AutoJudge

Name- Shubhi Gupta

Enrollment No- 24116093

Branch and year- ECE 2Y

Online competitive programming platforms such as Codeforces, CodeChef, and Kattis host thousands of programming problems that are categorized by difficulty levels such as Easy, Medium, and Hard. These difficulty ratings play an important role in guiding learners, recommending problems, and structuring contests. Traditionally, difficulty assignment is performed manually by problem setters which can be subjective, inconsistent, and time-consuming. Natural Language Processing (NLP) techniques provide an opportunity to analyze textual patterns as length of problem, certain keywords and extract meaningful information that can be used for prediction.

This project, **AutoJudge**, aims to develop an end-to-end machine learning system that predicts the difficulty of programming problems using only their textual descriptions. The system performs two tasks: classification of problems into Easy, Medium, or Hard categories, and regression to estimate a numerical difficulty score. By relying solely on textual information, the proposed approach remains independent of solution or human bias.

The project combines classical NLP techniques with machine learning models to build a practical solution. TF-IDF vectorization is used to convert textual problem descriptions into numerical features, followed by Logistic Regression for difficulty classification and a Random Forest Regressor for difficulty score prediction. In addition, a simple web-based interface is implemented using Streamlit to demonstrate real-time predictions.

Dataset Description

The dataset used in this project consists of programming problems provided by the problem statement. Each data sample represents a single programming problem along with its associated difficulty information. The dataset is provided in JSON format, where each line corresponds to one independent problem record.

Each problem entry contains multiple textual components that describe the problem in detail, as well as labels used for supervised learning.

Field Name	Description
title	Title of the programming problem
description	Main problem statement describing the task
input_description	Description of the input format
output_description	Description of the expected output
sample_io	Sample input and output pairs
problem_class	Difficulty class label (Easy / Medium / Hard)
problem_score	Numerical difficulty score
url	Source link of the problem

For model training and evaluation, the dataset is split into training and testing subsets using an 80–20 split. This ensures that the models are evaluated on unseen problems, providing a fair assessment of their generalization performance.

Data preprocessing

Data preprocessing is an essential step in preparing raw textual data for machine learning models. The dataset is first loaded from the JSONL format and converted into a structured Pandas DataFrame. Missing or null textual values are safely handled by replacing them with empty strings to ensure robustness during processing.

To capture complete contextual information, all the fields present in each problem statement is concatenated into a single unified text representation, which serves as the primary input for feature extraction.

The sample_io field is originally stored as a list of structured input–output examples. This information is converted into plain text by concatenating the sample inputs and outputs, allowing example-based details to contribute to the learning process. After preprocessing, each problem is represented by a single cleaned text document, and preparing the data for natural language processing techniques.

Feature Extraction

Feature extraction is performed using **Term Frequency–Inverse Document Frequency (TF-IDF)** vectorization to convert textual problem descriptions into numerical features. TF-IDF assigns higher importance to terms that occur frequently within a document but are less common across the dataset.

Both unigrams and bigrams are used to capture individual words as well as short phrases. English stopwords are removed to reduce noise and improve feature quality. To limit dimensionality and computational complexity, the number of TF-IDF features is capped at 3000.

The resulting TF-IDF feature vectors represent each programming problem as a numerical input suitable for machine learning models. TF-IDF is chosen due to its simplicity, interpretability, and effectiveness for text-based prediction tasks.

Models used

This project involves two supervised learning tasks: classification of problem difficulty and regression for predicting a numerical difficulty score. Separate models are used for each task.

- ### Classification model

For difficulty classification, Logistic Regression is used to predict one of three classes: Easy, Medium, or Hard. Logistic Regression is chosen due to its simplicity, interpretability, and strong performance on high-dimensional text features such as TF-IDF vectors. To address class imbalance in the dataset, the model is trained using balanced class weights.

- ### Regression model

For numerical difficulty estimation, a **Random Forest Regressor** is employed. Random Forest is an ensemble-based model that combines multiple decision trees to capture non-linear relationships in the data. It is robust to noise and performs well on complex feature

Experimental setup and Evaluation

The dataset is divided into training and testing sets using an 80–20 split, ensuring that model performance is evaluated on unseen data. All experiments are conducted on the same split to maintain consistency between classification and regression tasks.

- ## Classification evaluation

The performance of the classification model is evaluated using standard metrics including accuracy, precision, recall, and F1-score. These metrics provide a comprehensive assessment of model performance across different difficulty classes.

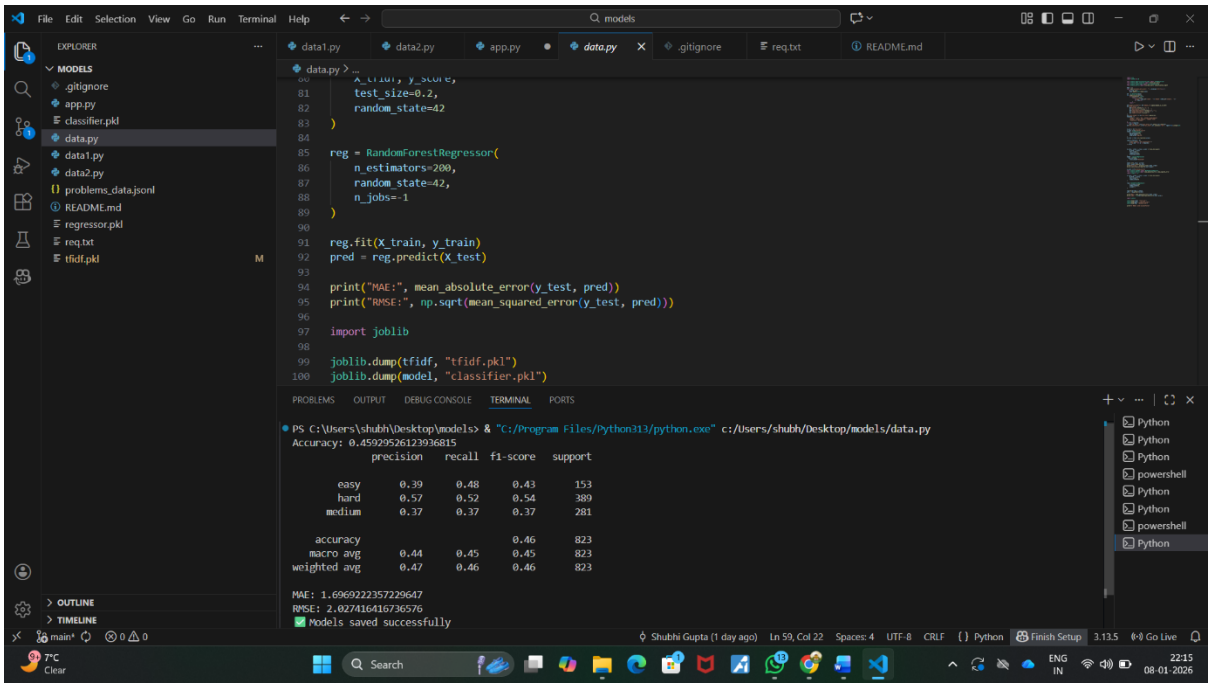
The Logistic Regression classifier achieves an overall accuracy of approximately **45–47%**. Given the challenging nature of the task and the reliance solely on textual information, this performance demonstrates the model's ability to capture meaningful difficulty-related patterns from problem descriptions. A confusion matrix is used to further analyze class-wise predictions and misclassifications.

- ## Regression evaluation

The regression model is evaluated using Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). These metrics quantify the difference between predicted difficulty scores and the ground truth values.

The Random Forest Regressor achieves an MAE of approximately **1.7–2.0**, indicating that the predicted difficulty scores are, on average, close to the true values. RMSE is also reported to capture larger prediction errors.

The evaluation results confirm that both models provide reasonable predictions given the limitations of text-only input.

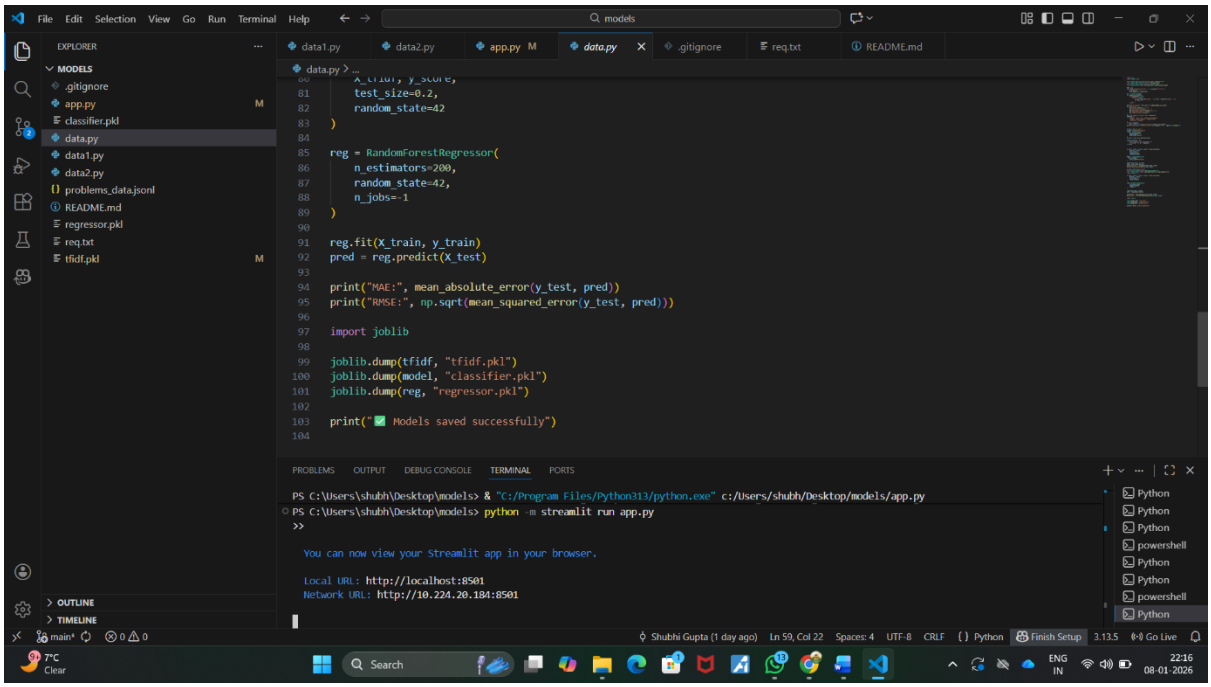


Web Interface and Sample Predictions

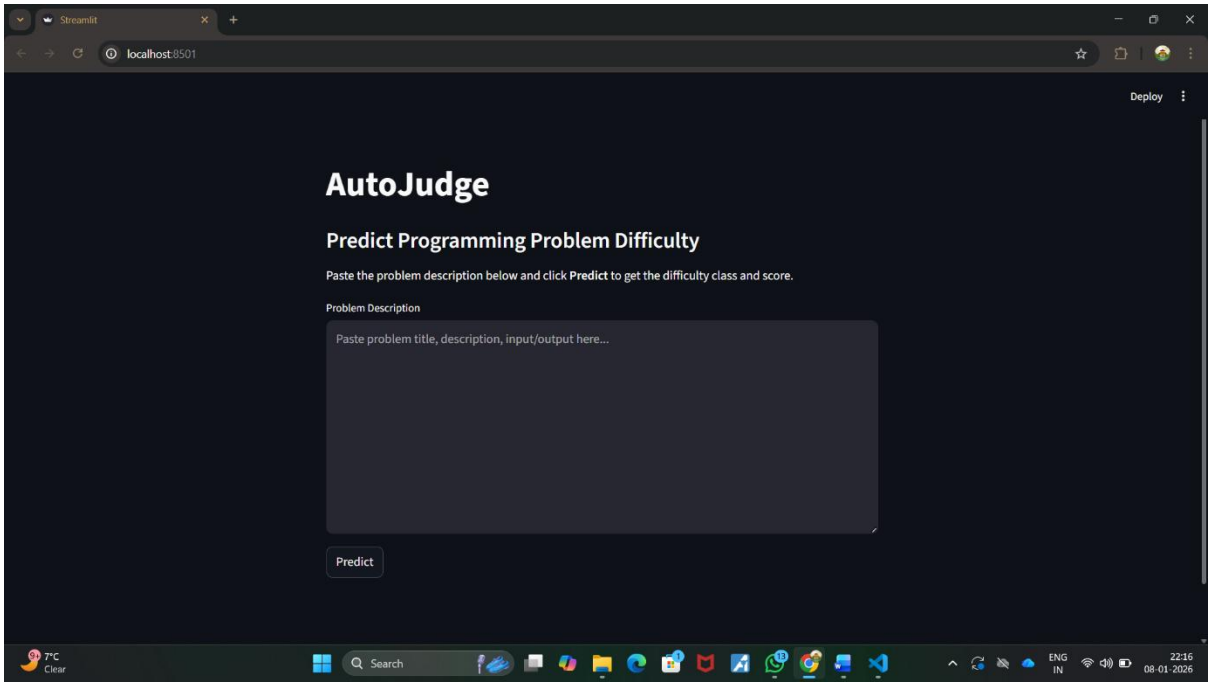
To demonstrate the practical usability of the proposed system, a simple web-based interface is implemented using Streamlit. The web interface provides an interactive way for users to test the trained models without directly interacting with the source code.

- ## Web interface description

The interface consists of a text input area where users can paste the complete description of a programming problem, similar to those found on competitive programming platforms. After entering the problem text, the user can click the Predict Difficulty button to initiate the prediction process.



Internally, the input text is processed using the same TF-IDF vectorizer that was used during training to ensure consistency. The transformed features are then passed to the pre-trained classification and regression models. The classification model predicts the difficulty category (Easy, Medium, or Hard), while the regression model outputs a numerical difficulty score. No model retraining occurs during runtime, ensuring fast and stable predictions.



- Sample predictions

To evaluate the behaviour of the system, multiple programming problem descriptions of varying complexity are tested through the web interface. For simple problems involving basic arithmetic or iteration, the model typically predicts an **Easy** difficulty with a low numerical score. Problems involving standard algorithms such as dynamic programming or graph traversal are generally predicted as **Medium**, while problems describing complex graph algorithms or large constraints are often classified as **Hard** with higher difficulty scores.

Screenshots of the web interface along with sample predictions for Easy, Medium, and Hard problems are included in this section to illustrate the working of the system.

1. Easy problem
2. Medium problem

You are given an array of n integers. You need to find the length of the longest increasing subsequence.

Input:

The first line contains an integer n.

The second line contains n integers.

Output:

Print the length of the longest increasing subsequence.

Constraints:

$1 \leq n \leq 10^5$

1 ≤ ai ≤ 10^9

Example:

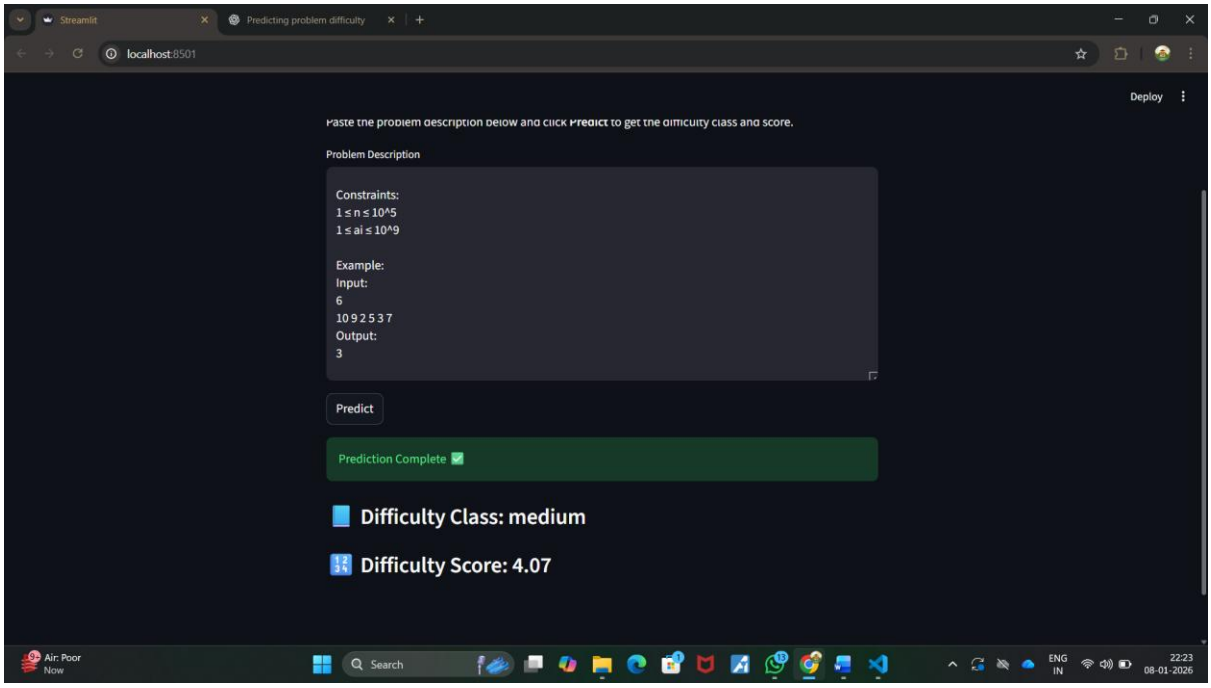
Input:

6

10 9 2 5 3 7

Output:

3



3. Hard problem

You are given a connected graph with n nodes and m edges. Each edge has a weight.

You need to find the minimum cost to travel from node 1 to node n.

Input:

The first line contains two integers n and m.

The next m lines contain three integers u, v, w describing an edge.

Output:

Print the minimum cost path from node 1 to node n.

Constraints:

1 ≤ n ≤ 2 × 10^5

1 ≤ m ≤ 2 × 10^5

$1 \leq w \leq 10^9$

Example:

Input:

5 6

1 2 4

1 3 2

3 2 1

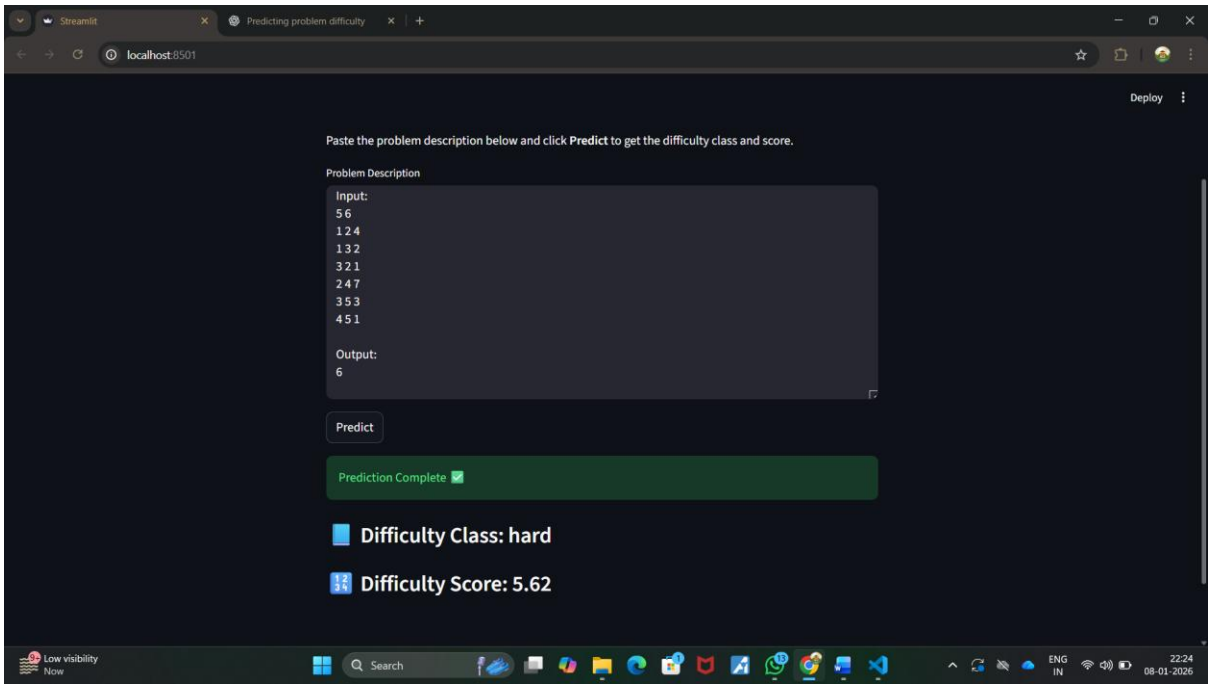
2 4 7

3 5 3

4 5 1

Output:

6



Conclusion

This project presented AutoJudge, an end-to-end machine learning system designed to predict the difficulty of programming problems using only their textual descriptions. By combining classical natural language processing techniques with supervised machine learning models,

the system successfully performs both difficulty classification and numerical difficulty score prediction. The use of TF-IDF for feature extraction, Logistic Regression for classification, and Random Forest for regression enables an interpretable and efficient solution without relying on deep learning models.

Experimental results demonstrate that meaningful difficulty-related patterns can be extracted from problem statements alone. Although the classification accuracy and regression error are moderate, the results are reasonable given the complexity of the task and the absence of solution code or runtime information. The implemented Streamlit web interface further highlights the practical applicability of the system by allowing real-time predictions on unseen problem descriptions.

Although the classification accuracy is not perfect, this is expected due to the subjective nature of difficulty labelling and the reliance solely on textual information. Programming problem difficulty is ambiguous, and textual descriptions often overlap across difficulty levels. Moreover, achieving extremely high accuracy in such tasks may indicate overfitting. The observed accuracy demonstrates that the model learns meaningful difficulty-related patterns while maintaining generalization to unseen problems.