AutoJudge: An Automated System for Predicting Programming Problem Difficulty

A Machine Learning and Natural Language Processing Based Approach

SUBMITTED BY:
BHOOMIKA CHOURASIYA (24114023)
2ND YEAR, B.TECH, COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, ROORKEE

---

1. Introduction

Competitive programming platforms such as Codeforces, CodeChef, and Kattis categorize problems into difficulty levels like *Easy*, *Medium*, and *Hard*, often accompanied by a numerical difficulty rating. These labels are typically assigned manually based on expert judgment, problem-solving statistics, and user feedback, making the process subjective and inconsistent.

The objective of this project, **AutoJudge**, is to design an intelligent system that automatically predicts the difficulty of programming problems using only their textual descriptions. The system formulates this as two machine learning tasks:

- A **classification task** to predict the difficulty class *(Easy / Medium / Hard)*
- A **regression task** to predict a **numerical difficulty score**

The project applies Natural Language Processing (NLP) and machine learning techniques to analyze problem statements and delivers predictions through a web-based interface.

---

2. Dataset Description

The dataset used in this project is derived from the **TaskComplexityEval-24** dataset, which contains programming problems collected from competitive programming platforms such as Kattis.

Each sample includes:

- Problem title
- Problem description
- Input description
- Output description
- Difficulty class *(Easy / Medium / Hard)*

- Numerical difficulty score

The original dataset was provided in **JSONL format** and was converted into **CSV format** for easier preprocessing and analysis.

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

The dataset contains **4,112 problems**, providing a realistic and diverse set of programming challenges.

---

3. Data Preprocessing

Text preprocessing was a crucial step in preparing the dataset for machine learning.

The following steps were applied:

3.1 Combining Text Fields

The following fields were concatenated into a single textual feature:

- Title
- Problem description
- Input description
- Output description

This formed a unified **full_text** representation of each problem.

3.2 Handling Missing Values

Missing text fields were replaced with empty strings to avoid information loss.

3.3 Text Cleaning

The combined text was normalized by:

- Converting all characters to lowercase
- Removing extra whitespace
- Applying basic text normalization

An additional numeric feature **text_length** was also computed to represent the size of each problem statement.

---

4. Feature Engineering

To convert textual data into numerical form, **TF-IDF (Term Frequency–Inverse Document Frequency)** vectorization was applied.

Key settings:

- **n-grams:** 1 to 3
- **Maximum features:** 20,000
- **Minimum document frequency:** 3
- **Maximum document frequency:** 90%
- **Stop words:** English
- **Sublinear term frequency scaling**

TF-IDF helps capture important words related to algorithms, constraints, and problem structure, which are highly indicative of difficulty.

---

5. Classification Models

5.1 Baseline Classification Model

The baseline model used was **Logistic Regression** trained on TF-IDF features.

Baseline Results:

- Accuracy ≈ **0.49**
- Hard class F1-score ≈ **0.59**

This model provided a strong baseline for text-based difficulty prediction.

5.2 Improved Classification Model

A **Linear Support Vector Machine (SVM)** was used as an improved model.

SVM is well suited for high-dimensional sparse text features and produced better class separation.

Final Classification Results (SVM):

Class	Precision	Recall	F1-score
Easy	0.41	0.40	0.41
Medium	0.39	0.32	0.35
Hard	0.57	0.66	0.61
Overall Accuracy			**0.49**

The SVM model showed **stronger performance on the "Hard" class**, which is the most important and dominant category.
Therefore, **Linear SVM** was chosen as the final classification model.

6. Regression Models

6.1 Ridge Regression

Ridge Regression was used as a baseline regressor due to its suitability for sparse TF-IDF features.

Results:

- **MAE = 1.64**
- **RMSE = 1.97**

This means the predicted difficulty score is typically within about **±1.6 points** of the true value.

6.2 Random Forest Regression

Random Forest was tested to capture non-linear relationships.

Results:

- **MAE = 1.68**
- **RMSE = 2.03**

6.3 Gradient Boosting Regression

Results:

- **MAE = 1.68**
- **RMSE = 2.02**

Final Regression Model

Ridge Regression performed best and was selected as the final regression model.
Tree-based models underperformed due to the **high-dimensional sparse nature** of TF-IDF features.

---

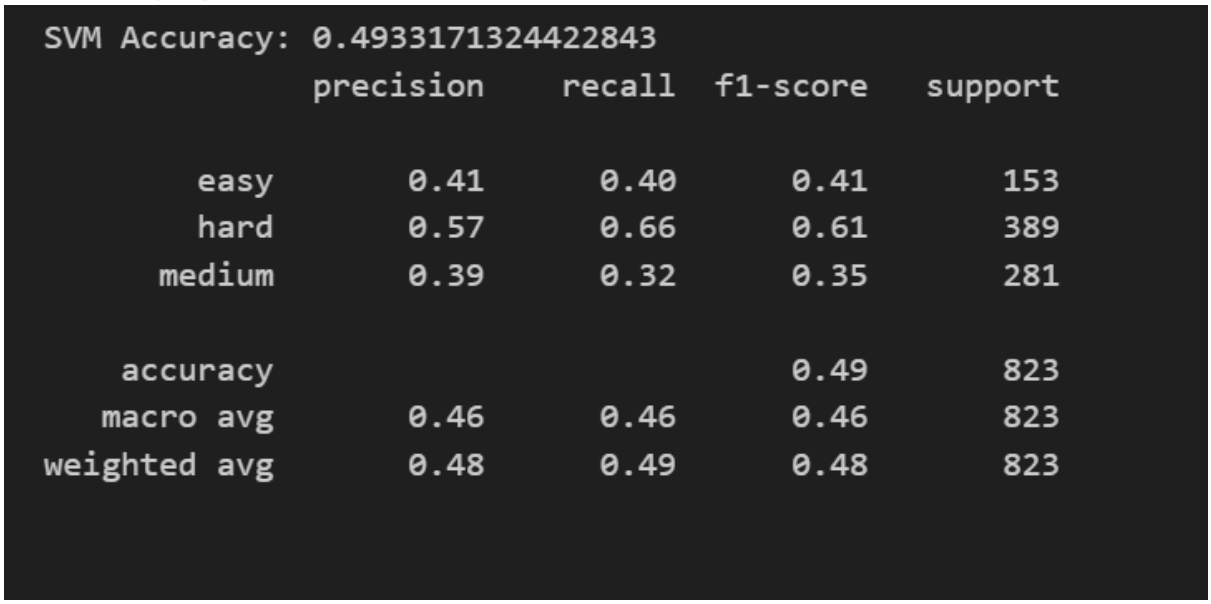
7. Experimental Setup and Results

The dataset was split using an **80:20 train-test split** with stratification on the difficulty class.

Summary of Results

Task	Model	Metric	Result
Classification	Logistic Regression	Accuracy	~0.49
Classification	Linear SVM	Accuracy	~0.49
Regression	Ridge Regression	MAE	~1.64
Regression	Random Forest	MAE	~1.68
Regression	Gradient Boosting	MAE	~1.68

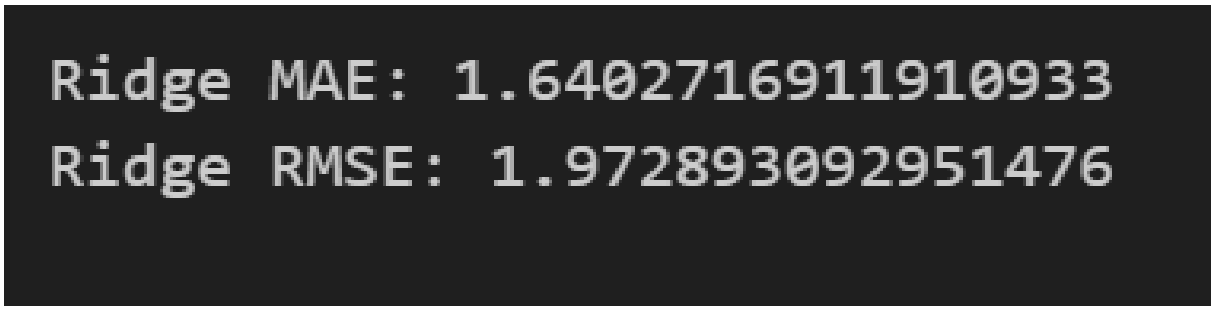
These results are realistic and meaningful given the **subjective nature of difficulty labels** and the **text-only input**.



```
SVM Accuracy: 0.4933171324422843
              precision    recall  f1-score   support

        easy       0.41      0.40      0.41       153
        hard       0.57      0.66      0.61       389
      medium       0.39      0.32      0.35       281

    accuracy                           0.49       823
   macro avg       0.46      0.46      0.46       823
weighted avg       0.48      0.49      0.48       823
```



```
Ridge MAE: 1.6402716911910933
Ridge RMSE: 1.972893092951476
```

---

8. Web Interface

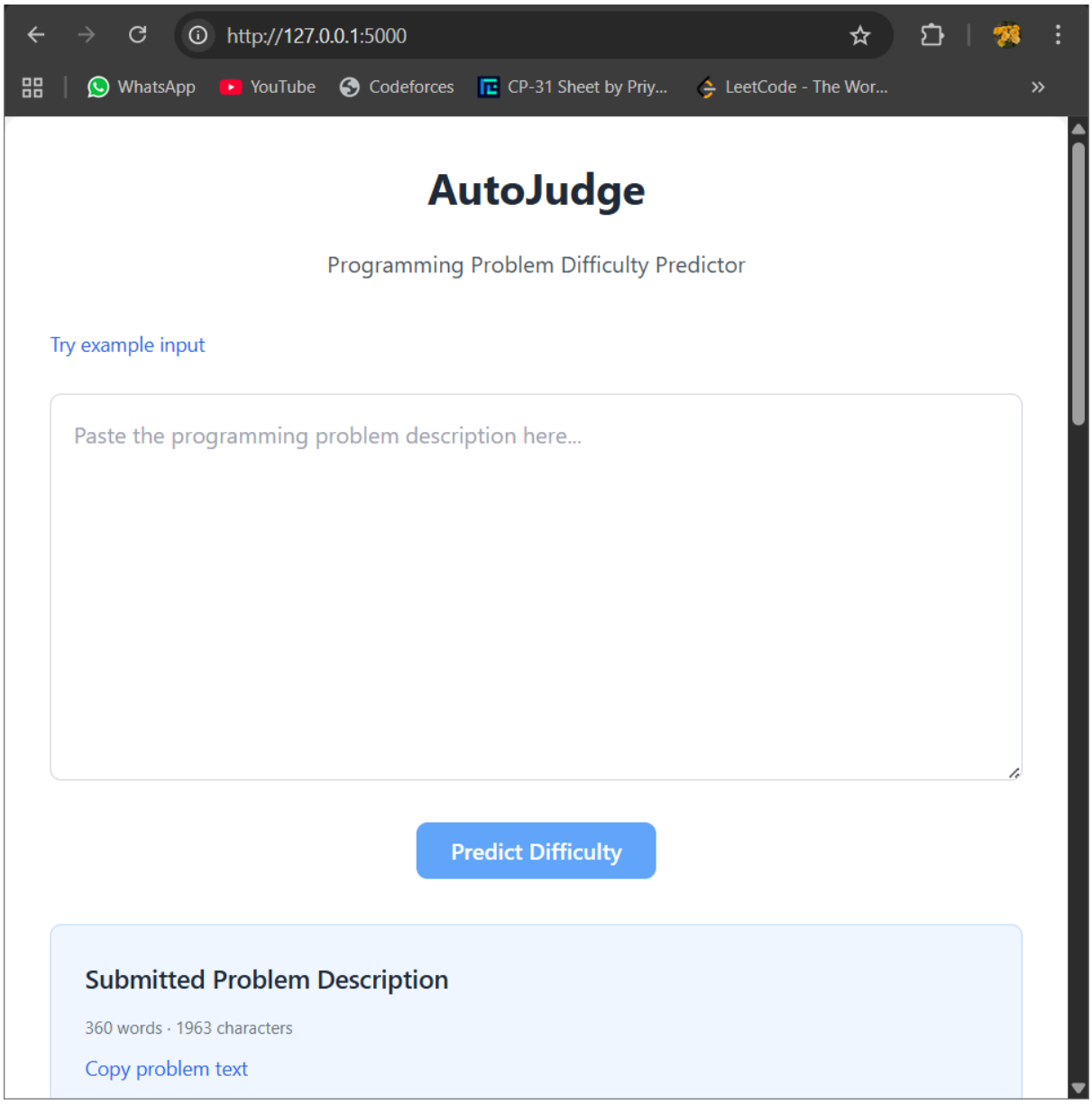
A **Flask-based web interface** was developed to make AutoJudge interactive.

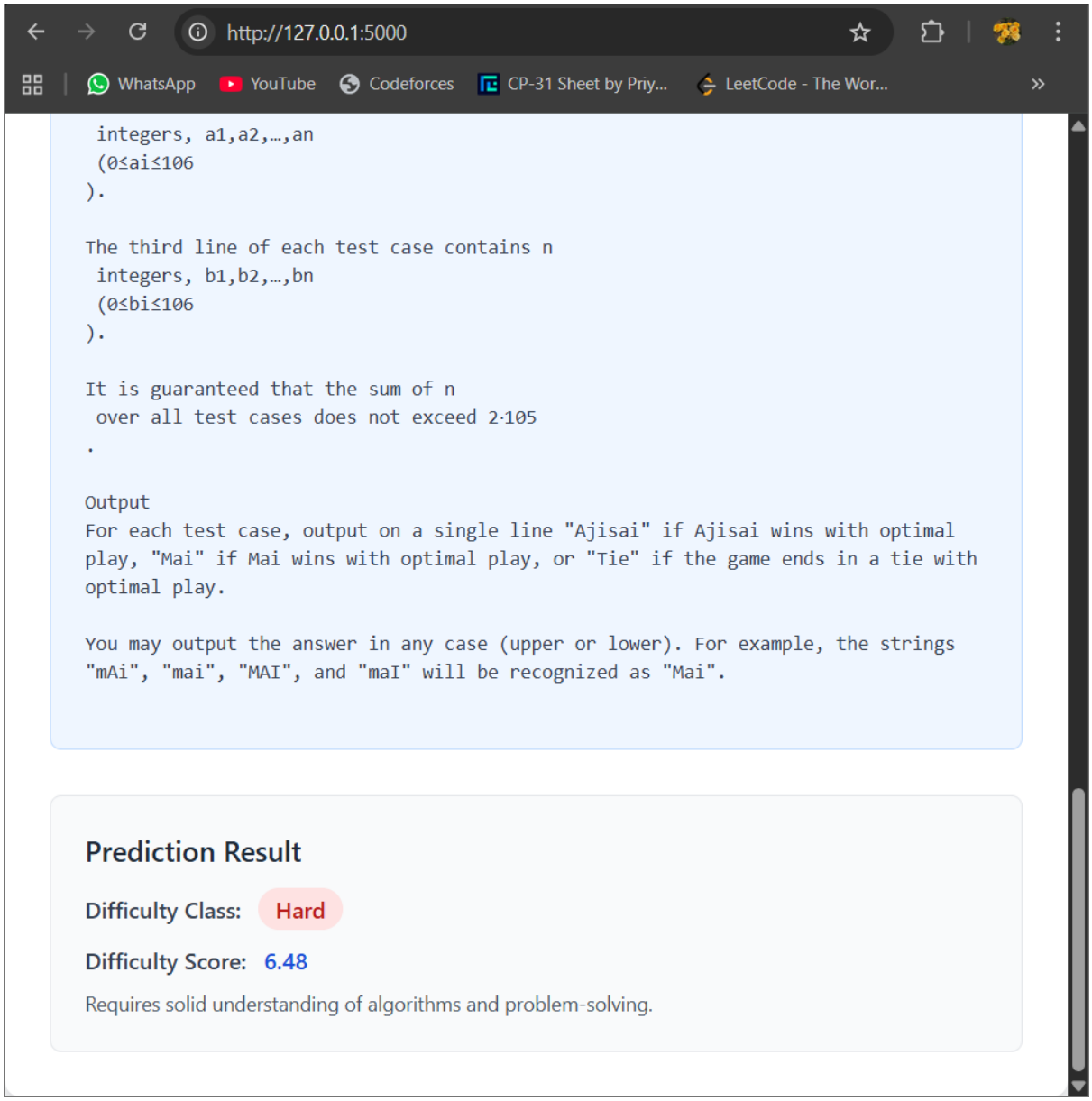
Users can:

- Paste a programming problem

- Click **Predict**
- View:
    - Predicted difficulty class
    - Predicted difficulty score

The interface uses **Tailwind CSS** for styling and provides a clean, modern experience.





---

9. Conclusion

This project demonstrates that machine learning models can extract useful patterns from programming problem text to estimate difficulty. Despite the subjective and noisy nature of difficulty labels, AutoJudge provides a reliable automated baseline for difficulty prediction.

The system integrates data preprocessing, feature engineering, classification, regression, and a web UI into a complete end-to-end machine learning pipeline.

---

10. Future Work

Possible future improvements include:

- Using transformer embeddings (e.g., BERT)
- Dataset balancing
- Incorporating problem constraints and tags
- Advanced hyperparameter tuning
- Multi-language problem support
- Public cloud deployment