AutoJudge: Programming Problem Difficulty and Score Prediction Using Machine Learning

Divyansh Bansal
IIT Roorkee
B.Tech (Second Year)

2024-28

Contents

1 Introduction

Competitive programming platforms contain a large number of problems with varying difficulty levels. These difficulty labels are typically assigned manually, making them subjective and inconsistent. Automating difficulty estimation can help learners, instructors, and problem setters.

This project, **AutoJudge**, uses Natural Language Processing (NLP) and Machine Learning techniques to predict the difficulty of programming problems using only their textual descriptions. The system outputs both a categorical difficulty label and a continuous difficulty score.

2 Problem Statement

The objective of this project is to build a machine learning system that:

- Predicts the difficulty class (Easy / Medium / Hard)
- Predicts a numerical difficulty score (1–5)
- Provides interpretable insights for predictions

The main challenges include class imbalance, overlapping difficulty characteristics, and stable regression.

3 Dataset Description

The dataset consists of 4,112 competitive programming problems stored in JSON Lines format. Each entry contains the problem title, description, input/output details, a difficulty label, and a numeric difficulty score.

The dataset is imbalanced, with Hard problems appearing more frequently than Easy ones.

4 Data Preprocessing

The following preprocessing steps were applied:

- Lowercasing and text normalization
- Removal of special characters and extra spaces
- Concatenation of all textual fields
- Title weighted twice to emphasize importance

These steps ensure clean and consistent input for feature extraction.

5 Feature Engineering

5.1 Text Features

TF-IDF vectorization was applied using unigrams, bigrams, and trigrams with stop-word removal and sublinear term frequency scaling.

5.2 Handcrafted Features

Additional features include:

- Text length and word count
- Sentence statistics
- Mathematical symbol density
- Numeric constraint detection (e.g., $n \geq 10^5$)
- Algorithmic pattern detection (DP, graphs, BFS/DFS, binary search)

All numeric features were standardized using `StandardScaler`.

6 Model Architecture

6.1 Classification

Three classifiers were trained: Gradient Boosting, Random Forest, and Support Vector Machine. SMOTE was used to handle class imbalance. An ensemble approach combined the probabilistic outputs of these models.

6.2 Regression

To avoid unstable predictions, class-conditioned regression was implemented. Separate Random Forest regressors were trained for Easy, Medium, and Hard problems.

7 Experimental Setup

- Train-test split: 80% training, 20% testing
- Classification metrics: Accuracy, Precision, Recall, F1-score
- Regression metrics: MAE and RMSE

All experiments were conducted using Python and Scikit-learn.

8 Results and Evaluation

8.1 Classification Results

The ensemble classifier achieved an accuracy of **54.92%**.

Class	Precision	Recall	F1-score
Easy	0.52	0.41	0.45
Medium	0.45	0.32	0.38
Hard	0.60	0.77	0.67

Table 1: Classification Performance

Actual \Predicted	**Easy**	**Hard**	**Medium**
Easy	62	46	45
Hard	24	299	66
Medium	34	156	91

Table 2: Confusion Matrix for Difficulty Classification

The confusion matrix shows that most misclassifications occur between neighboring difficulty levels (Easy–Medium and Medium–Hard), which reflects the inherent ambiguity in difficulty labeling rather than random errors.

8.2 Regression Results

- MAE: 1.366
- RMSE: 1.775

These results indicate stable and realistic score predictions.

9 Web Interface

A Flask-based web application was developed to demonstrate real-time predictions.

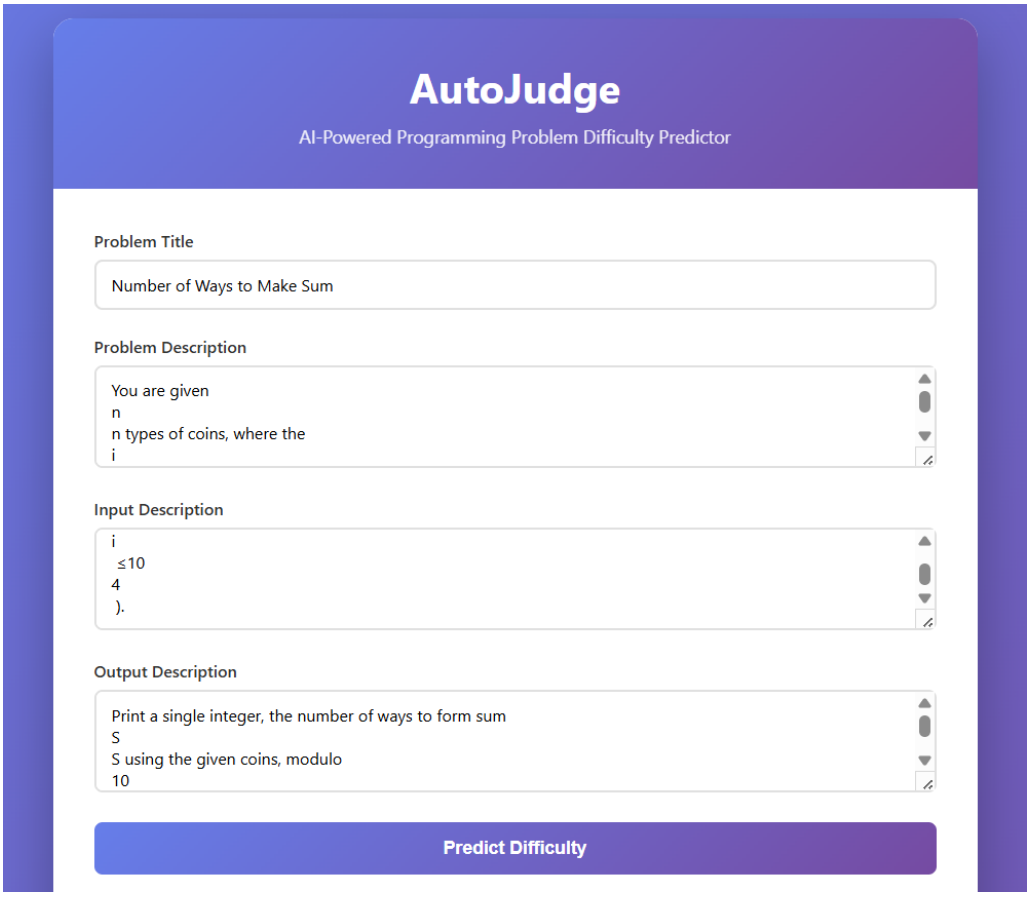

Figure 1: AutoJudge Web Interface

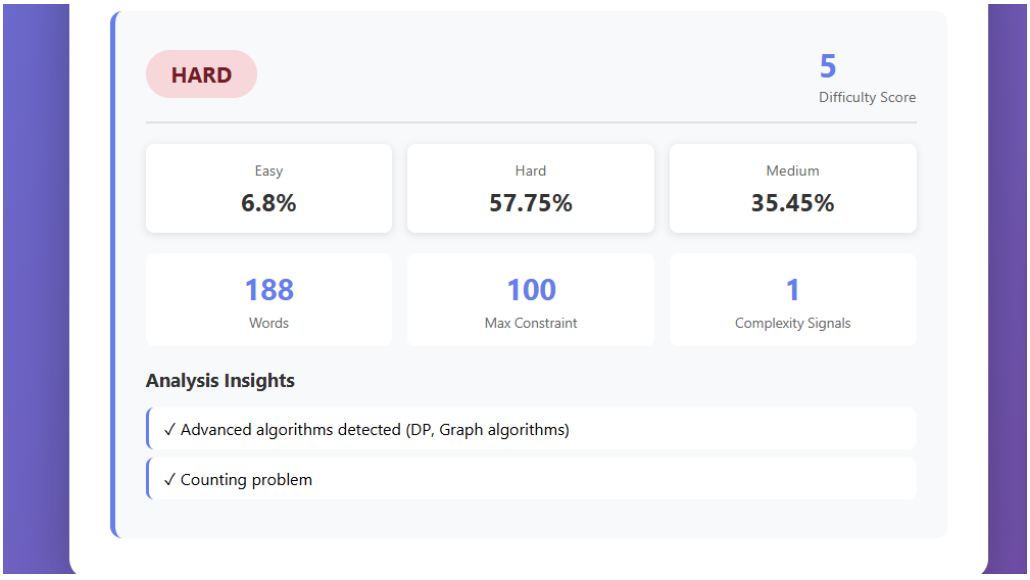

Figure 2: Predicted Difficulty and Score Output

The interface displays predicted difficulty, score, and algorithmic insights.

10 Sample Predictions

- Simple array problem: Easy, score ≈ 1.6
- Graph shortest path problem: Hard, score ≈ 4.8

11 Conclusion

AutoJudge demonstrates how NLP and machine learning can be used to estimate programming problem difficulty using only textual data. The ensemble classification and class-conditioned regression approach ensures stable and interpretable predictions.

Future improvements may include transformer-based models and the use of solution code.

12 References

- Scikit-learn Documentation
- NLP Feature Engineering Techniques
- Competitive Programming Platforms