AutoJudge: Predicting Programming Problem Difficulty Using Textual Descriptions

Author:
Aditya Yadav
Department of Computer Science
Indian Institute of Technology Roorkee

Introduction

Online competitive programming platforms such as Codeforces and CodeChef categorize programming problems into difficulty levels such as Easy, Medium, and Hard, and often assign numerical difficulty ratings. These difficulty levels help users select appropriate problems and track their progress. However, the process of assigning difficulty is largely manual and subjective, relying on expert judgment and user feedback.

The objective of this project is to build an automated system, named **AutoJudge**, that predicts the difficulty of programming problems using only their textual descriptions. The system performs two tasks: (i) classification of problems into Easy, Medium, or Hard categories, and (ii) regression to predict a numerical difficulty score.

The proposed approach uses natural language processing (NLP) techniques to extract features from problem statements and applies machine learning models for both classification and regression. A web-based interface is also developed to demonstrate real-time predictions.

Problem Statement

The goal of this project is to design an end-to-end machine learning system that, given the textual description of a programming problem, predicts:

1. The difficulty class (Easy / Medium / Hard)
2. A numerical difficulty score

The system must use only the textual content of the problem, including the problem description, input specification, and output specification. No external metadata such as contest division, tags, or editorial information is allowed.

The final system should be executable locally and provide predictions through a simple web interface.

Dataset Description

This project uses the TaskComplexityEval-24 dataset, a publicly available dataset designed for evaluating the complexity of programming tasks.

Each data sample in the dataset consists of:

- Problem title

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

The dataset contains thousands of programming problems collected from competitive programming sources and manually annotated with difficulty labels and scores.

Dataset Source:
https://github.com/AREEG94FAHAD/TaskComplexityEval-24

The dataset is stored in JSONL format and is used directly without any manual relabeling.

Data Preprocessing
Prior to model training, several preprocessing steps were applied to clean and standardize the textual data:

- All text was converted to lowercase.
- Special characters and punctuation were removed.
- Extra whitespace was eliminated.
- Missing values were handled by replacing them with empty strings.
- The title, description, input description, and output description were concatenated into a single combined text field.

These preprocessing steps ensure consistency across samples and reduce noise in the textual data before feature extraction.

Feature Engineering
To convert textual data into numerical features suitable for machine learning models, the Term Frequency–Inverse Document Frequency (TF-IDF) technique was used.

TF-IDF captures the importance of words in a document relative to the entire corpus. Both unigrams and bigrams were considered, and common English stopwords were removed.

The resulting feature vectors represent each problem statement as a high-dimensional sparse vector, which is well-suited for linear classifiers and tree-based regression models

Classification Model
The classification task aims to predict the difficulty class of a programming problem as Easy, Medium, or Hard.

Two models were evaluated:

- Logistic Regression (baseline model)
- Linear Support Vector Machine (SVM)

The dataset was split into training and testing sets using an 80–20 split, with stratification to preserve class distribution.

Model performance was evaluated using accuracy, precision, recall, and confusion matrix.

Regression Model
The regression task predicts a numerical difficulty score for each programming problem.

A Random Forest Regressor was used due to its ability to model non-linear relationships and handle high-dimensional feature spaces.

The model was trained using an 80–20 train-test split and evaluated using Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

Web Interface
A web-based interface was developed using Streamlit to demonstrate the system's functionality.

The interface allows users to:

- Paste the problem description
- Paste input and output specifications
- Obtain predicted difficulty class and score

The application loads pre-trained models and performs inference in real time. No external hosting is required, and the app runs locally.

Experimental Setup
All experiments were conducted on a local machine using Python and the scikit-learn library.

- Feature extraction: TF-IDF
- Classification models: Logistic Regression, Linear SVM
- Regression model: Random Forest Regressor
- Evaluation metrics: Accuracy, Confusion Matrix, MAE, RMSE

All reported results correspond exactly to the submitted code and trained models.

Conclusion
This project presents an end-to-end system for predicting programming problem difficulty using textual descriptions alone. The proposed approach combines NLP techniques with machine learning models to perform both classification and regression tasks.

While predicting difficulty from text is inherently challenging due to subjectivity, the system demonstrates reasonable performance and provides a practical automated solution. Future work may explore incorporating semantic embeddings or problem-solving metadata to further improve accuracy.