Name- Swara Vikram Chalikwar Enrollment 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 the textual description of problem statements. 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 the textual description of a programming 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 provided as part of the project guidelines. It consists of programming problems along with their titles, textual descriptions, and associated difficulty scores.

2.2 Dataset Structure

The dataset was provided as part of the project guidelines. Although the raw dataset is stored in JSON Lines format, its structure consists of problem titles, descriptions, input and output descriptions, sample i_o, problem_class, problem_score and URL.

In this project, only the problem title and main description were used as input features, as these fields capture the core semantic and conceptual complexity of programming problems. Other textual fields such as input/output descriptions and sample I/O were not included, as they are often highly structured and repetitive and contribute less to difficulty estimation.

3. Data Preprocessing

Data preprocessing involved handling missing textual values by replacing them with empty strings. The problem title and description were combined into a single text field to form a unified textual representation. This prepared the data for text-based feature extraction while ensuring robustness to incomplete inputs.

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

Feature engineering was performed by converting the combined problem text into numerical representations using TF-IDF (Term Frequency–Inverse Document Frequency). TF-IDF captures the importance of words in problem statements by assigning higher weights to informative terms.

4.2 Keyword Frequency Features

In addition to TF-IDF, keyword frequency features were extracted by counting the occurrences of algorithm-related keywords such as **graph, dp, recursion, and greedy** within the problem text. These features explicitly capture algorithmic cues that correlate with problem difficulty.

4.3 Final Feature Set

The final feature vector consists of TF-IDF features concatenated with keyword frequency features.

5. Models Used

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

5.1 Baseline Models

Logistic Regression was used as a baseline model for difficulty classification, while Linear Regression was initially applied for difficulty score prediction. These models provided a reference point for evaluating more complex approaches. However, Linear Regression showed unstable performance for regression due to the high dimensionality of text-based features.

Its Evaluation metrics:

```
swarachalikwar@swaras-MacBook-Air AutoJudge_system % python3 train.py

Dataset loaded successfully
                     title  ...                                          url
0                      Uuu  ...          https://open.kattis.com/problems/uuu
1            House Building  ...     https://open.kattis.com/problems/husbygge
2            Mario or Luigi  ...  https://open.kattis.com/problems/marioorluigi
3            The Wire Ghost  ...  https://open.kattis.com/problems/thewireghost
4  Barking Up The Wrong Tree  ...     https://open.kattis.com/problems/barktree

[5 rows x 8 columns]
Classification Accuracy: 0.4678
Regression MAE: 4232.50
Regression RMSE: 5808.37
Models saved successfully in /models
```

5.2 Model Exploration and Comparison

To identify the most suitable models for both classification and regression tasks, multiple approaches were explored in a systematic manner.

Initially, Random Forest models were evaluated for both difficulty classification and difficulty score prediction due to their ability to handle non-linear relationships and high-dimensional feature spaces generated from text data. Random Forest demonstrated stable and competitive performance in both tasks.

Random Forest Model Metrics:

```
swarachalikwar@swaras-MacBook-Air AutoJudge_system % python3 train.py
Dataset loaded successfully
                    title  ...                                        url
0                     Uuu  ...          https://open.kattis.com/problems/uuu
1            House Building  ...      https://open.kattis.com/problems/husbygge
2            Mario or Luigi  ...  https://open.kattis.com/problems/marioorluigi
3            The Wire Ghost  ...  https://open.kattis.com/problems/thewireghost
4  Barking Up The Wrong Tree  ...      https://open.kattis.com/problems/barktree

[5 rows x 8 columns]
Classification Accuracy: 0.5115
Regression MAE: 1.73
Regression RMSE: 2.07
Models saved successfully in /models
swarachalikwar@swaras-MacBook-Air AutoJudge_system %
```

To further improve results, **Gradient Boosting** was evaluated for the regression task in combination with Random Forest classifier for classification. However, Gradient Boosting did not provide any significant improvement in MAE and RMSE compared to Random Forest. The following are the results:

```
Classification Accuracy: 0.5115
Regression MAE: 1.78
Regression RMSE: 2.11
Models saved successfully in /models
```

Subsequently, **Support Vector Machines (SVM)** were explored for classification while retaining Random Forest for regression. Despite tuning the SVM model, its classification performance was lower than that of Random Forest on the given dataset. The results:

```
Classification Accuracy: 0.4131
Regression MAE: 1.73
Regression RMSE: 2.07
Models saved successfully in /models
swarachalikwar@swaras-MacBook-Air AutoJudge_system %
```

5.3 Final Model Selection

Based on experimental evaluation and comparative analysis, Random Forest Classifier and Random Forest Regressor consistently achieved better performance than alternative models. Random Forest provided higher classification accuracy and more stable regression results while maintaining robustness to noise in text-based features.

6. Experimental Setup and Results

6.1 Experimental Setup

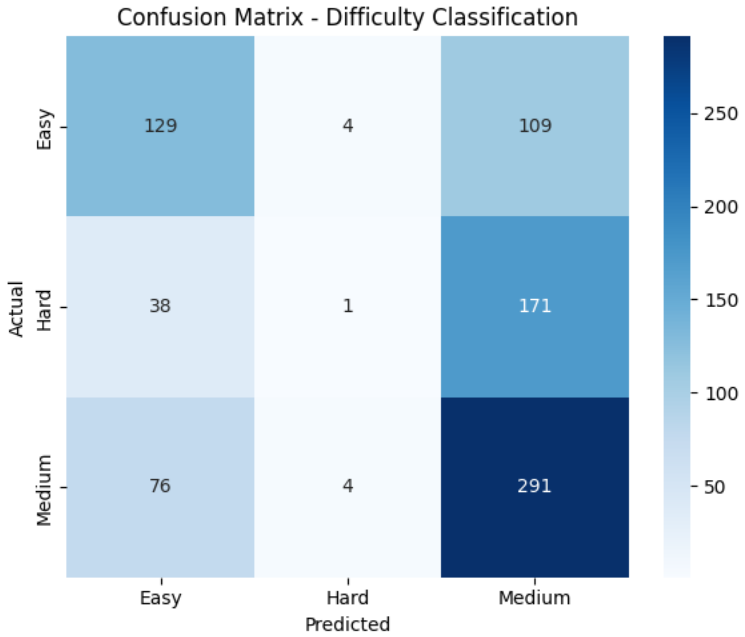
- The dataset is divided into training and testing sets.
- Classification performance is evaluated using **accuracy** and **confusion matrix**.
- Regression performance is evaluated using **Mean Absolute Error (MAE)** and **Root Mean Squared Error (RMSE)**.
- RMSE is computed as the square root of Mean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

6.2 Results

Task	Model	Metric	Value
Classification	Random Forest	Accuracy	0.5115
Regression	Random Forest	MAE	1.73
Regression	Random Forest	RMSE	2.07

A confusion matrix was generated to visualize the classification performance across Easy, Medium, and Hard categories. The matrix indicates a clear diagonal pattern, showing that a majority of predictions align with the true difficulty class.

The confusion matrix obtained for the Random Forest model is:



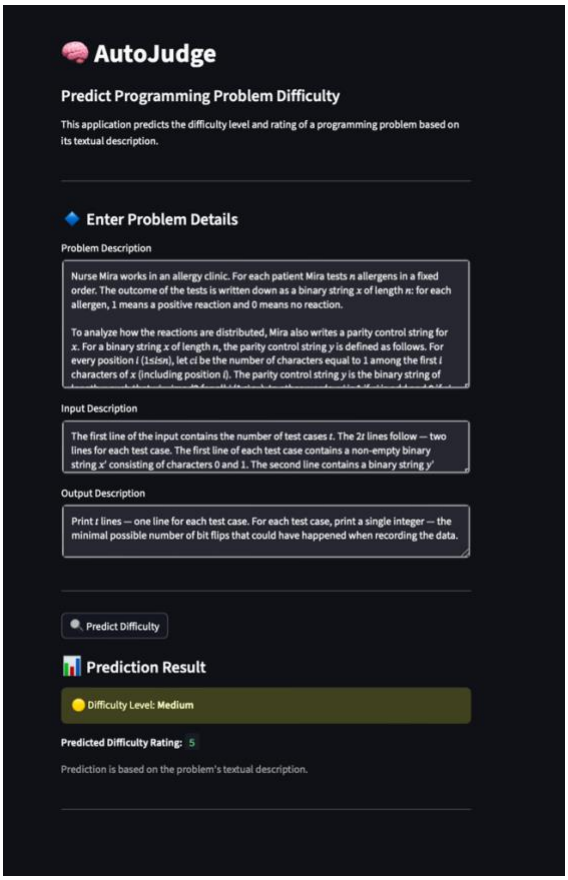
7. Web Interface

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

7.1 Interface Features

- Text area for entering the full problem description
- Display of predicted difficulty class and difficulty score

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 leveraging textual features extracted from problem descriptions leads to reliable difficulty predictions. The system can assist competitive programming platforms, educators, and learners in understanding problem complexity.