AutoJudge: Automated Prediction of Programming Problem Difficulty

Abstract

AutoJudge is a machine learning based system developed to automatically estimate the difficulty of programming problems using only their textual descriptions. The system performs two related tasks: classification of problems into Easy, Medium, or Hard categories, and regression-based prediction of a numerical difficulty score. This report explains the complete project in a clear and structured manner, describing what was implemented, why specific design choices were made, and why the achieved accuracy represents a realistic upper bound for this problem.

1. Introduction

Programming problem difficulty estimation is an essential component of competitive programming platforms and educational systems. Difficulty labels guide learners in selecting appropriate problems and help organizers create balanced contests and assignments. Traditionally, these labels are assigned manually by experts or inferred from user statistics such as acceptance rates and submission counts.

However, manual labeling is subjective and inconsistent, while user statistics are platform-dependent and unavailable for new problems. AutoJudge addresses this limitation by attempting to predict problem difficulty using only the problem's textual description. This makes the system independent of platform metadata and applicable in a wide range of settings.

2. Problem Statement and Objectives

The central question addressed in this project is whether a machine learning model can learn patterns from problem statements that correlate with their difficulty level. Based on this motivation, the following objectives were defined:

- Classify programming problems into Easy, Medium, and Hard categories.
- Predict a numerical difficulty score representing relative complexity.
- Build a complete end-to-end machine learning pipeline.
- Provide predictions through a simple and user-friendly web interface.

3. Dataset Description

The dataset used in this project consists of several thousand programming problems stored in JSON Lines (JSONL) format. Each entry contains a problem title, a detailed description, input specification, output specification, a difficulty class label, and a numerical difficulty score.

The dataset covers a broad range of algorithmic domains including data structures, graph algorithms, dynamic programming, greedy techniques, and mathematical problems. While this diversity makes the dataset realistic, it also introduces significant overlap between difficulty categories.

4. Data Preprocessing

Machine learning models cannot operate directly on raw text. Therefore, preprocessing is a crucial step in the pipeline. All textual components of each problem are combined into a single unified text field to provide complete context to the model.

- Combining title, description, input, output, and sample input-output into one text.
- Converting list-based fields into plain text.
- Handling missing values by replacing them with empty strings.

5. Feature Extraction

After preprocessing, the textual data is transformed into numerical features using Term Frequency–Inverse Document Frequency (TF-IDF) vectorization. TF-IDF assigns higher importance to informative words while reducing the impact of commonly occurring terms.

Algorithmic keywords such as 'dynamic programming', 'graph', or 'recursion' often indicate higher problem complexity. TF-IDF enables the model to focus on such terms without relying on manual rule-based systems.

6. Machine Learning Models

Two different machine learning models are used to address the classification and regression tasks. Logistic Regression is selected for difficulty classification due to its simplicity, interpretability, and effectiveness on high-dimensional text data.

For numerical difficulty prediction, a Random Forest Regressor is used. This model is capable of learning non-linear relationships and performs well in the presence of noisy and complex feature spaces.

7. Evaluation and Results

The dataset is split into training and testing sets using an 80–20 ratio. The classification model achieves approximately 50–52% accuracy. For a three-class problem, random guessing would yield around 33% accuracy, indicating that the model learns meaningful patterns from the text.

Regression performance is evaluated using Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). The obtained values indicate that predicted difficulty scores are reasonably close to the actual scores, considering the subjective nature of the task.

8. Why Accuracy Cannot Be Improved Significantly

Improving accuracy beyond the current level is fundamentally challenging due to inherent limitations of the problem. Difficulty labels are subjective, and boundaries between Easy, Medium, and Hard are often ambiguous. Many problems share similar vocabulary despite having different implementation complexity.

- Difficulty labels are subjective and overlap heavily.
- Text-only analysis cannot capture hidden edge cases or solution strategies.
- Important factors such as constraints and time limits are not fully represented in text.

9. Web Interface

A simple web interface is developed using Streamlit to make the system accessible to users. The interface allows users to paste problem details and receive predictions in real time. The trained models and TF-IDF vectorizer are loaded at runtime, ensuring fast and consistent predictions.

10. Conclusion

AutoJudge demonstrates the practical application of natural language processing and machine learning to automate programming problem difficulty estimation. While accuracy is constrained by the subjective nature of difficulty labeling, the system provides a reliable baseline and a complete end-to-end solution. The project is well-suited for academic evaluation and can be extended in future work using more advanced models or additional contextual data.