AutoJudge: Automated Programming Problem Difficulty Prediction System

Romit Singh
github.com/KNIGHT-29

1 Introduction

Competitive programming platforms and online coding education portals host thousands of programming problems with varying difficulty levels. Assigning appropriate difficulty labels is essential for learner progression, contest fairness, and structured problem recommendation. However, manual difficulty assignment is subjective and inconsistent.

This project presents **AutoJudge**, an automated machine learning–based system that predicts the difficulty of programming problems using both classification and regression techniques. The system predicts:

- A categorical difficulty label (Easy, Medium, Hard)
- A continuous numerical difficulty score

Predictions are generated by analyzing textual and numerical features extracted from problem statements and are presented through a web-based interface.

2 Dataset Description

2.1 Dataset Source

The dataset consists of programming problems collected from competitive programming platforms and curated repositories. Each entry corresponds to a single problem statement.

2.2 Dataset Attributes

Each problem contains:

- Problem statement text
- Constraints and limits
- Difficulty label
- Derived numerical features

2.3 Target Variables

- **Classification Target:** Easy / Medium / Hard
- **Regression Target:** Continuous difficulty score

3 Data Preprocessing

3.1 Text Cleaning

To improve model performance, the following preprocessing steps were applied:

- Conversion to lowercase
- Removal of punctuation and special characters
- Stopword removal
- Whitespace normalization

3.2 Tokenization

Problem statements were tokenized into individual words, enabling numerical representation using vectorization techniques.

3.3 Handling Missing Values

Problems with missing textual content were removed. Missing numerical values were handled using mean imputation.

4 Feature Engineering

4.1 TF-IDF Vectorization

Term Frequency–Inverse Document Frequency (TF-IDF) was used to transform textual problem statements into numerical feature vectors. This approach captures the relative importance of words across the dataset.

4.2 Numerical Features

Additional numerical features include:

- Length of the problem statement
- Number of constraints

- Frequency of algorithmic keywords (e.g., DP, graph, modulo)

All features were combined into a single feature vector for model training.

5 Models Used

5.1 Difficulty Classification Model

Model: Random Forest Classifier

The Random Forest classifier was selected for its robustness to noise, ability to handle high-dimensional feature spaces, and strong generalization performance.

5.1.1 Evaluation Metric

- Classification Accuracy

5.2 Difficulty Regression Model

Model: Random Forest Regressor

The regression model predicts a continuous difficulty score representing relative problem complexity.

5.2.1 Evaluation Metrics

- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)

6 Experimental Results

6.1 Classification Results

Metric	Value
Classification Accuracy	43.26%

Table 1: Difficulty Classification Performance

The classification model achieves a moderate accuracy of 43.26%. This result indicates that while the model captures general difficulty patterns, overlap between Medium and Hard classes exists due to similar textual and structural characteristics.

6.2 Regression Results

Metric	Value
Mean Absolute Error (MAE)	1.69
Root Mean Squared Error (RMSE)	2.02

Table 2: Regression Performance Metrics

The low MAE and RMSE values indicate that the regression model predicts difficulty scores with minimal deviation from the true values.

7 Web Interface

7.1 System Architecture

The web interface was implemented using Flask/Streamlit and integrates the trained classification and regression models for real-time prediction.

7.2 User Workflow

1. User inputs a programming problem statement
2. Input is preprocessed and vectorized
3. Models generate difficulty predictions
4. Results are displayed instantly

7.3 Output

The interface displays:

- Predicted difficulty category
- Predicted difficulty score

Screenshots of the web interface are included in the submitted report.

8 Saved Models

The trained models were serialized and saved for reproducibility:

- `difficulty_classifier.pkl`
- `difficulty_regressor.pkl`

9 Conclusion and Future Work

This project demonstrates an automated and scalable approach to programming problem difficulty prediction using machine learning. By combining text analysis with classification and regression models, AutoJudge reduces subjectivity and improves consistency in difficulty labeling.

9.1 Future Enhancements

- Integration with online judges
- Transformer-based NLP models (e.g., BERT)
- Multilingual problem support
- Docker-based deployment

Author Details

Name: Romit Singh
GitHub: https://github.com/KNIGHT-29
Project Repository: AutoJudge