

AutoJudge: Predicting Programming Problem Difficulty

A Machine Learning–Based Approach Using Textual Features

Submitted by:

Dakshata Aggrawal

2nd Year, B.Tech – Computer Science and Engineering

Indian Institute of Technology, Roorkee

1. Introduction

AutoJudge is a machine learning–based system designed to predict the difficulty of programming problems using only their textual descriptions. The system outputs both a **categorical difficulty label** (Easy, Medium, Hard) and a **numerical difficulty score** on a scale of 1–10.

The motivation behind this project is to explore whether natural language information alone is sufficient to estimate problem difficulty, without relying on external metadata such as submission statistics, editorial complexity, time limits or memory constraints.

In addition to model training and evaluation, a lightweight **Flask-based web application** was developed to demonstrate the system interactively.

2. Dataset

The dataset used in this project was provided as part of the problem statement.

Dataset Link: [GitHub - AREEG94FAHAD/TaskComplexityEval-24](https://github.com/AREEG94FAHAD/TaskComplexityEval-24)

Each entry in the dataset contains:

- Problem description
- Input description
- Output description
- Difficulty class label (Easy / Medium / Hard)
- Numerical difficulty score

No external datasets or additional annotations were used.

3. Preprocessing and Feature Extraction

3.1 Preprocessing

- Missing text fields were handled by replacing them with empty strings.
- Although the web interface accepts the problem description, input description, and output description separately, these fields are **combined internally** during preprocessing for feature extraction.

3.2 Feature Extraction

The primary feature extraction method used is **TF-IDF (Term Frequency–Inverse Document Frequency)** vectorization with a maximum of 5000 features and English stop-word removal.

In addition to TF-IDF, a small set of handcrafted textual features was explored, including:

- Text length
 - Mathematical symbol count
 - Keyword frequencies (e.g., `dp`, `graph`, `recursion`)
-

4. Setup

- The dataset was split into **80% training data and 20% test data**.
 - A fixed random seed was used to ensure reproducibility.
 - Classification and regression tasks were handled **independently** using separate models.
-

5. Classification Models and Results

5.1 Models Evaluated

The following models were evaluated for predicting the difficulty class:

- Logistic Regression
- Support Vector Machine (LinearSVC)
- Random Forest Classifier

Logistic Regression Accuracy: 0.5358

SVM Accuracy: 0.4897

Random Forest Accuracy: 0.5431

5.2 Final Classification Model

The **Random Forest Classifier** achieved the highest accuracy (54.31%) and was selected as the final classification model.

Final Classification Accuracy: 0.543134872417983

5.3 Confusion Matrix

The confusion matrix for the final classifier is shown below:

```
[[ 32  66  38]
 [ 11 374  40]
 [ 19 202  41]]
```

From this matrix, it can be observed that:

- Medium difficulty problems are classified most accurately.
- There is noticeable overlap between Medium and Hard classes.
- This overlap reflects the inherent ambiguity of predicting problem difficulty using text-only features.

6. Regression Models and Results

6.1 Models Evaluated

The following models were evaluated for predicting the numerical difficulty score:

- Linear Regression
- Random Forest Regressor
- Gradient Boosting Regressor

```
--- Regression Models ---
Linear Regression | MAE: 2.256 | RMSE: 2.815
Random Forest | MAE: 1.694 | RMSE: 2.045
Gradient Boosting | MAE: 1.712 | RMSE: 2.051
```

6.2 Final Regression Model

The **Random Forest Regressor** achieved the lowest error and was selected as the final regression model.

- **Mean Absolute Error (MAE):** 1.69
- **Root Mean Squared Error (RMSE):** 2.05

```
Final Regression MAE: 1.6944708383961118
Final Regression RMSE: 2.045078682901665
```

7. System Design

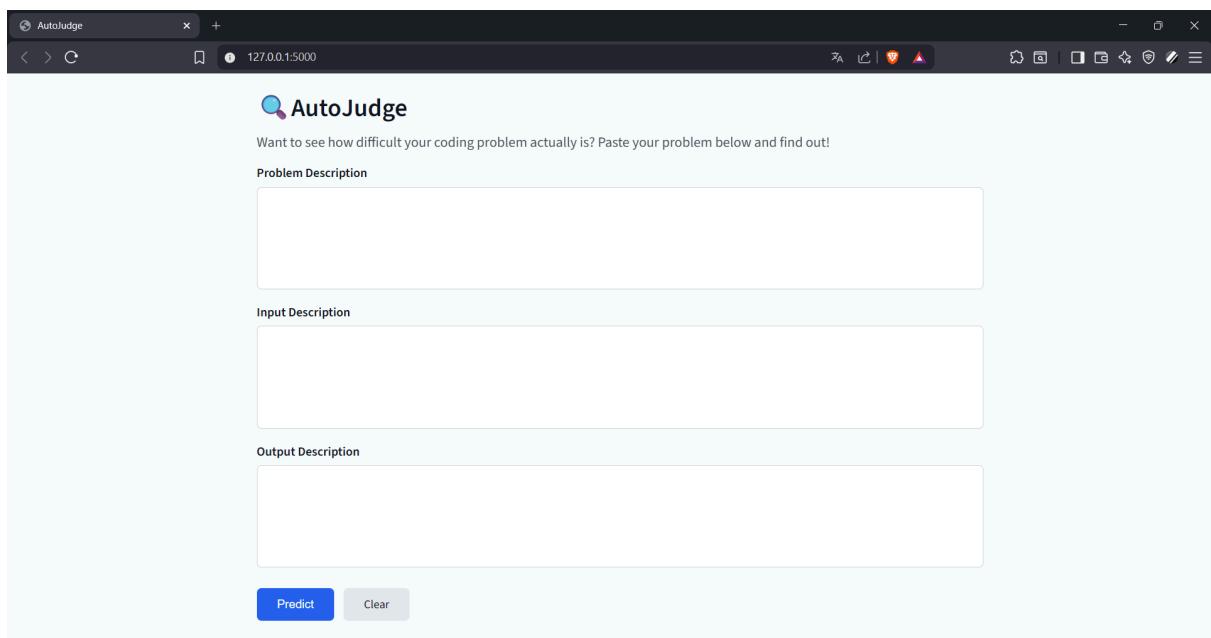
The AutoJudge system consists of two main components:

1. Model Training Pipeline

- Text preprocessing
- TF-IDF feature extraction
- Independent training of classification and regression models

2. Web Application

- Built using Flask
- Accepts problem description, input description, and output description
- Displays predicted difficulty class and score

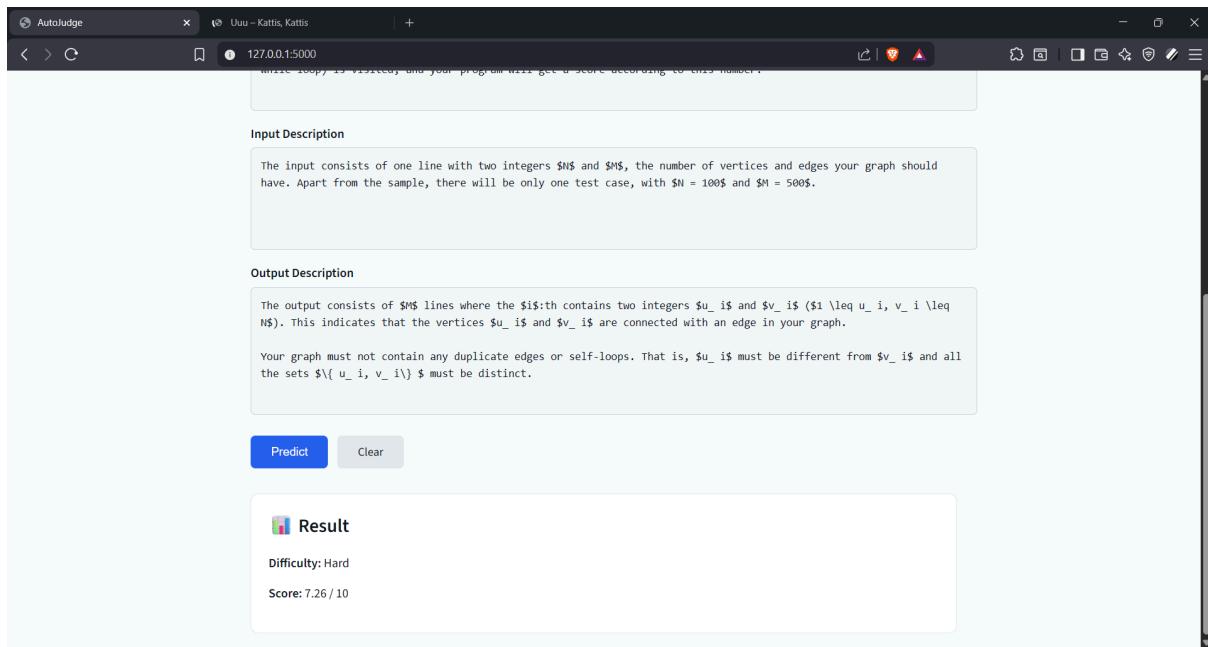


The classification and regression predictions are generated independently, and no explicit numeric thresholds are used to map scores to difficulty classes.

8. Demo and Validation

A demo video demonstrating the working of the system has been recorded and included in the repository. The video showcases:

- Inputting problem statements
- Model predictions for difficulty class and score
- End-to-end functionality of the web interface



9. Limitations

- Difficulty labels are subjective by nature.
 - Text-only features cannot fully capture constraints or hidden complexity.
 - Overlap between difficulty classes is unavoidable, given the dataset.
-

10. Future Improvements

- Incorporate richer features such as constraints and algorithm-specific patterns to better capture problem complexity.
 - Use joint modeling to predict difficulty class and score together for improved consistency.
 - Apply cross-validation and class-wise metrics for more reliable evaluation.
 - Address class imbalance using weighting or resampling techniques.
 - Improve the web interface with confidence scores and basic explainability.
 - Expand the dataset using problems from multiple competitive programming platforms.
-

11. Conclusion

AutoJudge demonstrates that machine learning models can reasonably estimate the difficulty of programming problems using only textual information. Despite inherent ambiguity, the system provides a meaningful baseline and a complete end-to-end pipeline, from data preprocessing and model training to deployment via a web interface.