AutoJudge: Predicting Programming Problem Difficulty using NLP and Machine Learning

Project Report

January 3, 2026

Abstract

Online coding platforms host thousands of programming problems, typically categorized by difficulty levels such as Easy, Medium, and Hard. Traditionally, these labels are assigned manually by problem setters or through community feedback. This project introduces **AutoJudge**, an intelligent system designed to automate this process. Using Natural Language Processing (NLP) techniques and XGBoost models, AutoJudge predicts both the discrete difficulty class and a continuous numerical difficulty score based solely on the problem's textual description. Our approach utilizes custom feature engineering, including constraint extraction and keyword analysis, achieving robust performance across both classification and regression tasks.

1 Introduction

Competitive programming platforms such as Codeforces, CodeChef, and Kattis host a vast collection of algorithmic problems designed to evaluate problem-solving skills. Each problem is typically labeled with a difficulty level and often accompanied by a numerical difficulty score. These labels are commonly assigned through expert judgment or community feedback, making the process subjective and time-consuming.

The objective of this project is to design and implement **AutoJudge**, a machine learning-based system capable of automatically estimating the difficulty of programming problems using only their textual descriptions. The system performs two tasks simultaneously:

- **Classification:** Predicting whether a problem is **Easy**, **Medium**, or **Hard**.
- **Regression:** Predicting a continuous **Difficulty Score**.

In addition, a web-based interface is developed to allow real-time predictions for unseen problems.

2 Problem Statement

Given a dataset containing structured information about programming problems — including title, description, input specification, and output specification — the task is to learn a mapping from textual features to:

1. $Y_{class} \in \{\text{Easy}, \text{Medium}, \text{Hard}\}$
2. $Y_{\text{score}} \in [0, 10] \subset \mathbb{R}$

The core challenge lies in extracting latent complexity signals embedded within the text. These signals often appear in constraints (e.g., 10^5 suggesting $O(N \log N)$ solutions) or in algorithmic keywords such as *shortest path*, *dynamic programming*, and *segment tree*.

3 Dataset and Preprocessing

The dataset consists of JSONL samples containing structural programming problem data. Each sample includes the title, problem statement, and specific descriptions for input and output formats. To emphasize the importance of constraints, the input description is repeated three times during concatenation.

Text cleaning includes HTML removal, lemmatization, stopword filtering, and preservation of important mathematical symbols.

4 Feature Engineering

The model leverages TF-IDF features combined with domain-specific signals including text length, mathematical symbol count, extracted constraints, and binary indicators for competitive programming keywords. Trigram TF-IDF representation further enhances semantic understanding.

5 Model Architectures

Both classification and regression tasks utilize XGBoost. For classification, SMOTE is applied to address class imbalance. For regression, hyperparameters are optimized using RandomizedSearchCV.

6 Experimental Setup

All experiments were conducted using Python 3.14.2 with the `scikit-learn`, `xgboost`, and `streamlit` libraries. The dataset was split into 80% training and 20% testing subsets.

7 Results and Evaluation

7.1 Classification Performance

The classifier achieves an overall accuracy of **0.67**. Performance across classes is shown in Table 1 and Figure 1.

Table 1: Classification Metrics

Class	Precision	Recall	F1-score	Support
Easy	0.83	0.83	0.83	388
Medium	0.60	0.51	0.55	388
Hard	0.59	0.68	0.63	389
Accuracy			0.67	1165

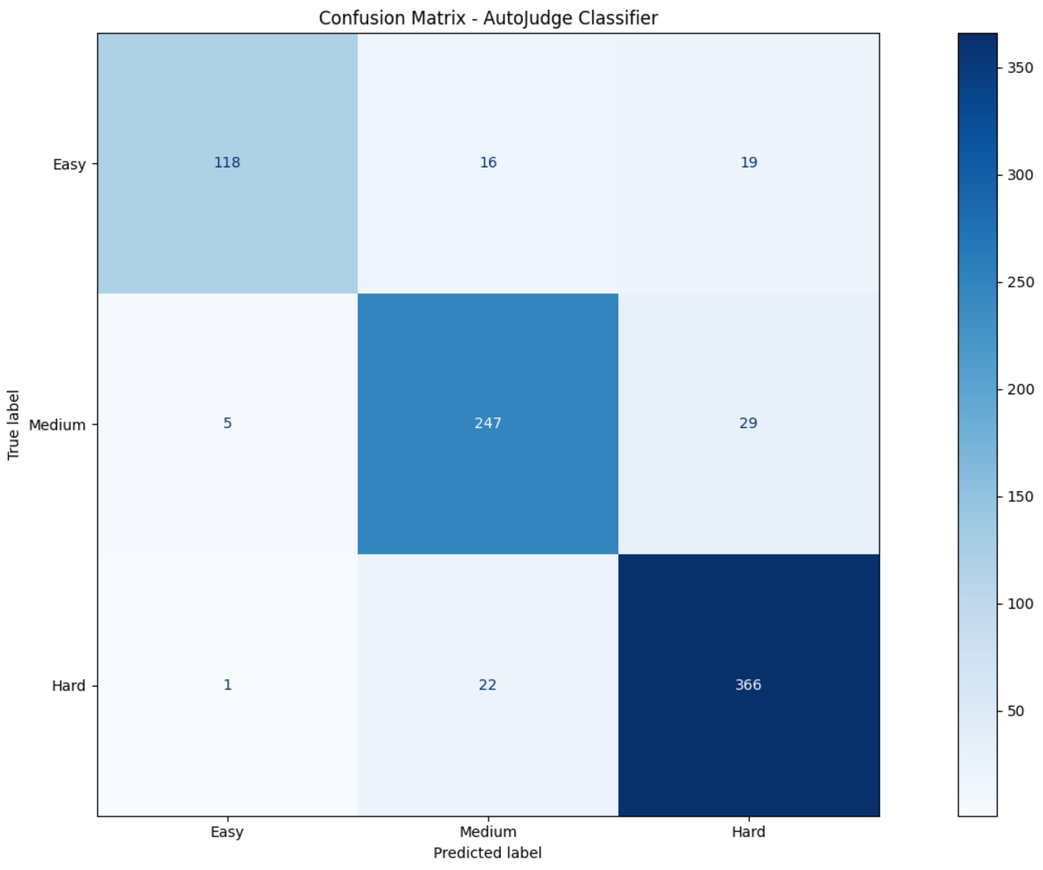


Figure 1: Confusion Matrix for Difficulty Classification

7.2 Regression Results

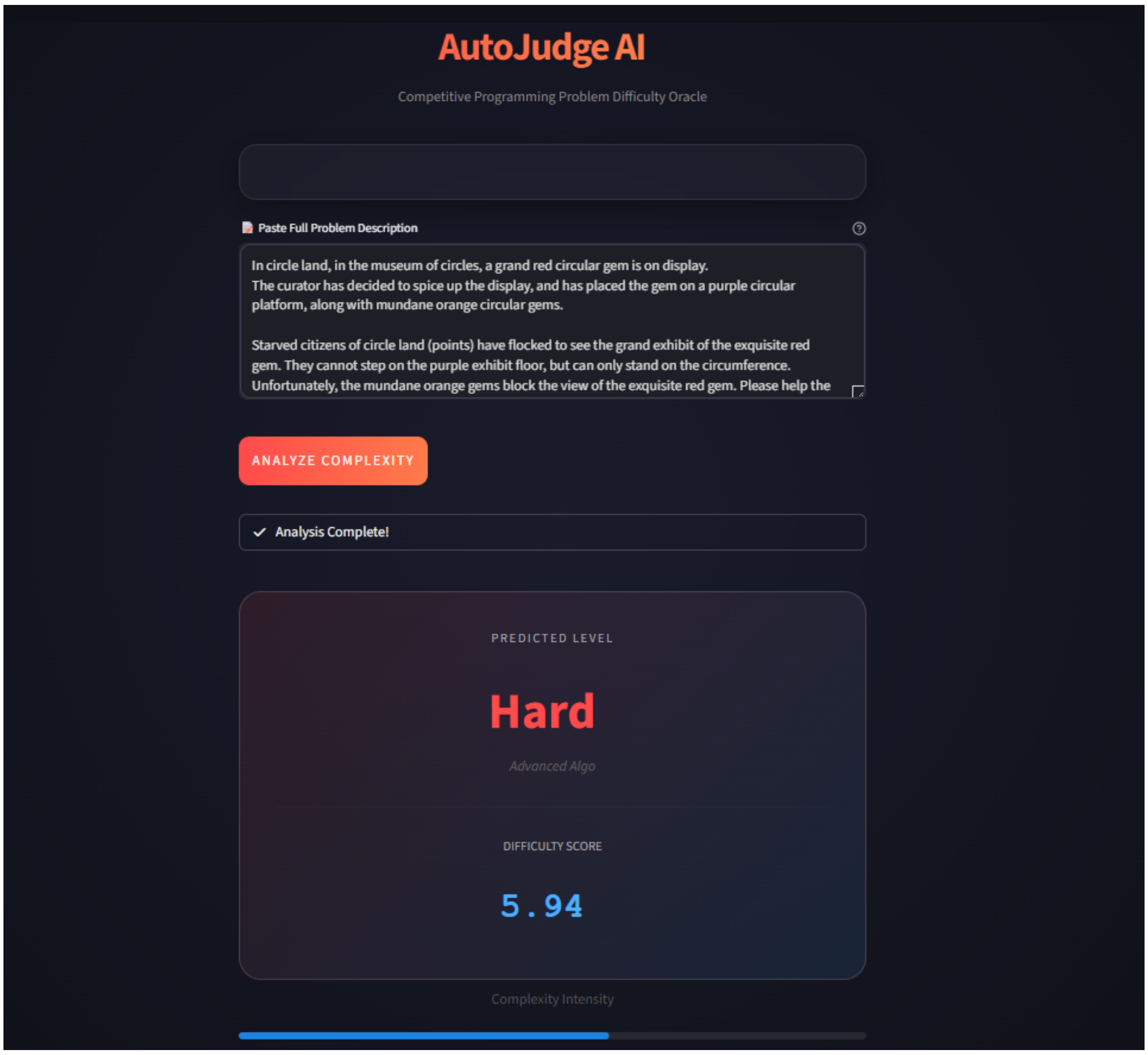
The regression model achieves MAE of **1.69** and RMSE of **2.01**.

Table 2: Regression Metrics

Metric	Value
MAE	1.69
RMSE	2.01

8 Web Interface

A Streamlit-based interface allows users to paste a problem description and obtain predictions in real time. The UI provides visual feedback and complexity indicators.



AutoJudge AI

Competitive Programming Problem Difficulty Oracle

Paste Full Problem Description

In circle land, in the museum of circles, a grand red circular gem is on display.
The curator has decided to spice up the display, and has placed the gem on a purple circular platform, along with mundane orange circular gems.

Starved citizens of circle land (points) have flocked to see the grand exhibit of the exquisite red gem. They cannot step on the purple exhibit floor, but can only stand on the circumference.
Unfortunately, the mundane orange gems block the view of the exquisite red gem. Please help the

ANALYZE COMPLEXITY

Analysis Complete!

PREDICTED LEVEL

Hard

Advanced Algo

DIFFICULTY SCORE

5.94

Complexity Intensity

Figure 2: AutoJudge Web Interface

9 Conclusions

AutoJudge demonstrates that programming problem difficulty can be accurately inferred from text alone. The system achieves strong classification and regression performance, providing a practical solution for automated difficulty assessment.