AUTOJUDGE
Predicting Programming Problem Difficulty

ACM Student Chapter
Open Projects 2025-26

Submitted by:

Name : Ammy Sunil Meshram
Enrollment : 22118009 (Fourth year)
Department of Metallurgical & Materials Engineering
Indian Institute of Technology, Roorkee

Date of Submission:

08th January 2026

INDIAN INSTITUTE OF TECHNOLOGY, ROORKEE

TABLE OF CONTENTS

1. INTRODUCTION

Online competitive programming platforms such as Codeforces, CodeChef, and Kattis host thousands of programming problems categorized by difficulty levels such as *Easy*, *Medium*, and *Hard*. Additionally, many platforms assign a numerical difficulty score to problems to better reflect their relative complexity. Traditionally, these difficulty labels and scores are determined using a combination of human judgment and user feedback, which can be subjective, inconsistent, and slow to adapt.

With the growing number of problems being added daily, there is a strong need for an automated, scalable approach to estimate problem difficulty. This project, **AutoJudge**, aims to address this challenge by developing a machine learning–based system that predicts the difficulty of programming problems using only their textual descriptions.

The system performs two tasks:

1. **Classification** – Predicting the difficulty class (*Easy*, *Medium*, or *Hard*)
2. **Regression** – Predicting a numerical difficulty score

The predictions are based solely on problem text, including:

1. Problem description
2. Input specification
3. Output specification

No information about user submissions, solution code, or platform statistics is used. The final system also includes a simple web interface that allows users to input a new problem description and instantly obtain predicted difficulty values.

2. PROBLEM STATEMENT & OBJECTIVES

2.1 Problem Statement

The goal of this project is to design an intelligent system that automatically predicts the difficulty of programming problems using textual information alone. The system must be capable of handling raw problem descriptions and output meaningful difficulty predictions without human intervention.

2.2 Project Objectives

1. Predict the **difficulty class** (Easy / Medium / Hard)
2. Predict a **numerical difficulty score**
3. Use **only textual features** extracted from problem statements
4. Provide predictions through a **simple web-based user interface**
5. Run locally without errors and without requiring online deployment

3. DATASET DESCRIPTION

The dataset used in this project consists of programming problems collected from online competitive programming platforms. Each data sample contains structured textual information along with corresponding difficulty labels.

3.1 Dataset Fields

Each problem record includes:

1. `title`: Name of the problem
2. `description`: Main problem statement
3. `input_description`: Description of input format
4. `output_description`: Description of output format
5. `problem_class`: Difficulty category (Easy / Medium / Hard)
6. `problem_score`: Numerical difficulty score

The dataset was originally provided in **JSONL format** and converted into a **CSV file** for ease of preprocessing and model training.

3.2 Dataset Characteristics

1. Contains thousands of problem samples
2. Includes both categorical and numerical difficulty annotations
3. Textual fields vary significantly in length and structure
4. Difficulty scores follow a continuous numerical scale

No manual labeling was required, as all ground-truth labels were already provided.

4. DATA PREPROCESSING

Before training machine learning models, extensive preprocessing was applied to clean and standardize the textual data.

4.1 Text Cleaning

The following preprocessing steps were applied:

1. Conversion of all text to lowercase
2. Removal of special characters and punctuation
3. Removal of extra whitespace
4. Handling of missing or null values

4.2 Text Combination

To ensure that all relevant textual information contributes to the prediction, the following fields were concatenated into a single text feature:

1. Problem description
2. Input description
3. Output description

This combined text represents the full semantic content of a programming problem.

4.3 Label Handling

1. Difficulty scores were converted to numeric format
2. Invalid or missing scores were removed
3. Difficulty class labels were normalized for consistency

This preprocessing ensured clean and uniform input for feature extraction.

5. FEATURE ENGINEERING

Feature engineering plays a crucial role in converting raw text into numerical representations suitable for machine learning models.

5.1 TF-IDF Vectorization

The primary feature extraction technique used was **Term Frequency–Inverse Document Frequency (TF-IDF)**. TF-IDF captures the importance of words by considering:

1. How frequently a word appears in a problem
2. How rare the word is across all problems

Configuration used:

1. Maximum number of features: 5000
2. N-gram range: (1, 2)
3. English stopwords removed

5.2 Motivation for TF-IDF

TF-IDF is effective for textual classification and regression tasks because:

1. It highlights informative keywords
2. It reduces the impact of common but uninformative words
3. It performs well with linear and tree-based models

The resulting TF-IDF vectors form the input feature matrix for both classification and regression models.

6. MODEL ARCHITECTURE & TRAINING

6.1 Classification Model

To predict the difficulty class, **Logistic Regression** was used.

Reasons for selection:

1. Efficient for high-dimensional sparse data
2. Interpretable and stable baseline model
3. Performs well with TF-IDF features

The model was trained using an 80–20 train-test split with stratification to preserve class distribution.

6.2 Regression Model

To predict numerical difficulty scores, a **Random Forest Regressor** was used.

Reasons for selection:

1. Handles non-linear relationships effectively
2. Robust to noise in textual features
3. Provides strong performance without extensive tuning

The regression model was trained on the same TF-IDF features using a standard train-test split.

7. EVALUATION & RESULTS

7.1 Classification Evaluation

The classification model was evaluated using:

1. Accuracy
2. Confusion matrix
3. Precision, recall, and F1-score

Observed Results:

1. Accuracy of approximately **48%**
2. Overlap between Medium and Hard classes due to similar textual patterns

```
Classification Accuracy: 0.48116646415552855

Confusion Matrix:
[[ 31  78  44]
 [ 11 296  82]
 [ 18 194  69]]

Classification Report:
              precision    recall  f1-score   support

        Easy       0.52      0.20      0.29       153
        Hard       0.52      0.76      0.62       389
      Medium       0.35      0.25      0.29       281

    accuracy                           0.48       823
   macro avg       0.46      0.40      0.40       823
weighted avg       0.46      0.48      0.45       823
```

7.2 Regression Evaluation

The regression model was evaluated using:

1. Mean Absolute Error (MAE)
2. Root Mean Squared Error (RMSE)

Observed Results:

1. MAE = **1.7165717332970971**
2. RMSE = **2.0565047516733683**

These results indicate that the predicted scores are reasonably close to the actual difficulty values.

7.3 Discussion

While accuracy is moderate, the results are acceptable given:

1. The inherent ambiguity in textual difficulty
2. Overlapping descriptions across difficulty levels
3. Absence of solution-level information

8. WEB INTERFACE & SYSTEM DEPLOYMENT

A lightweight web application was built using **Flask** to demonstrate the system.

8.1 Interface Features

Text boxes for:

1. Problem description
2. Input description
3. Output description

Predict button to trigger the display of results.

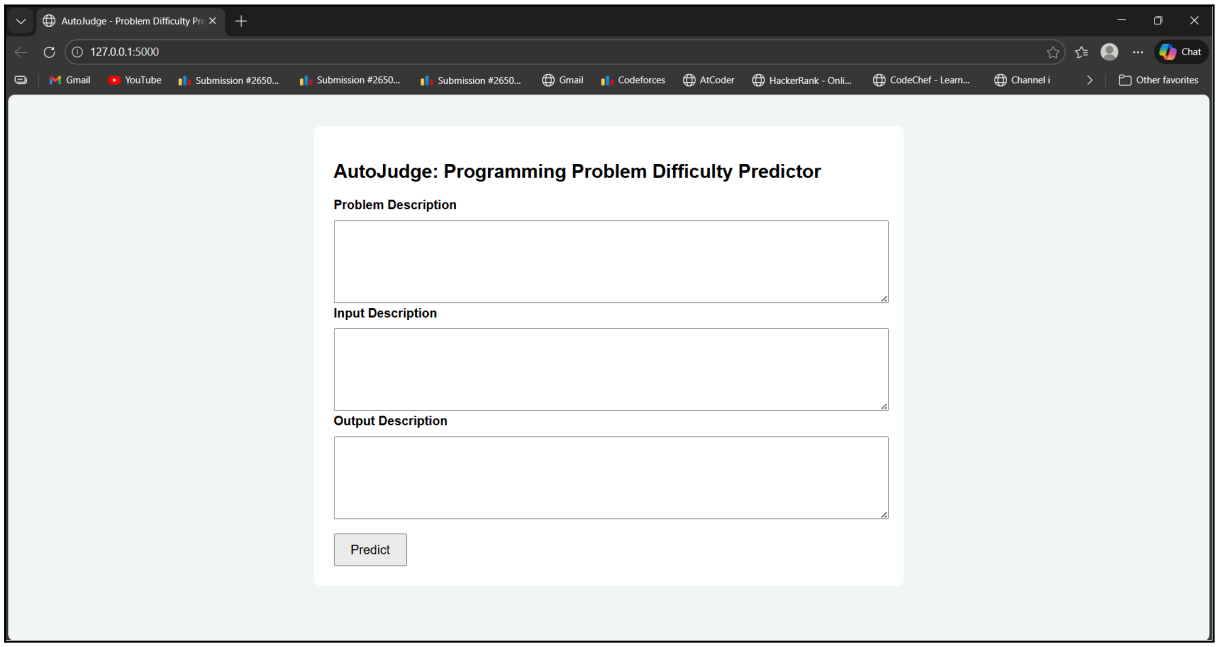
Display of:

1. Predicted difficulty class
2. Predicted difficulty score

8.2 Local Execution

The application runs locally using : `python app.py`

No hosting, authentication, or database integration is required.



9. CONCLUSION & FUTURE WORK

This project demonstrates that programming problem difficulty can be estimated using only textual descriptions. The AutoJudge system successfully performs both classification and regression tasks and provides predictions through a simple, user-friendly interface.

Limitations

1. Relies solely on textual features
2. Cannot capture algorithmic complexity directly
3. Performance affected by ambiguous or verbose descriptions

Future Enhancements

1. Incorporating additional linguistic features
2. Exploring advanced models such as gradient boosting
3. Improving class separation using hybrid features