AutoJudge: Automated Programming Problem Difficulty Predictor

Comprehensive Project Report

January 2026

Abstract

The rapid growth of competitive programming platforms has created a need for automated tools that can organize and assess problem difficulty at scale. AutoJudge is a machine learning–based system designed to predict the difficulty of programming problems using only their textual descriptions. The system performs both categorical classification (Easy, Medium, Hard) and numerical regression (difficulty score). By combining Natural Language Processing techniques with classical machine learning models, AutoJudge provides an end-to-end solution that includes data preprocessing, feature engineering, model training, evaluation, and deployment as a web application.

1. Problem Statement

Determining the difficulty level of a programming problem is a critical yet subjective task in computer science education. Difficulty depends on factors such as algorithmic complexity, input constraints, edge cases, and the logical depth of the solution. Traditionally, difficulty labels are assigned manually by experts, which is time-consuming and inconsistent.

With thousands of new problems being created across online platforms, manual classification is no longer scalable. AutoJudge addresses this challenge by automatically predicting difficulty using machine learning, producing both categorical and numerical difficulty estimates.

2. Dataset Description

The project uses the TaskComplexityEval-24 dataset, which consists of programming problems stored in JSON Lines (JSONL) format. Each record represents a single problem along with its associated metadata and difficulty labels.

2.1 Data Attributes

1. Title: Name of the programming problem.
2. Description: Detailed explanation of the task.

3. Input Description: Format and constraints of input.
4. Output Description: Expected output format.
5. Problem Class: Ground truth difficulty label (Easy, Medium, Hard).
6. Problem Score: Numerical difficulty rating.

2.2 Class Distribution Analysis

The dataset exhibits an imbalanced yet realistic distribution of difficulty classes. Hard problems are more frequent than Easy ones, reflecting real-world competitive programming platforms. This imbalance introduces additional challenges for classification.

3. Methodology

3.1 Data Preprocessing

Raw textual data contains inconsistencies such as missing fields and varying formats. To address this, all text fields were cleaned and combined into a single textual representation. Missing values were replaced with empty strings, and list-based fields were converted to plain text.

3.2 Feature Engineering

To convert text into numerical features, TF-IDF vectorization was applied. This approach assigns higher weights to informative terms such as algorithmic keywords while reducing the impact of commonly occurring words. The resulting feature space captures semantic patterns related to problem complexity.

4. Model Architecture

4.1 Classification Model

A Logistic Regression classifier was selected for difficulty classification. This model is well-suited for high-dimensional sparse data and provides interpretable decision boundaries.

4.2 Regression Model

A Random Forest Regressor was used to predict numerical difficulty scores. The ensemble nature of Random Forests allows the model to capture non-linear relationships between textual features and difficulty scores.

5. Experimental Results

The models were evaluated using a held-out test set. The classification model achieved an accuracy of 0.5139, which is significantly higher than random guessing (33%).

	Pred Easy	Pred Medium	Pred Hard
Actual Easy	29	57	50
Actual Hard	7	318	100
Actual Medium	16	170	76

For regression, the model achieved a Mean Absolute Error (MAE) of 1.71 and a Root Mean Squared Error (RMSE) of 2.06, indicating reasonable predictive accuracy given the subjective nature of difficulty scores.

6. Web Interface and Deployment

A web interface was developed using Streamlit to allow users to interact with the system. Users can input problem statements and instantly receive difficulty predictions. The application has been deployed publicly and is accessible at: https://autojudge-ggzpe7nvufkckhtbmtrmft.streamlit.app/

7. Limitations

The primary limitation of the system is the subjective nature of difficulty labels. Overlapping vocabulary between classes and the absence of constraints or solution logic limit achievable accuracy. Further improvements would require additional metadata or deep learning models.

8. Conclusion

AutoJudge demonstrates the feasibility of automating programming problem difficulty prediction using machine learning and natural language processing. The system provides a complete end-to-end solution and serves as a strong baseline for future enhancements.