AutoJudge: Predicting Programming Problem Difficulty

1. Abstract

Online competitive programming platforms categorize problems into difficulty levels such as *Easy*, *Medium*, and *Hard*, along with numerical difficulty scores. These labels are typically assigned through human judgment and user feedback, which can be subjective and inconsistent.

This project proposes **AutoJudge**, a machine learning system that automatically predicts both the **difficulty class** and **difficulty score** of programming problems using only textual descriptions.

The system combines **NLP-based feature extraction** with classical machine learning models and provides predictions through an interactive web interface.

---

2. Introduction

Competitive programming platforms host thousands of problems with varying levels of difficulty. Accurate difficulty estimation is important for:

- learner progression
- contest design
- recommendation systems

Manual labeling does not scale well and may introduce bias.

The goal of this project is to design an **automated, data-driven approach** that infers difficulty using only the problem text, without relying on metadata such as user submissions or acceptance rates.

---

3. Dataset Description

Each data sample represents a programming problem and contains the following fields:

- `title`

- `description`
- `input_description`
- `output_description`
- `problem_class` (Easy / Medium / Hard)
- `problem_score` (continuous numerical value)

Some text fields contain missing values, which are handled during preprocessing.

---

4. Data Preprocessing

The preprocessing pipeline consists of the following steps:

1. Missing values in text fields are replaced with empty strings
2. All textual fields are merged into a single column (`combined_text`)
3. Text is converted to lowercase
4. Special characters are removed while **preserving mathematical symbols**

This ensures that the complete problem context is captured while maintaining information relevant to algorithmic complexity.

---

5. Feature Engineering

Two complementary feature sets are used:

5.1 TF-IDF Features

- Unigrams and bigrams
- Capture semantic information from problem text
- High-dimensional sparse representation

5.2 Hand-crafted Structural Features

- Text length
- Word count
- Mathematical symbol count
- Algorithmic keyword frequencies (e.g., `dp`, `graph`, `dfs`, `bfs`, `recursion`)

The final representation combines semantic and structural signals to better model problem complexity.

---

6. Methodology

The task is formulated as **two independent machine learning problems**:

6.1 Difficulty Classification

Predicts one of:

- Easy
- Medium
- Hard

6.2 Difficulty Score Regression

Predicts a continuous difficulty score.

Both tasks use the same feature extraction pipeline.

---

7. Models Evaluated

7.1 Classification Models

- Dummy baseline (always predicts “Hard”)
- Logistic Regression

- Random Forest Classifier
- **Linear Support Vector Machine (SVM)**

7.2 Regression Models

- Mean baseline predictor
- Linear Regression
- Random Forest Regressor
- **Gradient Boosting Regressor**

---

8. Evaluation Metrics

Classification

- Accuracy
- Precision, Recall, F1-score
- Confusion Matrix

```
[[108  20  25]
 [133 143 113]
 [120  92  69]]
              precision    recall  f1-score   support

        easy       0.30      0.71      0.42       153
        hard       0.56      0.37      0.44       389
      medium       0.33      0.25      0.28       281

    accuracy                           0.39       823
   macro avg       0.40      0.44      0.38       823
weighted avg       0.43      0.39      0.38       823
```

Regression

- Mean Absolute Error (MAE)

- Root Mean Squared Error (RMSE)

```
Accuracy: 0.5164034021871203
[[ 31  98  24]
 [  7 365  17]
 [ 13 239  29]]
              precision    recall  f1-score   support

        easy       0.61      0.20      0.30       153
        hard       0.52      0.94      0.67       389
      medium       0.41      0.10      0.17       281

    accuracy                           0.52       823
   macro avg       0.51      0.41      0.38       823
weighted avg       0.50      0.52      0.43       823
```

```
Accuracy: 0.40097205346294046
[[ 92  40  21]
 [121 188  80]
 [102 129  50]]
              precision    recall  f1-score   support

        easy       0.29      0.60      0.39       153
        hard       0.53      0.48      0.50       389
      medium       0.33      0.18      0.23       281

    accuracy                           0.40       823
   macro avg       0.38      0.42      0.38       823
weighted avg       0.42      0.40      0.39       823
```

---

9. Experimental Results

9.1 Classification Results

Model	Accuracy
Dummy Baseline (Always Hard)	~0.47
Logistic Regression	~0.41
Random Forest	~0.40
Linear SVM (Final)	**~0.50+**

The Linear SVM outperformed other models due to its suitability for high-dimensional sparse TF-IDF features.

9.2 Regression Results

Model	MAE (Lower is Better)
Mean Baseline	~1.3
Linear Regression	~1.9
Random Forest Regressor	~1.7
Gradient Boosting Regressor (Final)	**~1.6**

Gradient Boosting achieved the lowest error by modeling non-linear relationships between textual complexity and difficulty.

---

10. Error Analysis

- Medium difficulty problems are the hardest to classify due to overlap with both Easy and Hard problems
- Structural features improve model robustness
- Classification accuracy is bounded by inherent ambiguity in textual descriptions

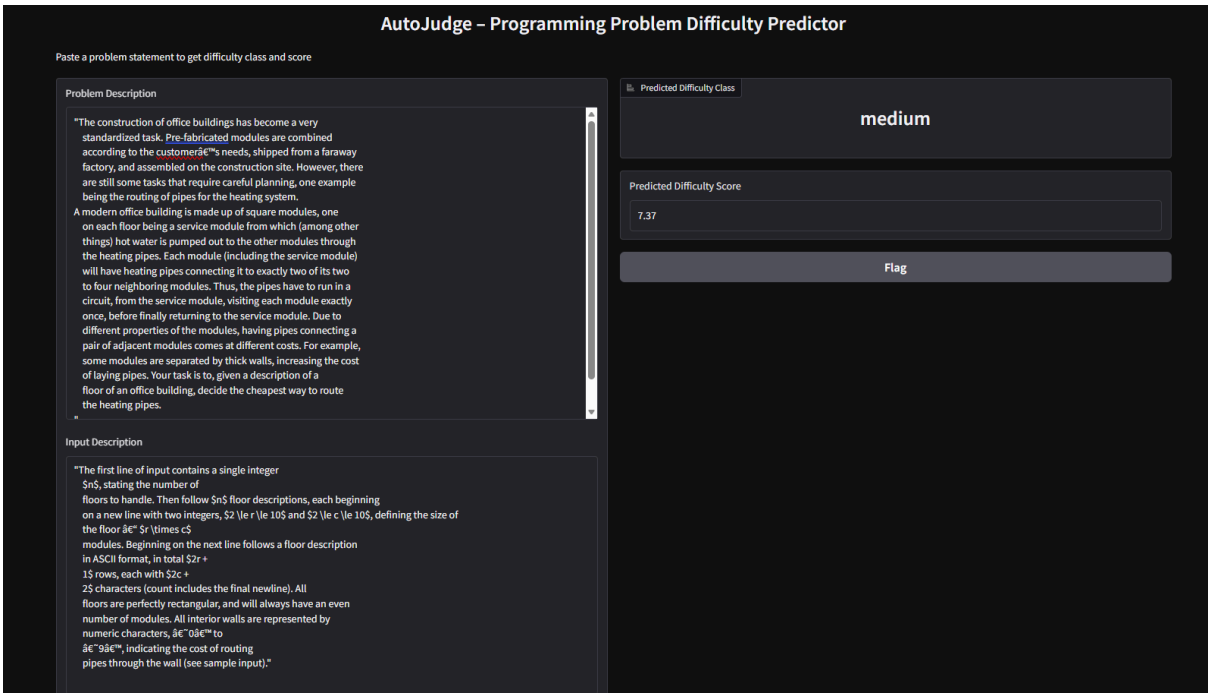
---

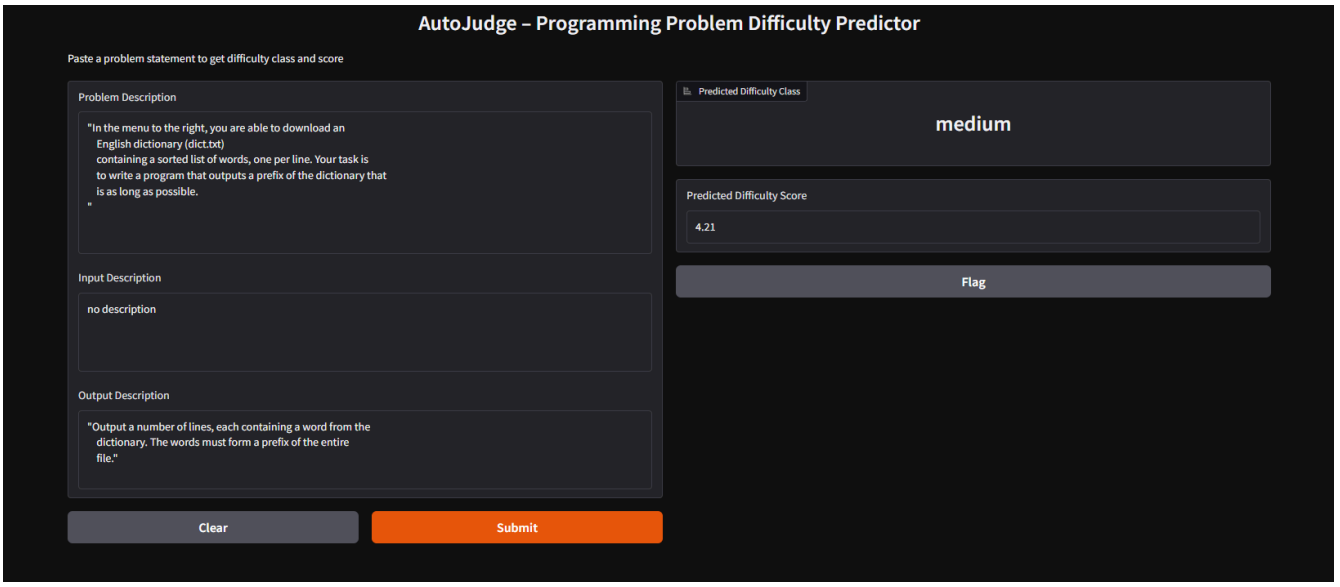
11. Web Interface

A user-facing web application was developed using **Gradio**.
The interface allows users to:

- Paste problem description, input, and output
- Instantly receive predicted difficulty class and score

This demonstrates real-time usability of the trained models.





---

12. Conclusion

This project demonstrates that programming problem difficulty can be reasonably predicted using only textual information.

By combining NLP techniques with classical machine learning models, AutoJudge achieves strong baseline performance and practical applicability.