Name- Swara Vikram Chalikwar Enrolment No.-24115153

REPORT- AutoJudge System

1. Introduction

Competitive programming platforms such as Codeforces host thousands of programming problems across a wide range of difficulty levels. Assigning appropriate difficulty ratings to these problems is essential for contest organization, user engagement, and learning progression. Traditionally, difficulty labelling is performed manually by problem setters and administrators, which can be subjective, time-consuming, and inconsistent as the number of problems grows.

With the increasing availability of historical problem data, machine learning techniques can be used to automatically predict the difficulty of programming problems based on their characteristics. Automating this process enables consistent and data-driven difficulty estimation and reduces dependency on manual judgment.

This project presents **AutoJudge**, a machine learning–based system that predicts the difficulty of programming problems using problem metadata. The system performs both classification and regression tasks and provides an interactive web interface for real-time predictions.

1.1 Problem Statement

The objective of this project is to design and implement an automated system capable of predicting the difficulty of programming problems. Given problem-related metadata such as problem tags and the number of users who solved the problem, the system aims to:

1. **Classify** programming problems into three difficulty categories: *Easy, Medium, and Hard*.
2. **Predict** a numerical difficulty score (problem rating) representing the relative complexity of the problem.

The proposed AutoJudge system integrates data preprocessing, feature extraction, machine learning models, and a web-based interface to deliver an end-to-end solution for difficulty prediction.

2. Dataset Description

2.1 Dataset Source

The dataset used in this project was obtained from Kaggle and is titled *'6000+ Codeforces Problems Rating Dataset'*. It contains information for over 6,000 programming problems sourced from the Codeforces platform and includes problem tags, solve counts, and difficulty ratings.

2.2 Dataset Structure

The dataset is stored in CSV format and includes the following key attributes:

Codeforces_Dataset

id	Problem_tag_1	Problem_tag_2	Problem_tag_3	Problem_tag_4	Solved	Problem Rating
2123G	brute force	data structures	greedy	math	x1046	
2123F	constructive algorithms	number theory			x4419	
2123E	binary search	data structures	greedy	sortings	x7633	
2123D	constructive algorithms	games	greedy		x12348	
2123C	brute force	data structures			x19142	
2123B	greedy				x22312	
2123A	math				x26951	
2121H	binary search	brute force	data structures	dp	x1076	2300
2121G	data structures	divide and conquer	math	sortings	x2731	1900
2121F	binary search	brute force	data structures	greedy	x4586	1800
2121E	dp	greedy	implementation	strings	x8888	1500
2121D	implementation	sortings			x12368	1300
2121C	greedy	greedy	implementation		x14964	1200
2121B	constructive algorithms	greedy	strings		x25305	800
2121A	brute force	math			x29787	800
2120G	graphs	greedy	math		x53	3000
2120F	2-sat	graphs			x174	2600
2120E	binary search	dp	ternary search		x1067	2300
2120D	combinatorics	math			x3707	1800
2120C	constructive algorithms	greedy	math	sortings	x8084	1400
2120B	geometry				x12738	1000
2120A	geometry	math			x15076	800

Column name	Description
Id	Unique problem identifier
Problem_tag_1	Primary problem category
Problem_tag_2	Secondary problem category
Problem_tag_3, Problem_tag_4	Additional problem categories
Solved	Number of users who solved the problem
Problem rating	Official difficulty rating

The dataset contains problem metadata that is suitable for both text-based and numerical feature extraction.

3. Data Preprocessing

Data preprocessing is a crucial step to ensure that the dataset is clean and suitable for machine learning models. The following preprocessing steps were performed:

- Rows containing missing or invalid values in critical columns were removed.
- The **Solved** column, which contained values in the format **x1234,** was converted into a numerical format.
- Multiple problem tags were combined into a single textual feature to represent the problem category.
- The target variables for classification and regression were separated.
- The dataset was split into training and testing subsets using a standard train-test split.

These steps ensured data consistency and improved model reliability.

4. Feature Engineering

Feature engineering was performed to convert raw data into meaningful numerical representations suitable for machine learning algorithms.

4.1 Text Feature Extraction

Problem tags are provided in the dataset across multiple columns (Problem_tag_1 to Problem_tag_4). Since machine learning models require a single text representation per sample, all available problem tags for a given problem were first **concatenated into a single string**. After concatenation, **Term Frequency–Inverse Document Frequency (TF-IDF)** vectorization was applied to the combined tag string. TF-IDF converts the textual data into numerical feature vectors by assigning higher weights to tags that are more informative and less frequent across the dataset, while reducing the impact of very common tags. The resulting numerical features are then used for training the classification and regression models.

4.2 Numerical Features

The number of users who solved the problem is also used as a numerical feature. This feature has a strong relationship with problem difficulty. The final feature set consists of TF-IDF vectors combined with numerical features.

5. Models Used

The project involved experimentation with multiple machine learning models to determine the most effective approach.

5.1 Classification and Regression Models

Logistic Regression and Linear Regression were initially used as **baseline models** for the classification and regression tasks respectively, due to their simplicity and ease of interpretation. While these baseline models achieved reasonable classification performance, the regression model resulted in comparatively higher error values, as measured by Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

To improve performance, I explored **Random Forest models** for both classification and regression. The Random Forest Regressor led to a noticeable reduction in MAE and RMSE, indicating improved regression accuracy. However, when Random Forest was used for classification, a slight decrease in classification accuracy was observed compared to Logistic Regression.

Based on these observations, a **hybrid modelling approach** was adopted. **Logistic Regression** was retained as the final classification model due to its higher accuracy (approximately **87.5%**), while **Random Forest Regressor** was selected as the final regression model because of its superior error performance. This combination achieved a balance between classification accuracy and regression precision and was therefore chosen as the final model configuration for the AutoJudge system.

6. Experimental Setup and Results

6.1 Experimental Setup

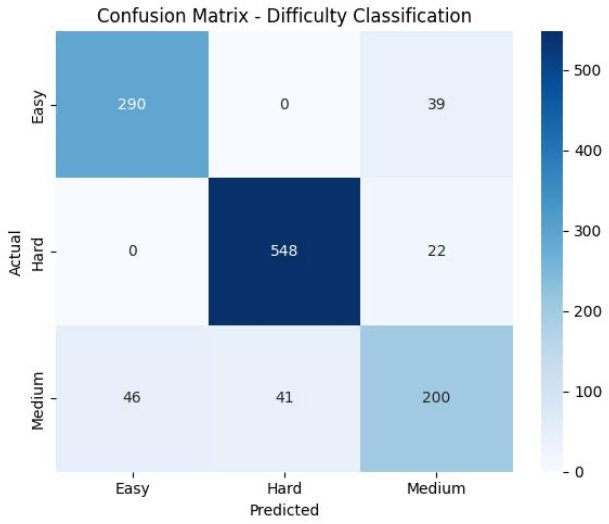
- The dataset is divided into training and testing sets.
- Classification performance is evaluated using **accuracy** and **confusion matrix**.
- Regression performance is evaluated using **Mean Absolute Error (MAE)** and **Root Mean Squared Error (RMSE)**.
- RMSE is computed as the square root of Mean Squared Error (RMSE = √MSE).

6.2 Results

Task	Model	Metric	Value
Classification	Logistic Regression	Accuracy	0.8752
Regression	Random Forest	MAE	167.94
Regression	Random Forest	RMSE	228.15

A confusion matrix was generated to visualize the classification performance across Easy, Medium, and Hard categories. The matrix indicates strong diagonal dominance, showing that most predictions were correct.

The confusion matrix I obtained for the hybrid model used:



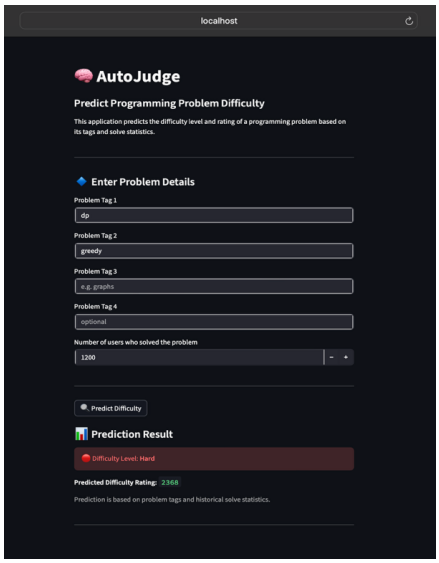
7. Web Interface

A web-based interface was developed using **Streamlit** to allow interactive difficulty prediction.

7.1 Interface Features

- Input fields for problem tags
- Input for the number of users who solved the problem
- Display of predicted difficulty level and difficulty rating

7.2 Sample Prediction



The interface demonstrates real-time inference using trained machine learning models.

8. Conclusion and Future Work

This project successfully demonstrates the application of machine learning techniques to predict programming problem difficulty. The AutoJudge system integrates preprocessing, feature extraction, classification, regression, and a web interface into a complete pipeline.

The results indicate that combining textual features with numerical metadata leads to reliable difficulty predictions. The system can assist competitive programming platforms, educators, and learners in understanding problem complexity.