AutoJudge: Programming Problem Difficulty Prediction

January 8, 2026

1 Project Overview

AutoJudge is a Machine Learning–based system designed to automatically predict the difficulty of programming problems using their textual descriptions. Instead of relying on manual labeling, the system analyzes the problem statement text and estimates both a categorical difficulty level and a numerical difficulty score.

The model uses the following textual inputs:

- Problem Description
- Input Description
- Output Description

These text fields are combined and processed using Natural Language Processing (NLP) techniques. AutoJudge performs two tasks:

- **Classification** – Predicts whether a problem is Easy, Medium, or Hard
- **Regression** – Predicts a continuous difficulty score

The project aims to support competitive programming platforms, educators, and learners by providing an automated, consistent, and scalable difficulty estimation system.

2 Dataset Description

The dataset used in this project contains programming problems collected from competitive programming sources. Each data record includes both textual descriptions and difficulty annotations.

Each problem consists of:

- Problem title
- Problem description
- Input description
- Output description
- Difficulty class (Easy / Medium / Hard)
- Difficulty score (numeric value)

The dataset contains a total of **4112 programming problems**. No missing values were observed in the dataset.

3 Data Preprocessing

Data preprocessing was performed to clean and prepare the text for feature extraction. The following steps were applied:

- Combined problem description, input description, and output description into a single text field
- Converted all text to lowercase
- Removed special characters and punctuation
- Removed extra whitespace
- Ensured consistency across all samples

These steps help reduce noise and improve the quality of extracted features.

4 Exploratory Data Analysis (EDA)

Exploratory Data Analysis was conducted to understand the dataset distribution and patterns.

Key observations from EDA include:

- The dataset shows class imbalance, with Medium difficulty problems being the most frequent
- Difficulty scores are unevenly distributed, indicating varying problem complexity
- Text lengths vary significantly across problems, motivating the use of length-based features

The following visualizations were generated and saved:

- Difficulty class distribution bar chart

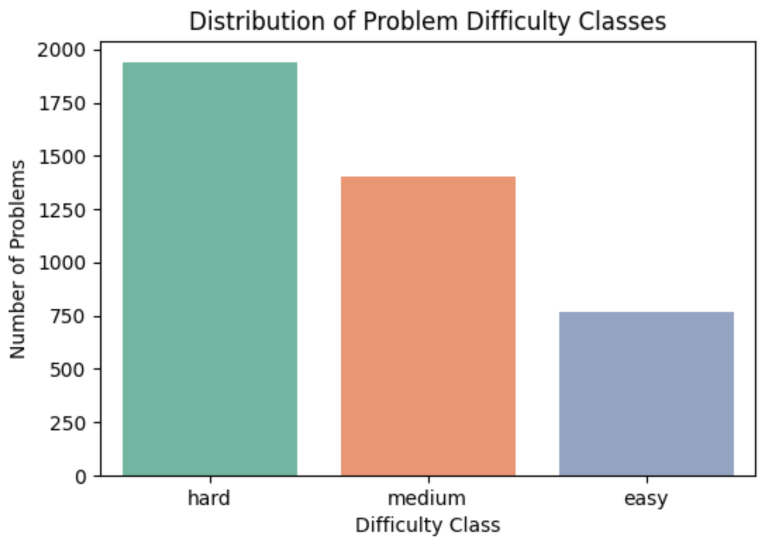


Figure 1: Distribution of Difficulty Classes

- Difficulty score distribution histogram

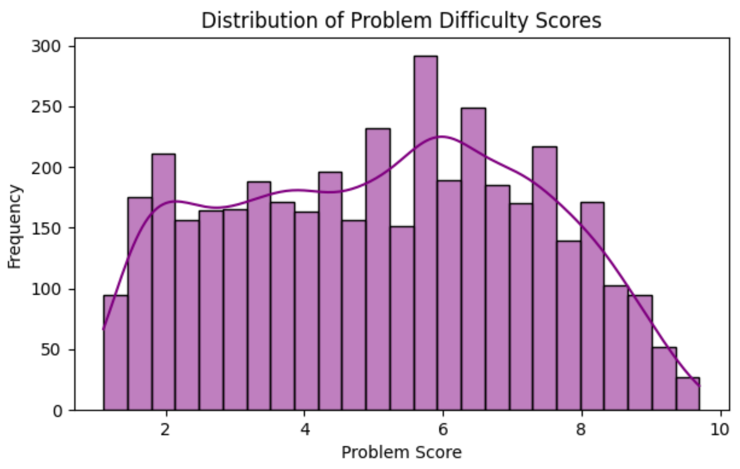


Figure 2: Distribution of Difficulty Classes

- Text length distribution

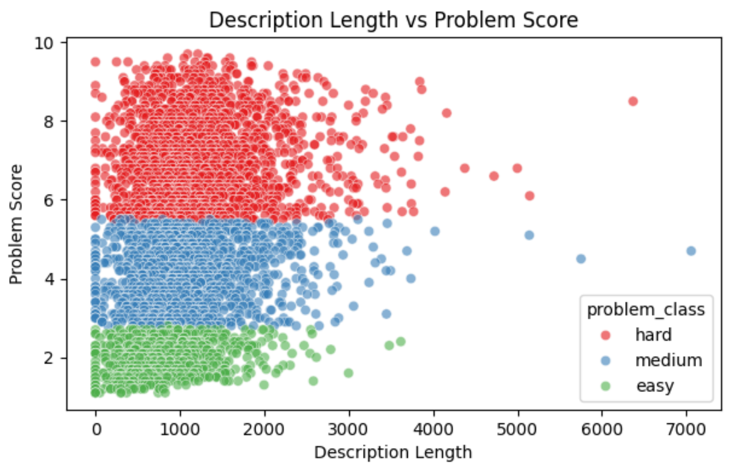


Figure 3: Distribution of Difficulty Classes

- Confusion matrices for classification models

5 Feature Engineering

Two types of features were extracted from the dataset.

5.1 Text-Based Features

TF-IDF (Term Frequency–Inverse Document Frequency) vectorization was applied to the cleaned text. Both unigrams and bigrams were used to capture contextual information from problem statements.

5.2 Handcrafted Features

In addition to TF-IDF features, the following handcrafted features were added:

- Total text length
- Count of mathematical symbols (+, -, *, /, =, <, >, ^)
- Frequency of key programming-related keywords such as:
  - graph
  - dp
  - tree
  - recursion
  - greedy

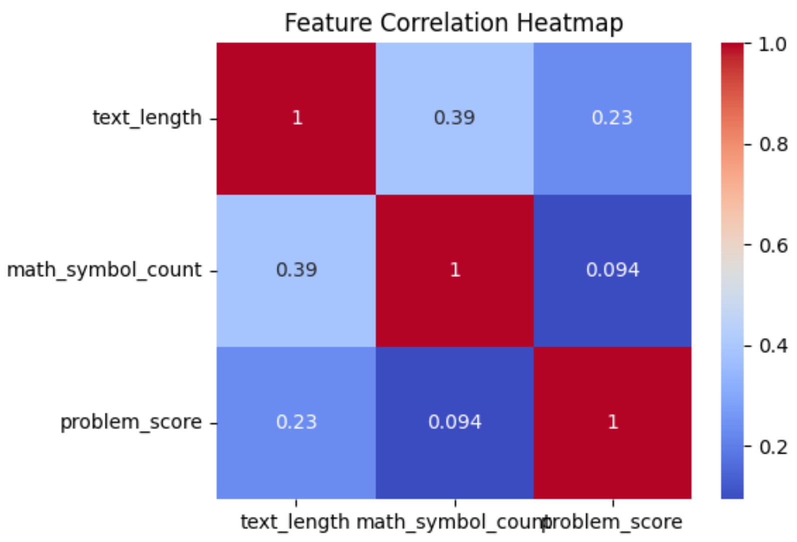


Figure 4: Distribution of Difficulty Classes

These features help capture structural and semantic complexity beyond raw text.

6 Label Encoding

The target variable for classification consists of categorical difficulty labels:

- Easy
- Medium
- Hard

Label encoding was applied as follows:

- Easy → 0
- Medium → 1
- Hard → 2

7 Model Development

Two separate modeling tasks were performed: classification and regression.

7.1 Classification Models

The following models were trained and evaluated:

- Logistic Regression
- Linear Support Vector Machine (SVM)
- Random Forest Classifier

7.2 Regression Models

The following regression models were evaluated:

- Linear Regression
- Random Forest Regressor
- Gradient Boosting Regressor

All models were trained using the same feature set to ensure fair comparison.

8 Model Comparison and Final Selection

8.1 Classification Models Comparison

Model	**Accuracy**	**Key Observations**
Logistic Regression	0.49	Baseline performance, affected by class imbalance
Linear SVM	0.47	Unstable predictions across classes
Random Forest Classifier	0.52	Best overall performance, captures non-linear patterns

Table 1: Classification Model Comparison

Final Classification Model Selected: Random Forest Classifier

8.2 Confusion Matrix Analysis

The confusion matrix for the final selected classification model (Random Forest Classifier) is shown below. It illustrates the model's performance across Easy, Medium, and Hard difficulty classes.

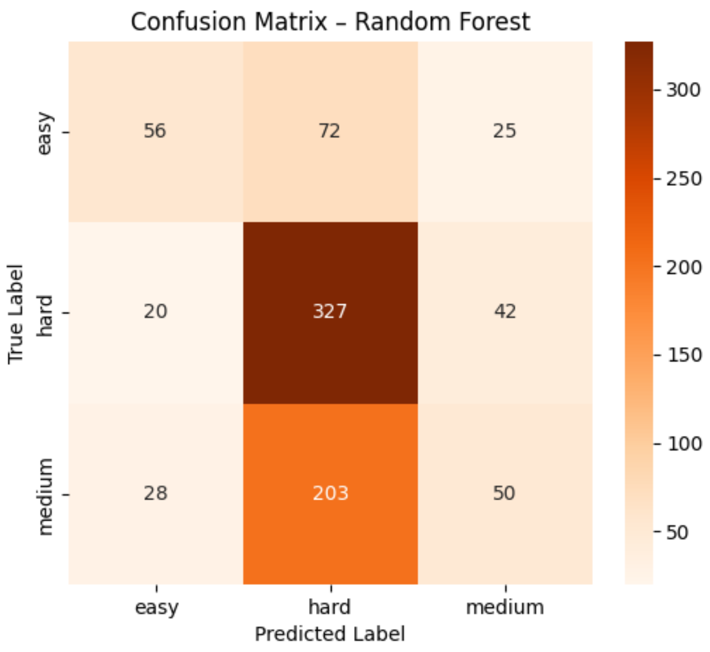


Figure 5: Confusion Matrix – Random Forest Classifier

The model demonstrates stronger predictive performance for Medium and Hard difficulty problems, which aligns with the dataset distribution. Minor misclassifications occur primarily between adjacent difficulty levels.

8.3 Regression Models Comparison

Model	MAE	RMSE	R^2	Key Observations
Linear Regression	2.38	2.97	-0.81	Poor baseline, fails on non-linear patterns
Random Forest Regressor	1.70	2.04	0.14	Lowest error, best performance
Gradient Boosting Regressor	1.71	2.04	0.14	Comparable to Random Forest

Table 2: Regression Model Comparison

Final Regression Model Selected: Random Forest Regressor

9 Web Application

The web application is implemented in `app.py` using Streamlit. It loads the pre-trained models and preprocessing objects saved as `.pkl` files. Users input the problem description, input, and output format. The script computes TF-IDF features, handcrafted features, and then performs:

- Classification to predict difficulty class
- Regression to predict difficulty score

Results are displayed instantly in the interface.

10 Running the Project Locally

To run the project locally:

```
source ~/myenv/bin/activate
python -m streamlit run app.py
```

10.1 Sample Prediction

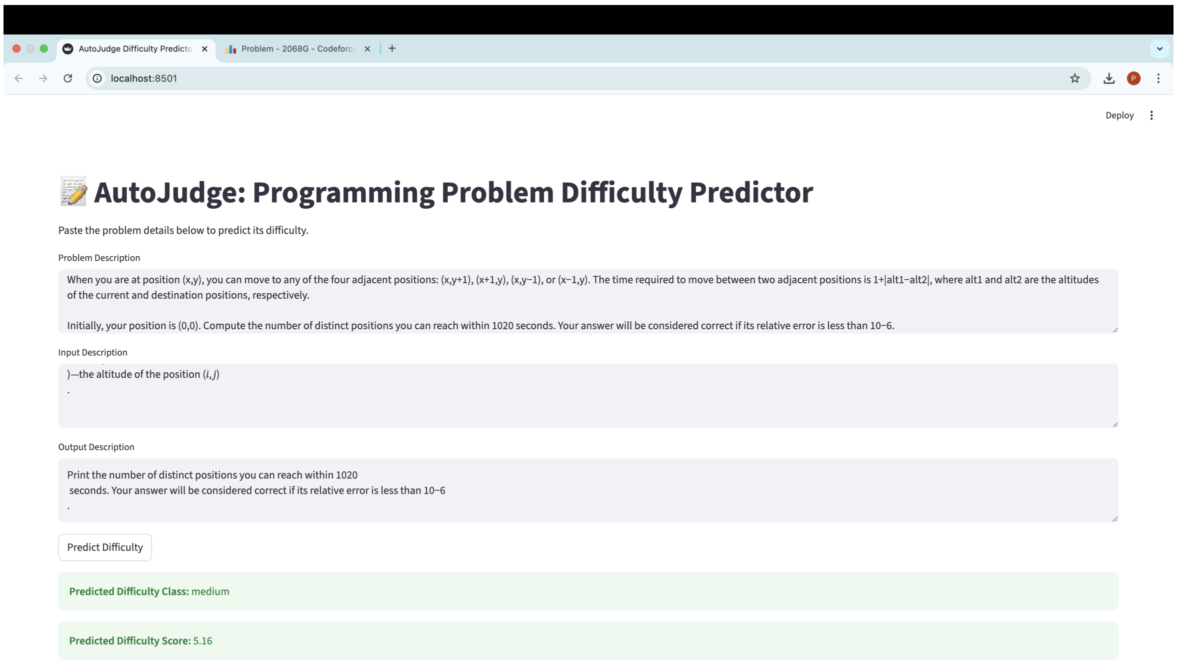
To demonstrate the functionality of the web application, a sample programming problem was tested through the Streamlit interface. The problem used for this demonstration was taken from **Codeforces**:

- **Problem Name:** G. A Very Long Hike
- **Source:** Codeforces

The problem description, input format, and output format were provided as inputs to the web interface. Based on the textual analysis, the system produced the following predictions:

- **Predicted Difficulty Class:** Medium
- **Predicted Difficulty Score:** 5.16

A screenshot of the web application displaying the prediction results is shown below.



AutoJudge: Programming Problem Difficulty Predictor

Paste the problem details below to predict its difficulty.

Problem Description

When you are at position (x,y), you can move to any of the four adjacent positions: (x,y+1), (x+1,y), (x,y−1), or (x−1,y). The time required to move between two adjacent positions is 1+|alt1−alt2|, where alt1 and alt2 are the altitudes of the current and destination positions, respectively.

Initially, your position is (0,0). Compute the number of distinct positions you can reach within 1020 seconds. Your answer will be considered correct if its relative error is less than 10−6.

Input Description

)—the altitude of the position (*i*,*j*)
.

Output Description

Print the number of distinct positions you can reach within 1020
 seconds. Your answer will be considered correct if its relative error is less than 10−6
.

Predict Difficulty

Predicted Difficulty Class: medium

Predicted Difficulty Score: 5.16

Figure 6: Sample Prediction Output from AutoJudge Web Application

11 Saved Models

The following trained models and preprocessing objects were saved using joblib:

- TF-IDF Vectorizer
- Label Encoder
- Random Forest Classifier
- Random Forest Regressor

12 Conclusion

This project demonstrates how NLP and machine learning techniques can be used to automatically estimate the difficulty of programming problems from textual descriptions. Random Forest models achieved the best performance for both classification and regression tasks.

Author

Name: Payili Meenakshi
Enrollment Number: 23323028

GitHub Repository:
https://github.com/payilimeenakshi2/AutoJudge-Difficulty-Predictor
Demo Video:
Google Drive Demo Video