

# **AutoJudge : Predicting Programming Problem Difficulty**

**Author:** Vishnu Bishnoi

**Date:** January 03, 2026

**GitHub Repository:** <https://github.com/vishnubishnoi17/MyAutoJudge>

## **1. Introduction and Problem Statement**

Online competitive programming platforms such as Code forces, Kattis, and Code Chef categorize problems into difficulty levels: Easy, Medium, or Hard, and often assign a numerical difficulty score. Traditionally, this classification relies on human judgment, contest organizers' experience, and post-contest user feedback (e.g., solve rates). This process can be subjective, time-consuming, and inconsistent.

The objective of this project is to develop an automated machine learning system, **MyAutoJudge**, that predicts:

1. Problem Class: Easy, Medium, or Hard (multi-class classification task).
2. Problem Score: A numerical difficulty rating on a 0–10 scale (regression task).

Predictions are based solely on the textual content of the problem, including the problem description, input description, and output description.

The system includes a user-friendly web interface built with Flask, allowing users to paste a new problem's text and receive instant predictions for both the class and score.

This addresses the need for an objective, scalable tool to assist platform administrators, educators, or contest setters in estimating problem difficulty upfront.

## **2. Dataset Description**

The dataset used is provided in JSONL format (data/problems\_data.jsonl) and consists of competitive programming problems with pre-labeled difficulties.

Total Samples: 4,112 problems (split into 3,289 training and 823 test samples).

Fields:

Field	Type	Description
title	string	Problem name
description	string	Full problem statement
input_description	string	Input format specification
output_description	string	Expected output format
problem_class	string	Difficulty label: "easy", "medium", or "hard"
problem_score	float	Numerical difficulty (0–10 scale)

Class Distribution (Test Set):

- Easy: 153 samples (18.6%)
- Medium: 281 samples (34.1%)
- Hard: 389 samples (47.2%)

Significant class imbalance, with Hard problems dominating.

Score Statistics: Range 1.10 to 9.70.

The dataset reflects real-world competitive programming problems in English. No additional datasets were used.

### 3. Data Preprocessing and Feature Engineering

#### *Data Preprocessing*

- Loaded from problems\_data.jsonl using src/data\_preprocessing.py.
- Combined relevant text fields: description, input\_description, and output\_description into a single concatenated text.
- Cleaned text: lowercasing, normalization, and handling of missing/NaN values.
- Saved processed data as data/processed\_data.csv.
- Train-test split: 80% training (3,289 samples), 20% test (823 samples).

#### *Feature Engineering*

A total of 540 features were engineered in src/feature\_engineering.py to capture textual indicators of difficulty:

1. Text Statistics (7 features): character count, word count, average word length, sentence count, and separate lengths for problem/input/output descriptions.

2. Mathematical Features (4 features): count of math operators, parentheses/brackets, numbers, and formula presence indicator.
3. Keyword Detection (29 features): binary/count features for algorithm-specific terms (e.g., "graph", "tree", "dfs", "bfs", "dp", "recursion", "greedy", "two-pointer", etc.).
4. TF-IDF Vectorization (500 features): bi-gram TF-IDF on concatenated text.

Features were scaled and saved separately for classification and regression pipelines.

## 4. Models Used and Experimental Setup

### *Models*

Traditional machine learning models from scikit-learn were used.

Classification: Random Forest (final model); compared with Logistic Regression and SVM.

Regression: Random Forest Regressor (final model); compared with Gradient Boosting Regressor and Linear Regression.

Training scripts: src/train\_classifier.py and src/train\_regressor.py. Pre-trained models saved in the models/ folder.

### *Experimental Setup*

- Environment: Python 3.8+, scikit-learn 1.3.2.
- Evaluation on hold-out test set (823 samples).
- Default Random Forest hyperparameters used.
- Consistent feature scaling during training and inference.

## 5. Results and Evaluation

### *Classification Results (Random Forest)*

Overall Accuracy: 51.15%

Classification Report:

Class	Precision	Recall	F1-Score	Support
Easy	0.60	0.24	0.34	153
Hard	0.53	0.86	0.66	389

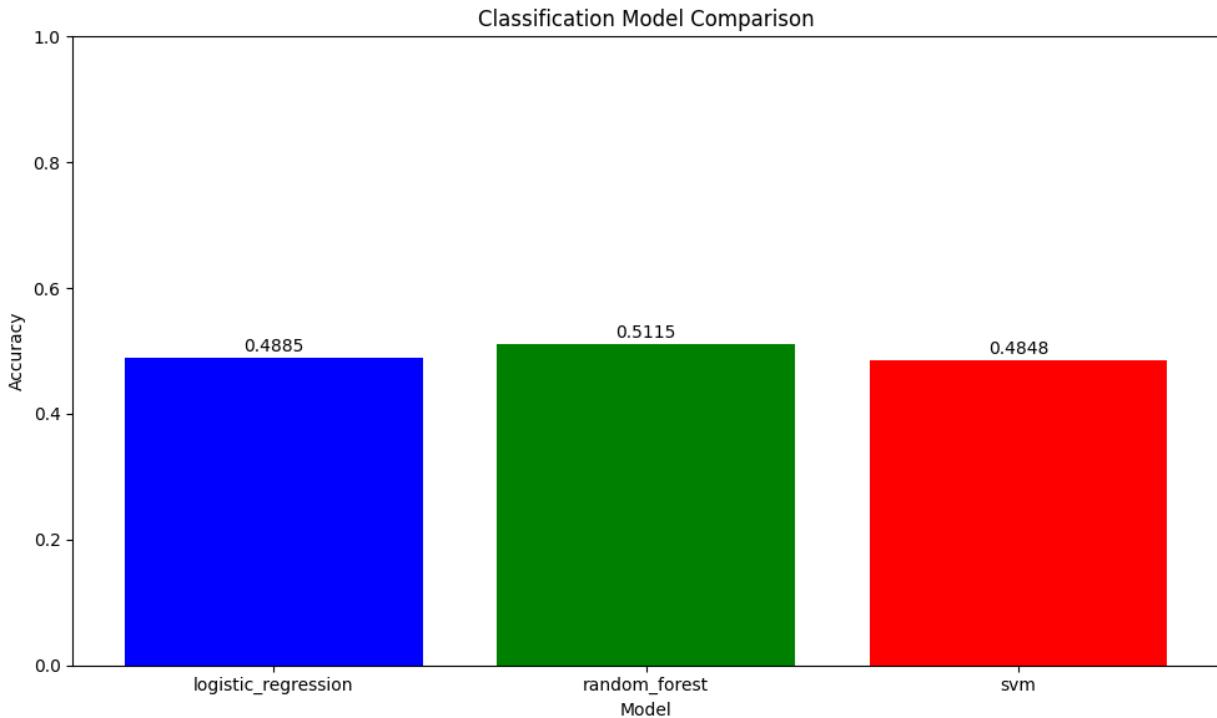
Medium	0.37	0.17	0.23	281
Accuracy			0.51	823

Confusion Matrix:

Predicted		text		
		Easy	Hard	Medium
Actual	Easy	37	77	39
	Hard	10	336	43
	Medium	15	218	48

Model Comparison (Accuracy):

Model	Accuracy
Random Forest	0.5115
Logistic Regression	0.4885
SVM	0.4848



*Classifier Model Comparison chart*

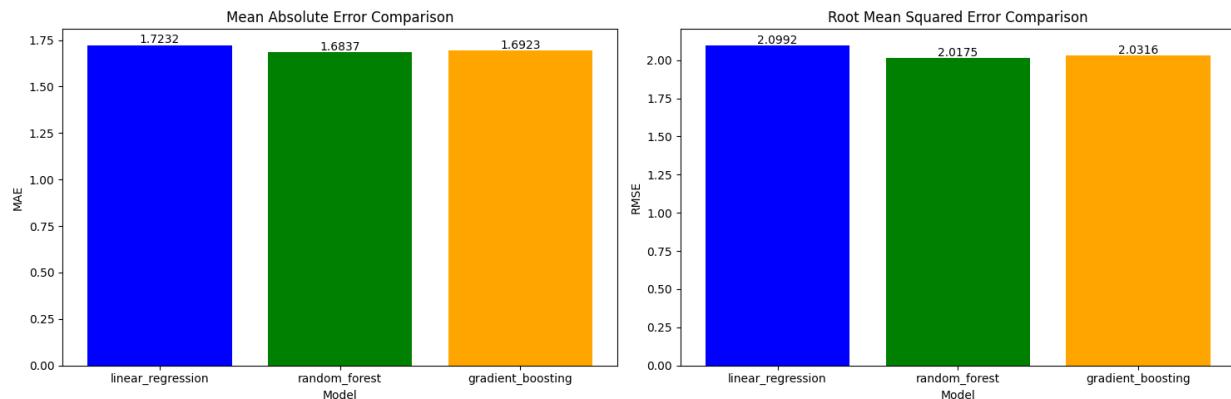
### **Regression Results (Random Forest)**

#### **Metric    Value**

MAE	1.6837
RMSE	2.0175
R <sup>2</sup>	0.1520

Model Comparison:

<b>Model</b>	<b>MAE</b>	<b>RMSE</b>	<b>R<sup>2</sup></b>
Random Forest	1.6837	2.0175	0.1520
Gradient Boosting	1.6923	2.0316	0.1402
Linear Regression	1.7232	2.0992	0.0820



*Regression Model Comparison Chart*

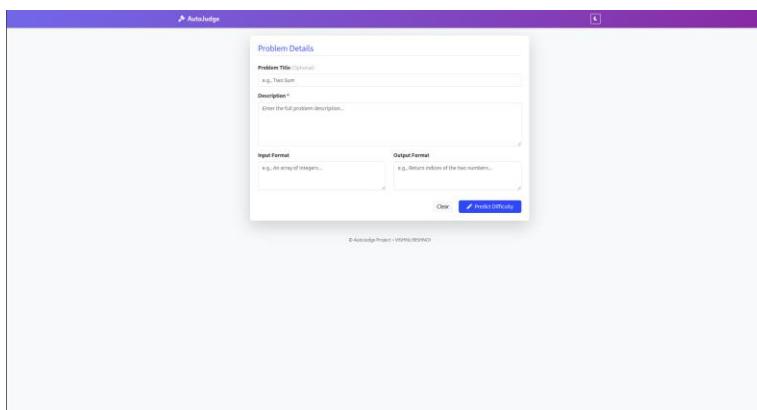
**Analysis:** The model performs well on the dominant Hard class but struggles with Easy and Medium due to imbalance. Regression error is approximately  $\pm 1.68$  on a 0–10 scale.

## 6. Web Interface

The web application is built with Flask (app/app.py) and provides a responsive UI.

**Features:**

- Separate text areas for problem description, input description, and output description.
- “Predict” button triggers real-time prediction via AJAX.
- Displays predicted class with confidence probabilities (progress bars) and numerical score.
- Additional UX elements: loading animation, form validation, reset button, mobile-responsive design.



*Web Interface*

**AutoJudge**

### Problem Details

**Problem Title** (Optional): Palindrome Number

**Description \***  
Given an integer  $x$ , return true if  $x$  is a palindrome, and false otherwise.  
An integer is a palindrome when it reads the same forward and backward. For example, 121 is a palindrome while 123 is not.

**Input Format**  
The input consists of a single integer  $x$  ( $-2^{31} \leq x \leq 2^{31} - 1$ ).

**Output Format**  
Output "true" if  $x$  is a palindrome, otherwise "false".

**Predict Difficulty**

### Analysis Result

<b>DIFFICULTY CLASS</b>	<b>COMPLEXITY SCORE</b>
<b>easy</b>	<b>3.63</b>

**CONFIDENCE LEVELS**

Easy		60.0%
Medium		25.0%
Hard		15.0%

### Sample Prediction Result 1

**AutoJudge**

### Problem Details

**Problem Title** (Optional): Persistent Segment Tree with Centroid Decomposition and Maximum Flow on Dynamic Graphs

**Description \***  
Constraints:  $n \leq 2 \times 10^5$ ,  $m \leq 3 \times 10^5$ , weights up to  $10^9$ , requiring overall  $O(m \log^2 n)$  time complexity using advanced techniques like persistent segment trees for range maximum queries on flow capacities, combined with suffix arrays for string-based vertex labeling in subproblems, and convex hull optimization for cost scaling in minimum cost flow variants if needed.  
You also need to handle multiple connected components merging with union-find with rollback for persistence.

**Input Format**  
Type 2: Delete edge  $uv$  if exists  
Type 3: Query  $s t$  version  $t$  ≤ version ≤ current operation count, compute max flow from  $s$  to  $t$  in the graph at that version

**Output Format**  
For each type 3 query, output a single integer: the maximum flow value from  $s$  to  $t$  in the specified historical graph version.

**Predict Difficulty**

### Analysis Result

<b>DIFFICULTY CLASS</b>	<b>COMPLEXITY SCORE</b>
<b>hard</b>	<b>5.34</b>

**CONFIDENCE LEVELS**

Easy		13.0%
Medium		54.0%
Hard		33.0%

### Sample Prediction Result 2

The application runs locally at <http://localhost:5000> after executing python app/app.py.

## 7. Conclusions and Limitations

The MyAutoJudge system successfully demonstrates automated difficulty prediction using textual features and traditional machine learning models. Random Forest proved to be the most effective model for both tasks.

Achievements:

- Complete dual-model pipeline with 540 engineered features.
- Functional, user-friendly web interface.
- Reasonable performance given class imbalance.

Limitations:

- Moderate classification accuracy (51%) and low recall for Easy/Medium classes.
- Regression explains only ~15% of variance ( $R^2 = 0.15$ ).
- Performance sensitive to description quality and language (English only).

Future Work:

- Incorporate advanced NLP embeddings (e.g., BERT).
- Apply techniques to handle class imbalance (SMOTE, class weights).
- Perform hyperparameter tuning and explore ensemble methods.
- Add model explainability (SHAP values) and online deployment options.

This project provides a solid foundation for automated problem difficulty estimation in competitive programming.