REPORT - AutoJudge

Mishka Singla
24116052
ECE

1. Abstract

Competitive programming platforms classify problems by difficulty and assign numerical difficulty scores, typically based on human judgment and user feedback. This process is subjective and requires significant manual effort.

This project presents **AutoJudge**, a machine learning system that automatically estimates the difficulty of programming problems using only textual information. The system performs two tasks: predicting a categorical difficulty class (Easy, Medium, Hard) and predicting a numerical difficulty score. Textual features are extracted using TF-IDF representations along with additional structural features capturing problem length, algorithmic keyword frequency, and mathematical symbol density. Multiple classification and regression models are evaluated. A Linear Support Vector Machine is selected for classification, while a Random Forest Regressor is used for difficulty score prediction. A Streamlit-based web interface demonstrates real-time predictions. Experimental results show realistic performance given the subjective nature of difficulty estimation and the constraint of using text-only inputs.

2. Introduction

Competitive programming platforms such as Codeforces and CodeChef rely on difficulty ratings to guide users in selecting appropriate problems. These ratings influence learning progression, contest design, and problem recommendation systems. However, difficulty assignment is inherently subjective and typically depends on user feedback collected over time.

Automating difficulty estimation is challenging because difficulty depends on multiple factors such as algorithmic complexity, constraints, and intended solution strategies. In this project, we restrict ourselves to **textual information only**, making the task even more challenging but more generalizable. The objective is to explore how far natural language processing and machine learning techniques can go in estimating problem difficulty from problem statements alone.

3. Problem Statement

The objective of this project is to design a system that, given the textual description of a programming problem, can:

1. Predict a **difficulty class**: Easy, Medium, or Hard (classification task).
2. Predict a **numerical difficulty score** (regression task).

Constraints:

- Only textual data (problem description, input description, output description) is used.

- No access to constraints, editorial information, tags, or user statistics.

4. Dataset Description

The dataset consists of competitive programming problems annotated with difficulty labels and scores. Each sample contains the following fields:

Field	Description
title	Problem title
description	Problem statement
input_description	Input format
output_description	Output format
problem_class	Difficulty label (Easy / Medium / Hard)
problem_score	Numerical difficulty score

The dataset exhibits **class imbalance**, with Hard problems being more frequent than Easy problems. This imbalance significantly affects classification performance and is addressed during model selection.

5. Data Preprocessing

All textual fields are combined into a single text input:

title + description + input_description + output_description

Preprocessing steps include:

- Converting text to lowercase
- Removing extra whitespace
- Retaining programming-specific terms (no stemming or lemmatization)

Aggressive preprocessing was avoided to preserve important technical vocabulary such as algorithm names and mathematical expressions.

6. Feature Engineering

Feature engineering plays a crucial role in capturing both semantic and structural aspects of problem difficulty.

6.1 TF-IDF Features

TF-IDF (Term Frequency–Inverse Document Frequency) vectors are used to represent the combined problem text. Unigrams and bigrams are included to capture relevant programming terminology and contextual phrases. The resulting feature space is high-dimensional and sparse.

6.2 Structural Features

To complement TF-IDF features, additional handcrafted features are introduced:

Feature	Motivation
Log-scaled text length	Longer problems often involve more complex logic
Algorithmic keyword frequency	Indicates presence of advanced techniques
Mathematical symbol density	Reflects mathematical complexity

These features inject domain knowledge into the model and improve predictive stability.

7. Models and Experimental Setup

The problem is approached using **separate models** for classification and regression.

7.1 Classification Models

The following models were evaluated:

- **Logistic Regression**
  Collapsed to predicting the majority class due to class imbalance.
- **Multinomial Naive Bayes**
  Performed poorly due to strong feature independence assumptions and biased predictions.
- **Linear Support Vector Machine (Final Model)**
  Selected for its robustness in high-dimensional sparse spaces and support for class balancing.

The final classifier uses:

- TF-IDF + structural features
- class_weight = "balanced"

7.2 Regression Models

The following regression models were evaluated:

- Linear Regression (baseline)
- Random Forest Regressor (final model)
- Gradient Boosting Regressor

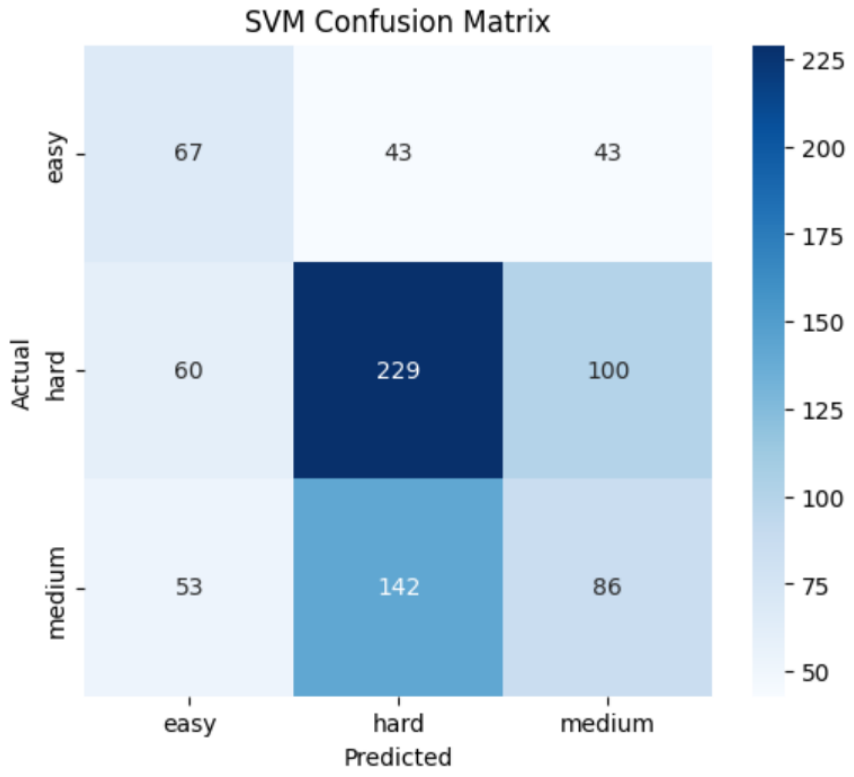
The **Random Forest Regressor** was chosen due to its robustness, non-linear modeling capability, and stable performance.

8. Results and Evaluation

8.1 Classification Results

- Accuracy ≈ **46–47%**
- Macro-averaged F1-score ≈ **0.44**

	precision	recall	f1-score	support
easy	0.37	0.44	0.40	153
hard	0.55	0.59	0.57	389
medium	0.38	0.31	0.34	281
accuracy			0.46	823
macro avg	0.43	0.44	0.44	823
weighted avg	0.46	0.46	0.46	823



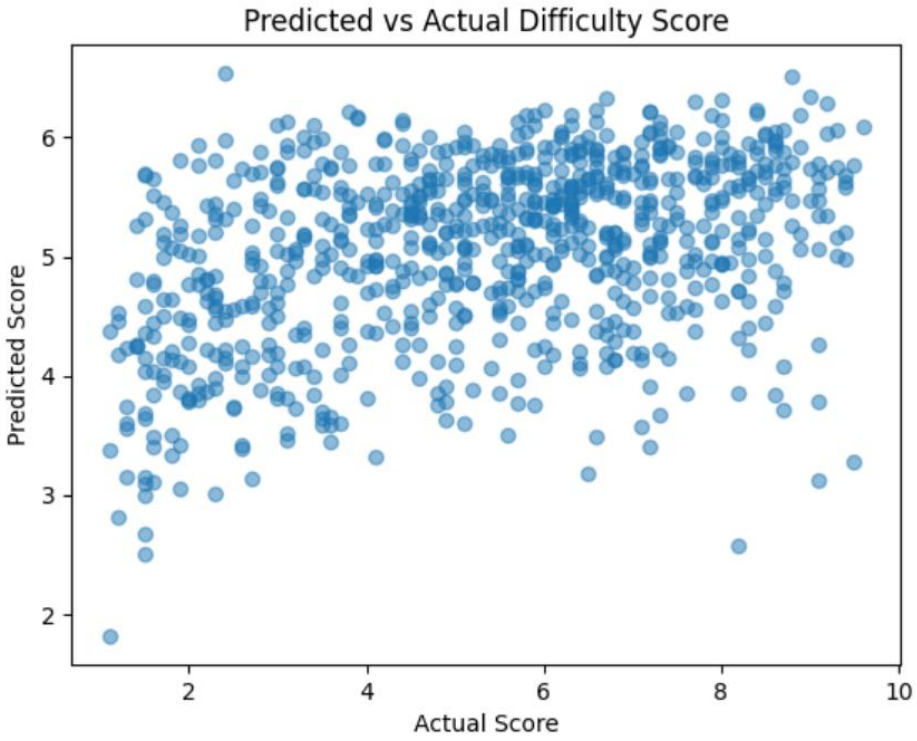
The moderate accuracy reflects the subjective nature of difficulty labels and overlapping vocabulary between classes. The model significantly outperforms random guessing (≈33%) while maintaining balanced performance across classes.

8.2 Regression Results

Metric

```
MAE: 1.6963844968489958
RMSE: 2.039589898011407
```

The difficulty score range is approximately 1–10, making the relative error around 20%. Given the subjectivity of difficulty estimation and the use of text-only features, these results are considered reasonable.

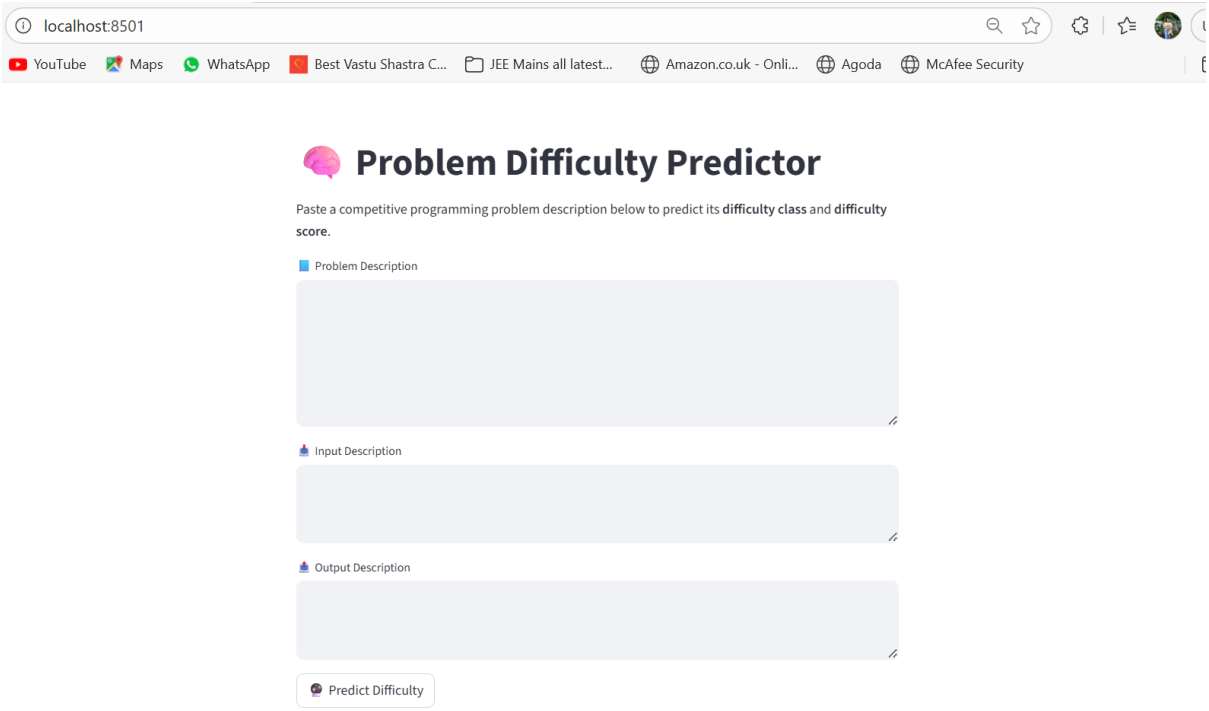


9. Web Interface

A web interface is implemented using **Streamlit** to demonstrate the trained models.

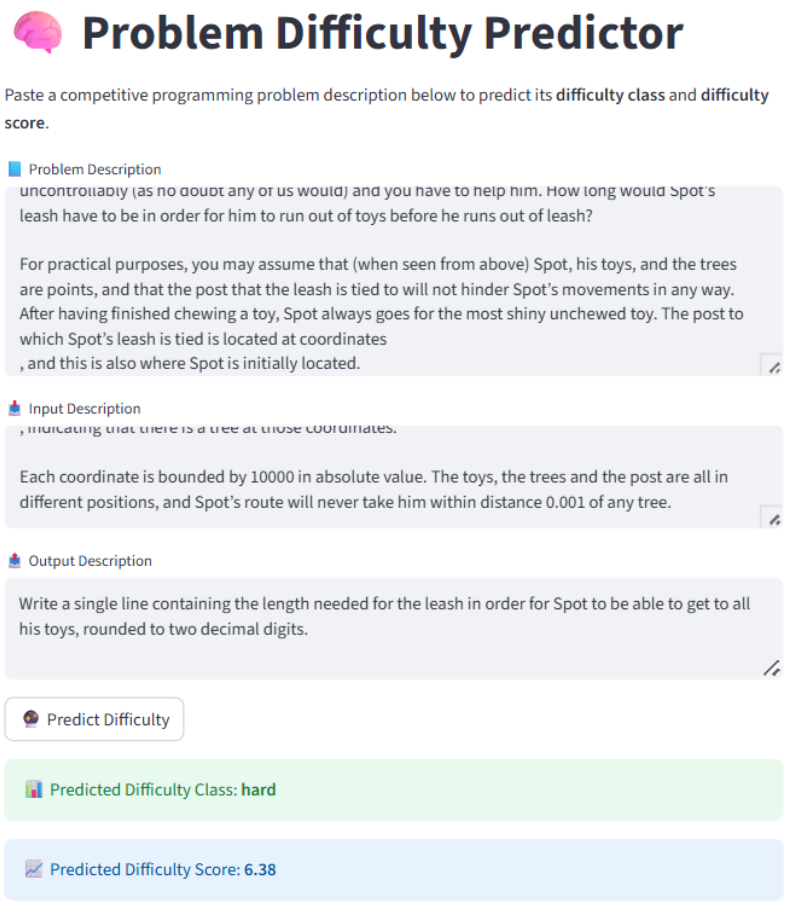
Features:

- Text input fields for problem description, input description, and output description
- Real-time prediction of difficulty class and score
- No model retraining at runtime



The interface provides an intuitive way to test unseen problems and showcases the practical applicability of the system.

10. Sample Predictions



The system successfully produces consistent predictions that align with the perceived complexity of input problems.

11. Conclusion

This project demonstrates an end-to-end machine learning pipeline for automatic difficulty estimation of programming problems using textual data. While classification accuracy is inherently limited by subjective labels and overlapping vocabularies, regression-based difficulty prediction provides stable and meaningful estimates. The results highlight both the potential and limitations of text-based difficulty estimation. Future improvements could include incorporating constraint parsing, richer semantic embeddings, and hierarchical classification approaches.