AutoJudge: Programming Problem Difficulty Prediction Using Machine Learning

---

Abstract

Difficulty classification of programming problems is an important aspect of competitive programming platforms, as it helps learners choose appropriate problems and assists in contest organization. Traditionally, difficulty labels are assigned manually or inferred from user feedback, making the process subjective and inconsistent.

This project presents **AutoJudge**, a machine learning–based system that predicts the difficulty of programming problems using only textual descriptions. The system performs two tasks: classification of problems into Easy, Medium, or Hard categories, and regression-based prediction of a relative numerical difficulty score. The approach relies on natural language processing techniques and classical machine learning models. A Flask-based web interface is also developed to demonstrate real-time predictions along with confidence and explanation.

---

1. Introduction and Problem Statement

Online competitive programming platforms such as Codeforces, CodeChef, and Kattis host a large collection of programming problems. These problems are generally labeled with difficulty levels such as Easy, Medium, and Hard, which guide users in selecting problems suitable for their skill level. Difficulty labels are also used for contest design and recommendation systems.

Currently, difficulty labeling is primarily performed through **manual judgment** or derived indirectly from **user submission statistics and feedback**. This process is inherently subjective and may vary across platforms. As a result, problems with similar algorithmic complexity may receive different difficulty labels.

The objective of this project is to **automatically predict the difficulty of programming problems using only their textual descriptions**, without relying on solution code or submission history. The system aims to:

- Predict a categorical difficulty label (Easy / Medium / Hard)
- Predict a relative numerical difficulty score

This project demonstrates how textual analysis and machine learning can be used to estimate problem difficulty in an automated and scalable manner.

---

2. Dataset Description

The dataset used in this project consists of competitive programming problems collected from online platforms. The dataset is publicly available at the following link:

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

The dataset is provided in **JSONL (JSON Lines)** format, where each line corresponds to one programming problem.

2.1 Dataset Attributes

Each data sample contains the following fields:

Field Name	Description
title	Problem title
description	Main problem statement
input_description	Input format
output_description	Output format
sample_io	Sample input/output
problem_class	Difficulty class (Easy / Medium / Hard)
problem_score	Numerical difficulty score
url	Problem source URL

The dataset already includes labeled difficulty classes and scores. Therefore, **no manual labeling was required** for this project.

---

3. Data Preprocessing

Before training machine learning models, several preprocessing steps were applied to ensure data consistency and usability.

All textual fields, including the title, description, input description, output description, and sample input/output, were **combined into a single text representation**. This ensures that the full problem context is captured in one input.

Missing values in any of the textual fields were handled by replacing them with empty strings. This approach prevents data loss while maintaining consistency across samples. Basic normalization was applied to ensure uniform text formatting.

These preprocessing steps prepare the raw dataset for effective feature extraction and model training.

---

4. Feature Engineering

Feature engineering is a crucial step in transforming raw textual data into numerical representations suitable for machine learning models. Two types of features were used in this project.

4.1 Textual Features (TF-IDF)

Textual features were extracted using **TF-IDF (Term Frequency–Inverse Document Frequency)**. TF-IDF highlights important terms in a document while reducing the weight of commonly occurring words. This representation captures semantic information present in programming problem statements.

TF-IDF is well suited for text-based classification and regression tasks where word importance plays a significant role.

4.2 Handcrafted Features

In addition to TF-IDF vectors, several handcrafted features were introduced to capture structural complexity. These include:

- Length of the combined problem text
- Count of numeric values and mathematical symbols
- Frequency of algorithmic keywords such as *graph*, *dfs*, *bfs*, *dp*, and *recursion*

These handcrafted features complement TF-IDF by explicitly encoding algorithmic hints and complexity indicators.

---

5. Models Used and Experimental Setup

The project performs two machine learning tasks: classification and regression. Both tasks use the same feature extraction pipeline.

5.1 Classification Model

A **Random Forest Classifier** was used to predict the difficulty class (Easy / Medium / Hard). Random Forest was chosen due to its robustness, ability to handle non-linear relationships, and strong performance on mixed feature sets.

5.2 Regression Model

A **Random Forest Regressor** was used to predict a relative numerical difficulty score. The regression output provides a continuous measure of difficulty rather than discrete labels.

5.3 Experimental Setup

The dataset was split into training and testing sets using a standard hold-out strategy. The same feature pipeline was used for both classification and regression tasks. Default hyperparameters were retained to reduce the risk of overfitting. No deep learning models were used, in accordance with project guidelines.

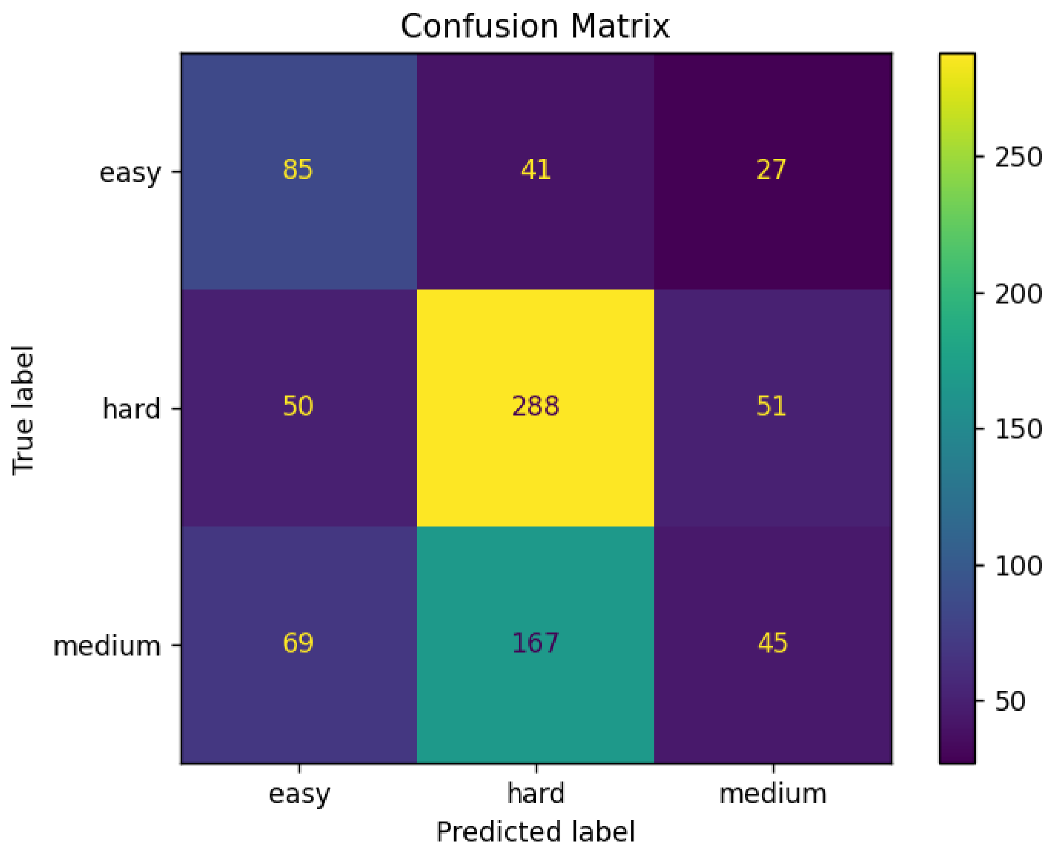
---

6. Evaluation and Results

6.1 Classification Results

The classification model was evaluated using **accuracy** and **confusion matrix**. The achieved classification accuracy was approximately **50%**.

For a three-class classification problem, random guessing would result in an accuracy of approximately 33%. Therefore, the model performs significantly better than the random baseline, especially considering the subjective nature of difficulty labels.



The confusion matrix shows that the model performs best in identifying medium- and hard-difficulty problems, with a high number of correct predictions along the diagonal. Most misclassifications occur between neighboring difficulty classes, such as Easy being predicted as Medium and Medium being predicted as Hard. This behavior is expected, as difficulty is inherently subjective and adjacent classes often share similar characteristics. The relatively lower accuracy for extreme classes highlights the challenge of difficulty estimation using textual descriptions alone.

Most misclassifications occur between neighboring classes (Easy and Medium, Medium and Hard), which is expected for difficulty prediction tasks.

6.2 Regression Results

The regression model was evaluated using **Mean Absolute Error (MAE)** and **Root Mean Squared Error (RMSE)**.

Metric	**Value**
MAE	1.668
RMSE	2.018

The predicted scores are interpreted as **relative difficulty measures**, not absolute difficulty values.

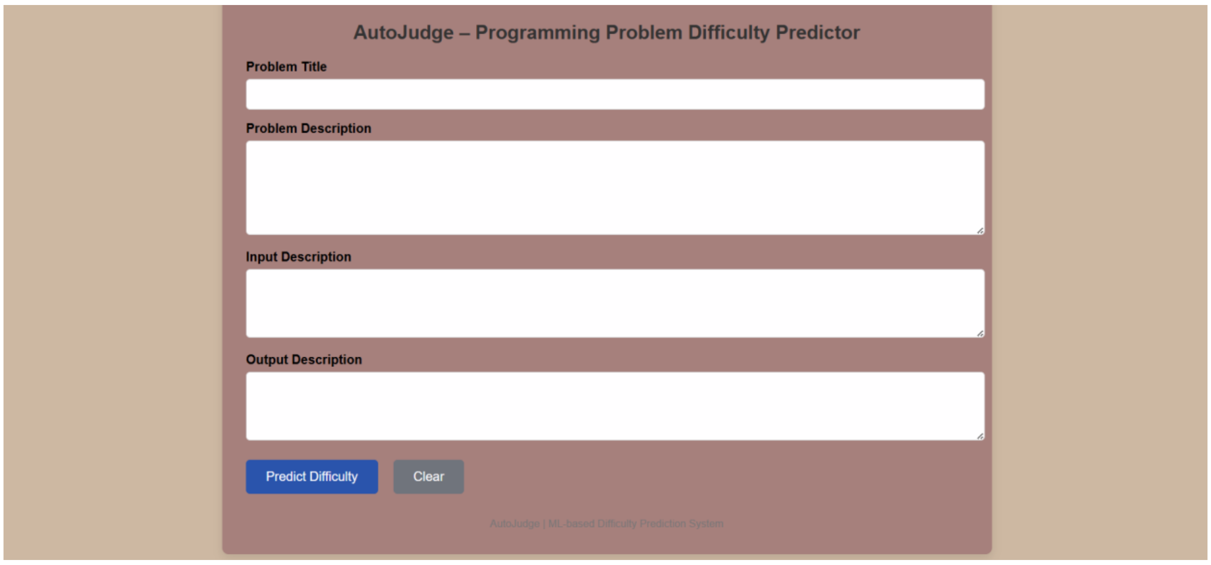
---

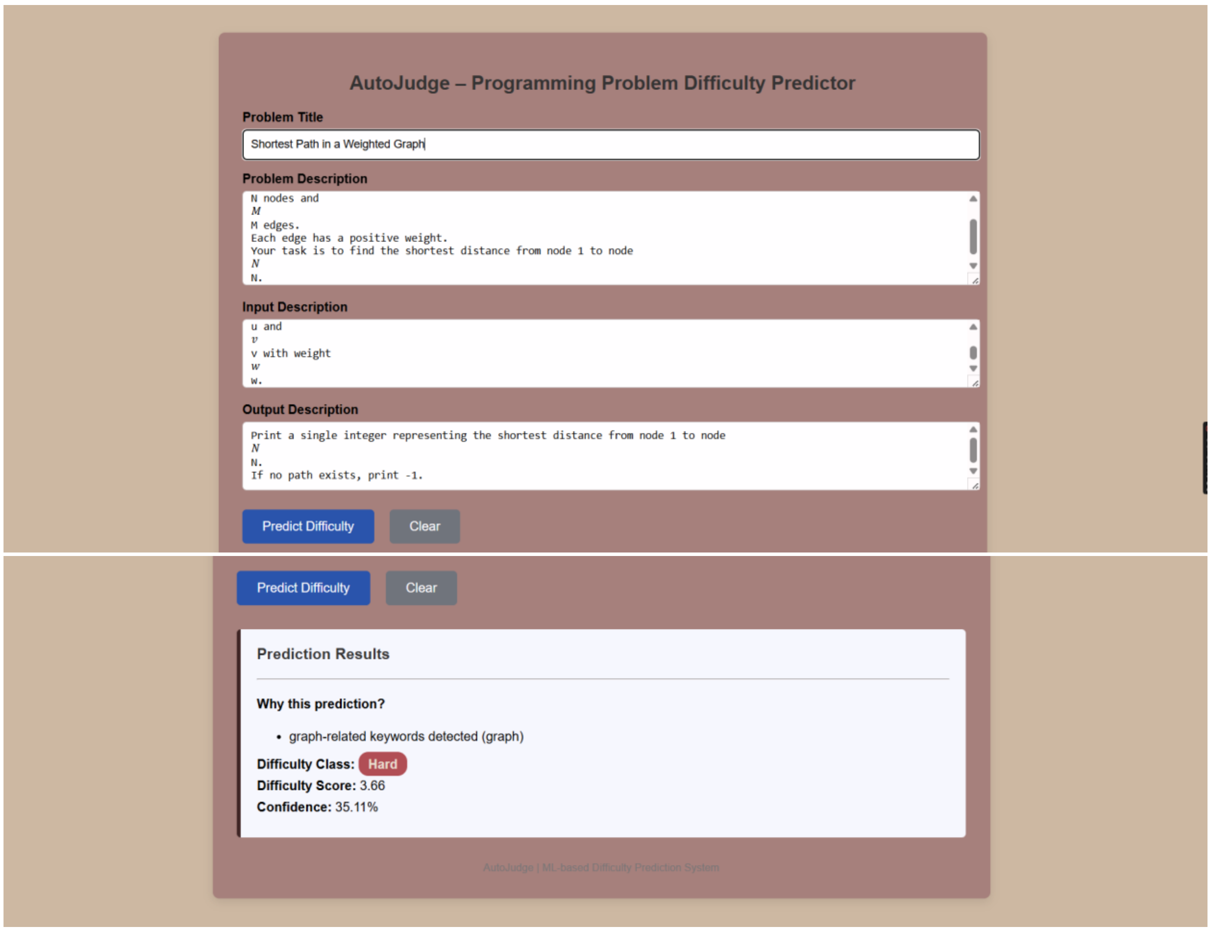
7. Web Interface

A **Flask-based web interface** was developed to demonstrate real-time predictions. The interface allows users to enter problem descriptions and obtain instant results.

Web Interface Features

- Input fields for problem title and descriptions
- Predict button
- Display of predicted difficulty class
- Relative difficulty score
- Prediction confidence
- Explanation of prediction


AutoJudge – Programming Problem Difficulty Predictor
Problem Title
Problem Description
Input Description
Output Description
Predict Difficulty
Clear
AutoJudge | ML-based Difficulty Prediction System


AutoJudge – Programming Problem Difficulty Predictor

Problem Title

Shortest Path in a Weighted Graph

Problem Description

```
N nodes and
M
M edges.
Each edge has a positive weight.
Your task is to find the shortest distance from node 1 to node
N
N.
```

Input Description

```
u and
v
v with weight
w
w.
```

Output Description

```
Print a single integer representing the shortest distance from node 1 to node
N
N.
If no path exists, print -1.
```

Prediction Results

Why this prediction?

- graph-related keywords detected (graph)

Difficulty Class: Hard
Difficulty Score: 3.66
Confidence: 35.11%

AutoJudge | ML-based Difficulty Prediction System

The application runs locally using Flask's development server.

---

8. Sample Prediction

A sample programming problem description was entered into the web interface. The system predicted the difficulty class along with a relative difficulty score and confidence value. The explanation highlighted detected algorithmic keywords and text length, providing insight into the prediction.

---

9. Explainable Difficulty Prediction

To improve transparency, a lightweight explainability mechanism was implemented. The system highlights:

- Detected algorithmic keywords
- Text length and complexity
- Presence of numeric constraints

This explainability feature helps users understand **why** a particular difficulty was predicted, without affecting the underlying prediction logic.

10. Production Readiness

The application uses Flask's development server for local execution. For production deployment, the same application can be served using a production WSGI server such as **Gunicorn**, with debug mode disabled, without changing application logic.

11. Conclusion and Future Work

Conclusion

This project demonstrates an end-to-end machine learning system for predicting programming problem difficulty using textual information alone. It integrates data preprocessing, feature engineering, classification, regression, evaluation, and a working web interface.

Despite the subjective nature of difficulty labels, the system provides meaningful predictions and interpretability.

Future Work

- Training on larger datasets
- Exploring deep learning-based NLP models
- Incorporating submission statistics
- Improving explainability techniques

Author Details

Name: Ronak Kumar
Enrollment No: 23113130
Department: Civil Engineering (3rd Year)
GitHub: https://github.com/ronak-kumar06