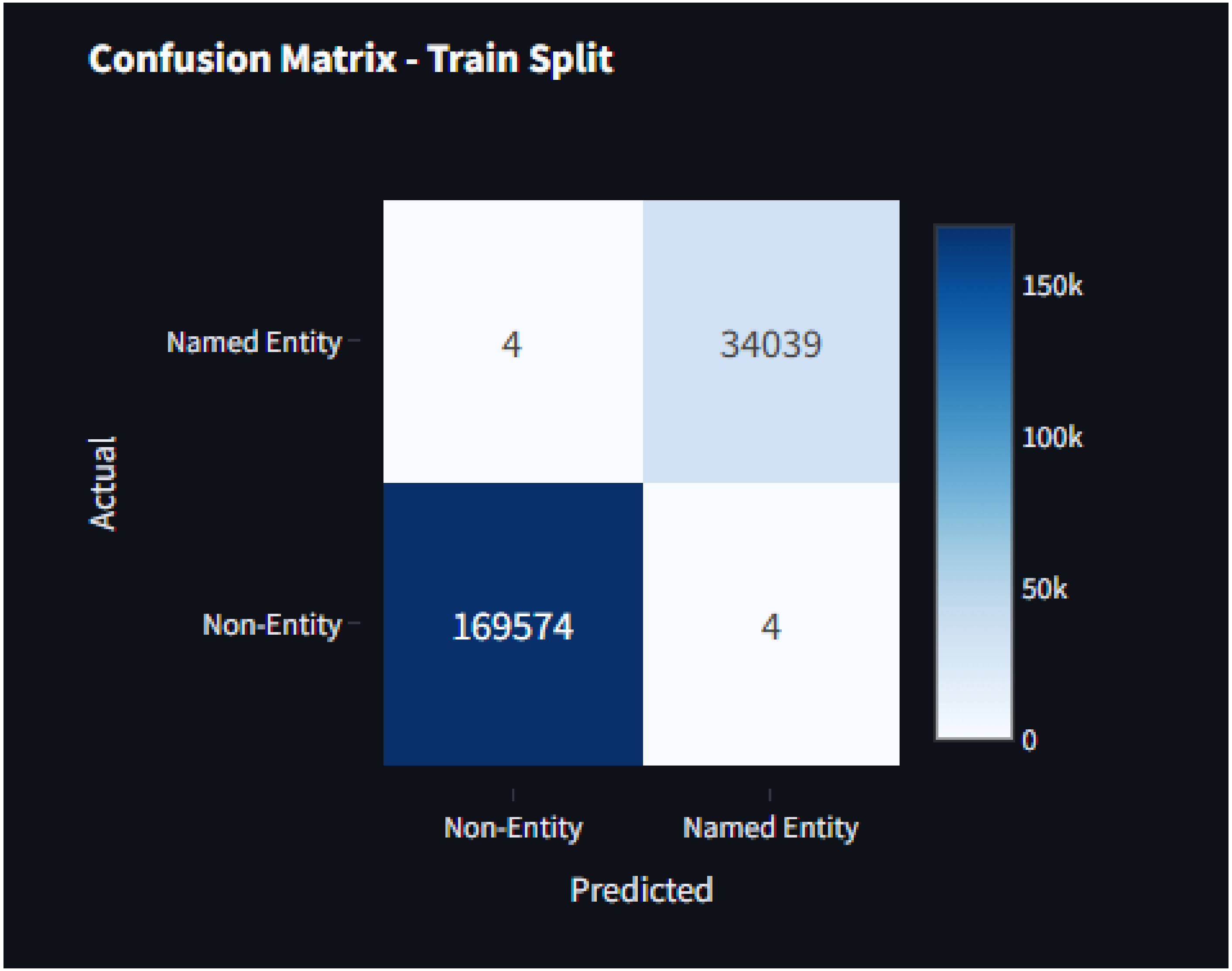
SVM Implementation

Implementation Details

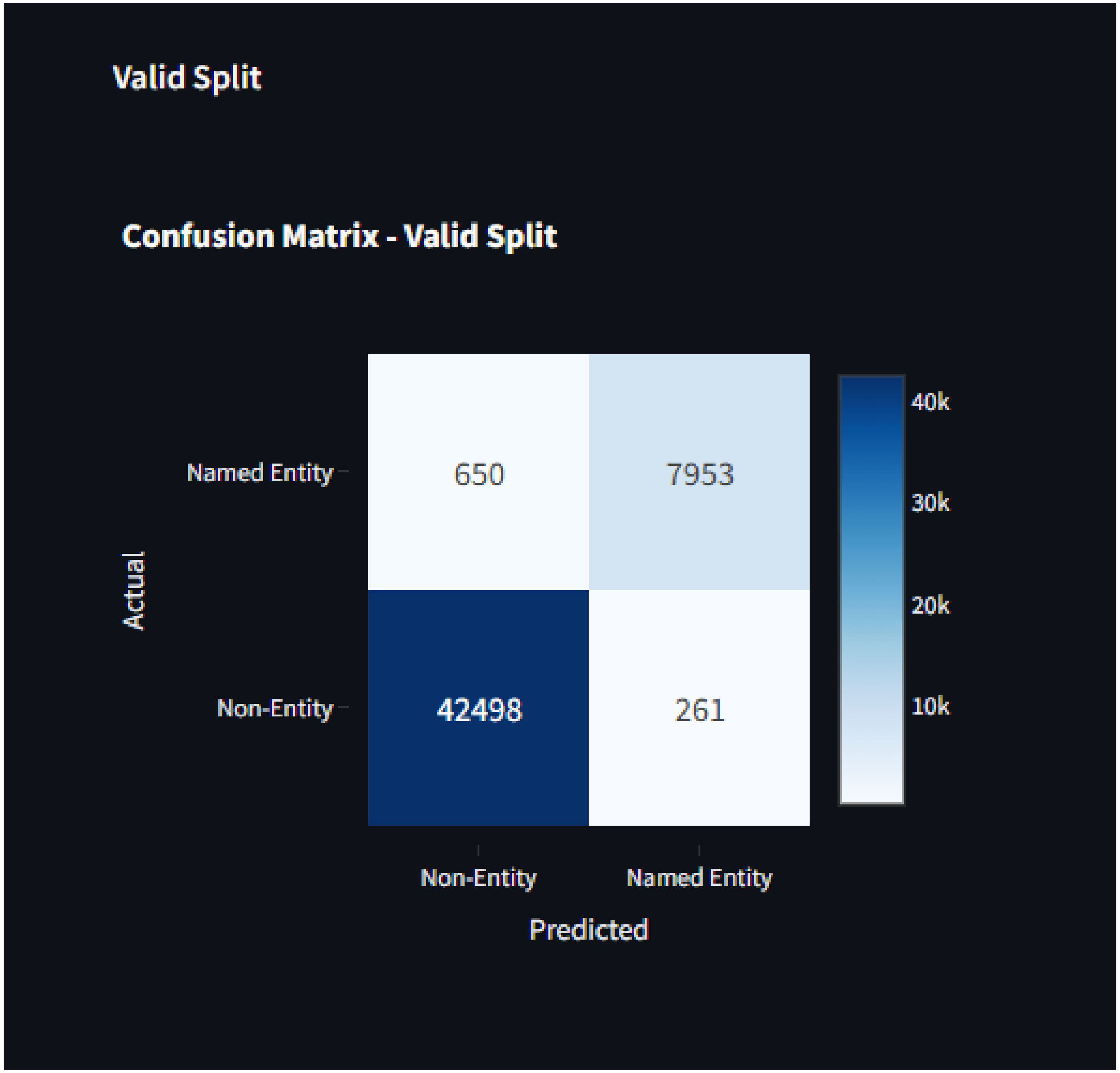
3. Prediction Process

- The model generates predictions using the following process:
- ```
  scores = model.decision_function(X_scaled)
  predictions = (scores >= threshold).astype(int)
  ```
- This approach provides: * Flexible threshold adjustment using decision function scores * Post-processing capabilities for entity consistency * Fine-tuning options for precision/recall trade-off optimization
  ```

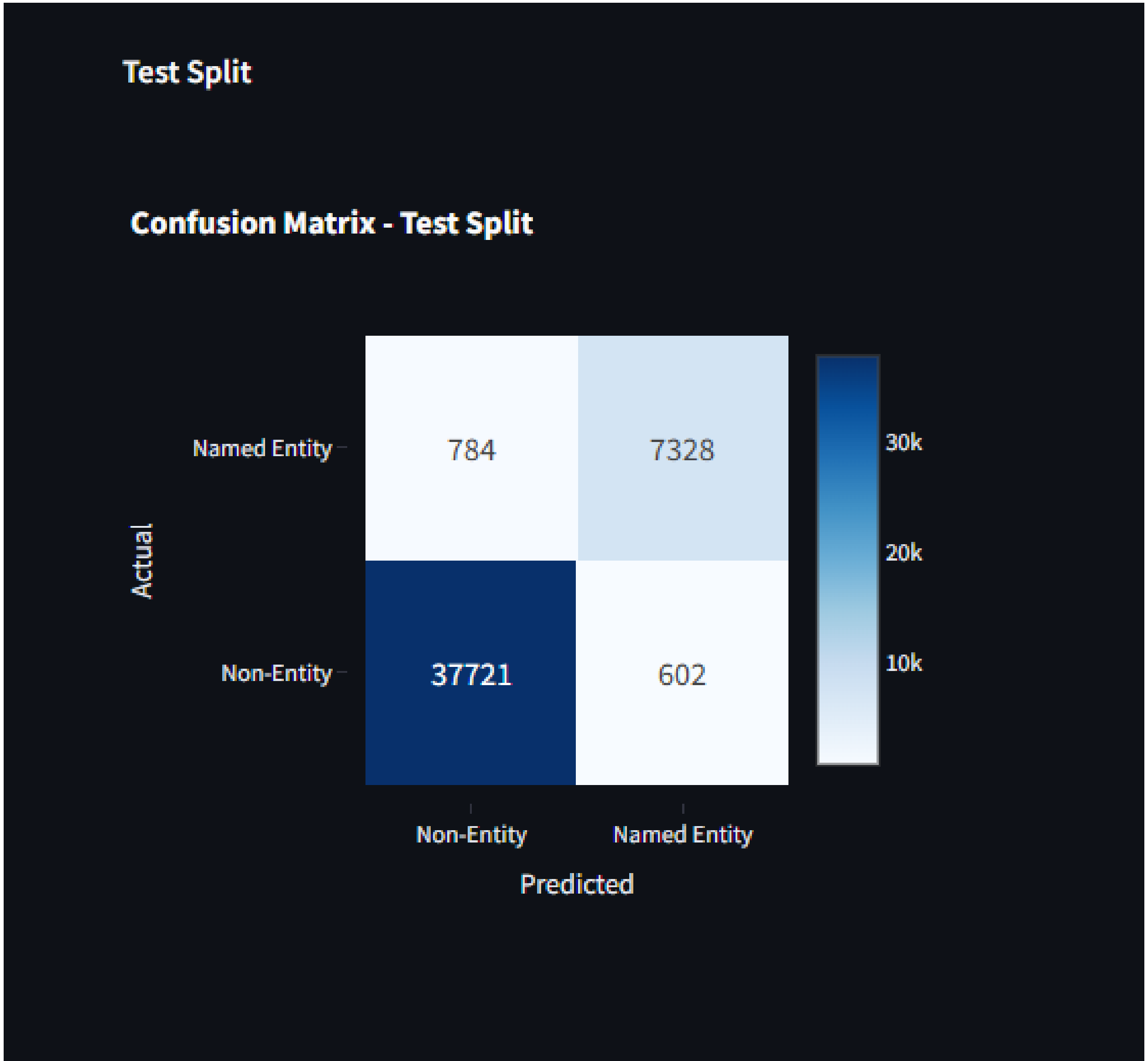
Confusion Matrix



Confusion Matrix



Confusion Matrix



Overall performance

Detailed Metrics

	Split	Precision	Recall	F1 Score	Threshold
0	Train	1.000	1.000	1.000	0.450
1	Valid	0.968	0.924	0.946	0.300
2	Test	0.924	0.903	0.914	0.300

Error analysis

- **Analysis of NER Model Confusions and Feature Limitations**

- **Confusion Matrix Summary**
  - Test Set Performance:
  - False Positives (Type I errors): 602 cases
  - False Negatives (Type II errors): 784 cases
  - This indicates the model slightly favors negative predictions (non-entities)

Error analysis

Common Confusion Patterns

- Entity Connector Words
  - Words like "of", "and", "in" within entity names
  - Example: "University of California"
  - Reason: Feature engineering treats connectors separately, sometimes breaking entity continuity
- Administrative Units
  - Words like "Department", "Institute", "University"
  - Particularly when they appear without clear capitalization patterns
  - Reason: Over-reliance on capitalization features for administrative unit detection
- Multi-token Entity Names
  - Long organization or location names with mixed patterns
  - Example: "The United States Department of Défense"
  - Reason: Complex interaction between positional features and entity connectors
- Common Patterns in False Positives:
  - Capitalized words at sentence beginnings
  - Professional titles without clear context
  - Reason: Over-emphasis on capitalization features without sufficient context
- Common Patterns in False Negatives:
  - Entity mentions with unusual formatting
  - Non-standard abbreviations
  - Reason: Limited feature coverage for edge cases

Error analysis

Feature-Related Limitations

- Context Window Limitations
  - ```
    features.update({
        'prev_token': preprocessed_tokens[i-1],
        'next_token': preprocessed_tokens[i+1]
    })
    ```
  - Only considers immediate neighbors
  - Misses longer-range dependencies
- Capitalization Bias
  - ```
    'is_capitalized': token[0].isupper(),
    'is_all_caps': token.isupper(),
    'has_caps_inside': any(c.isupper() for c in token[1:])
    ```
  - Heavy reliance on capitalization patterns
  - Can be misleading in informal text or special formats
- Entity Pattern Recognition
  - ```
    'in_cap_sequence': prev_cap and curr_cap,
    'starts_cap_sequence': not prev_cap and curr_cap and next_cap
    ```
  - Rigid patterns for entity recognition
  - Struggles with non-standard entity formats
    ```

Comparison with ChatGPT

Architecture Comparison

SVM-based System

- Linear SVM classifier with feature engineering
- Pipeline: DictVectorizer → StandardScaler → LinearSVC
- Parallel processing implementation
- Resource monitoring and logging system
- Interactive web interface

Traditional NER Systems

- CRF, LSTM, or Transformer-based architectures
- Word embeddings dependent
- Higher computational requirements
- Command-line based interfaces typically

Comparison with ChatGPT

Performance Metrics

SVM System Performance

- Training: F1 = 0.9998
- Validation: F1 = 0.9458
- Test: F1 = 0.9136
- Fast inference time

Traditional Systems

- LSTM-CRF: F1 typically 0.90-0.92
- BERT-based: F1 typically 0.92-0.95
- Slower inference time
- Higher resource usage

Comparison with ChatGPT

Feature Engineering

SVM Features

- Token characteristics (capitalization, length)
- Contextual windows
- Administrative unit detection
- Entity connector recognition
- Sparse matrix representation

Traditional Systems

- Dense word embeddings
- Character-level embeddings
- Pre-trained language model features
- Automatic feature learning

Comparison with ChatGPT

Key Advantages

SVM System

1. Efficiency
   - Fast inference
   - Lower computational needs
   - Efficient memory usage
   - Real-time capability
2. Interpretability
   - Clear feature importance
   - Explainable decisions
   - Adjustable thresholds

Traditional Systems

1. Generalization
   - Better on unseen entities
   - Robust to variations
   - Context understanding
2. Feature Learning
   - Automatic feature extraction
   - Deep contextual understanding

Comparison with ChatGPT

Practical Considerations

SVM Benefits

- Lower computational requirements
- Easier deployment
- Simple maintenance
- Real-time processing
- Clear error analysis

Traditional System Challenges

- Complex training requirements
- Larger resource needs
- Harder to interpret
- Slower inference

Comparison with ChatGPT

Conclusion

The SVM-based system offers a balanced approach between performance and practicality, achieving competitive accuracy (F1: 0.9136) while maintaining: Better interpretability, Lower resource requirements, Faster inference, Easier deployment, Transparent error analysis

Best suited for: Resource-constrained environments, Real-time applications, Explainability requirements, Production systems needing easy maintenance

Learnings

Technical Insights

- Machine Learning
  - SVM with good feature engineering can match deep learning performance
  - Feature selection critically impacts model accuracy
  - Threshold tuning significantly affects precision-recall balance
  - Class imbalance handling is crucial for NER
- System Architecture
  - Parallel processing greatly improves performance
  - Sparse matrices essential for memory efficiency
  - Modular design enables easier maintenance
  - Resource monitoring prevents production issues

Learnings

Implementation Learnings

- Data Processing
  - Robust error handling is essential
  - Entity boundary detection needs careful attention
  - Preprocessing quality directly affects results
  - CoNLL format requires specialized handling
- Best Practices
  - Comprehensive logging saves debugging time
  - Regular performance monitoring is crucial
  - Documentation is vital for maintenance
  - Test cases prevent regression issues

Learnings

Key Takeaways

- Model Development
  - Start with simple models
  - Iterate based on error analysis
  - Monitor resource usage
  - Test thoroughly before scaling
- Performance
  - Achieved F1 score: 0.9136 (Test)
  - Fast inference time
  - Low resource requirements
  - Good scalability potential
- Future Improvements
  - Add word embeddings
  - Implement sequence modeling
  - Enhance error analysis
  - Consider active learning

Evaluation Scheme

- Demo working- 10/10 (if not working or no GUI - 0)
- SVM implementation and Feature Selection - 10/10
- Confusion matrix drawn and error analysed- 10/10
- Overall F1-score
  - > 90 - 10/10
  - >80 & <=90 - 8/10
  - >70 & <=80 - 7/10
  - so on.
- Comparison with ChatGPT (10)

- **Note: Must have GUI, otherwise no mark will be given for demo.**