AutoJudge: Programming Problem Difficulty Prediction Using Machine Learning

1. Introduction

1.1 Problem Statement

Online coding platforms such as competitive programming judges and educational portals categorize problems into difficulty levels like *Easy*, *Medium*, and *Hard*. This categorization is usually done **manually**, making it subjective, inconsistent, and time-consuming.

The goal of this project is to **automatically predict**:

- The **difficulty class** (Easy / Medium / Hard)
- A **numerical difficulty score (1–10)**

using **machine learning and natural language processing (NLP)** techniques based on the textual description of programming problems.

1.2 Objectives

- Build a **classification model** to predict difficulty level
- Build a **regression model** to predict difficulty score
- Compare multiple ML models and select the best
- Deploy the trained models using a **web interface**
- Provide an end-to-end automated judging system

---

2. Dataset Description

2.1 Dataset Overview

- File name: problems_data.jsonl
- Format: JSON Lines
- Each line corresponds to **one programming problem**

2.2 Dataset Fields

Field Name	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 (1–10)

```
df = pd.read_json("problems_data.jsonl", lines=True)
df.head()
```

[80] ✓ 0.8s Python

	title	description	input_description	output_description	sample_io	problem_class	problem_score	url
0	Uuu	Unununiun (Uuu) was the name of the chemical\n...	The input consists of one line with two intege...	The output consists of M lines where the i...	[{'input': '7 10', 'output': '1 2 2 3 1 3 3 4 ...	hard	9.7	https://open.kattis.com/problems/uuu
1	House Building	A number of eccentrics from central New York h...	The input consists of 10 test cases, which a...	Print K lines with\n the positions of the...	[{'input': '0 2 3 2 50 60 50 30 50 40', 'outpu...	hard	9.7	https://open.kattis.com/problems/husbygge
2	Mario or Luigi	Mario and Luigi are playing a game where they ...			[{'input': '', 'output': ''}]	hard	9.6	https://open.kattis.com/problems/marioorluigi
3	The Wire Ghost	Žofka is bending a copper wire. She starts wit...	The first line contains two integers L and $...	The output consists of a single line consistin...	[{'input': '4 3 3 C 2 C 1 C', 'output': 'GHOST...	hard	9.6	https://open.kattis.com/problems/thewireghost
4	Barking Up The Wrong Tree	Your dog Spot is let loose in the park. Well, ...	The first line of input consists of two intege...	Write a single line containing the length need...	[{'input': '2 0 10 0 10 10', 'output': '14.14'...	hard	9.6	https://open.kattis.com/problems/barktree

---

3. Data Preprocessing

3.1 Text Cleaning

Each text field undergoes the following preprocessing:

- Conversion to lowercase
- Removal of special characters and symbols
- Removal of extra whitespaces
- Handling missing values by replacing with empty strings

```
def clean_text(text):
    text = str(text).lower()
    text = re.sub(r"[^a-z0-9\s]", " ", text)
    text = re.sub(r"\s+", " ", text)
    return text.strip()
```

3.2 Text Combination

All text fields are combined into a single feature:

combined_text = title + description + input_description + output_description

This helps capture the **complete semantic meaning** of the problem.

---

4. Feature Engineering

4.1 TF-IDF Vectorization

- Technique used: **TF-IDF (Term Frequency–Inverse Document Frequency)**
- N-gram range: (1, 2) → unigrams and bigrams
- Maximum features: 15,000
- Stopwords removed: English stopwords

This converts raw text into numerical vectors suitable for ML models.

4.2 Adding Text Complexity Features

I have tried this features scaling but this resulted in decreasing accuracy so I dropped that idea and sticked to simple model.

---

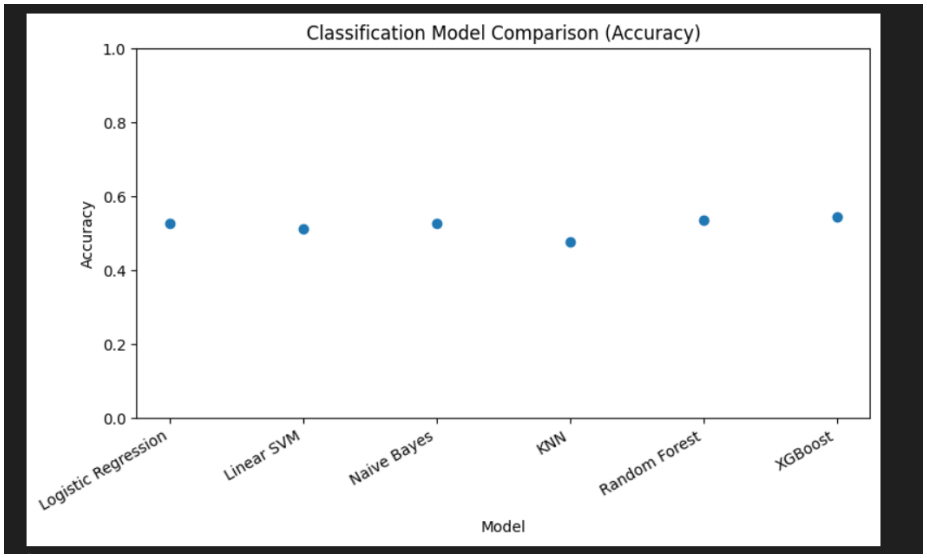
5. Models Used

5.1 Classification Models

The following models were trained to predict **difficulty class**:

Model	**Accuracy**
Logistic Regression	0.5286
Linear SVM	0.5128
Naive Bayes	0.5261
KNN	0.4763
Random Forest	0.5358
XGBoost	**0.5444**

✅ **Best Classification Model: XGBoost**

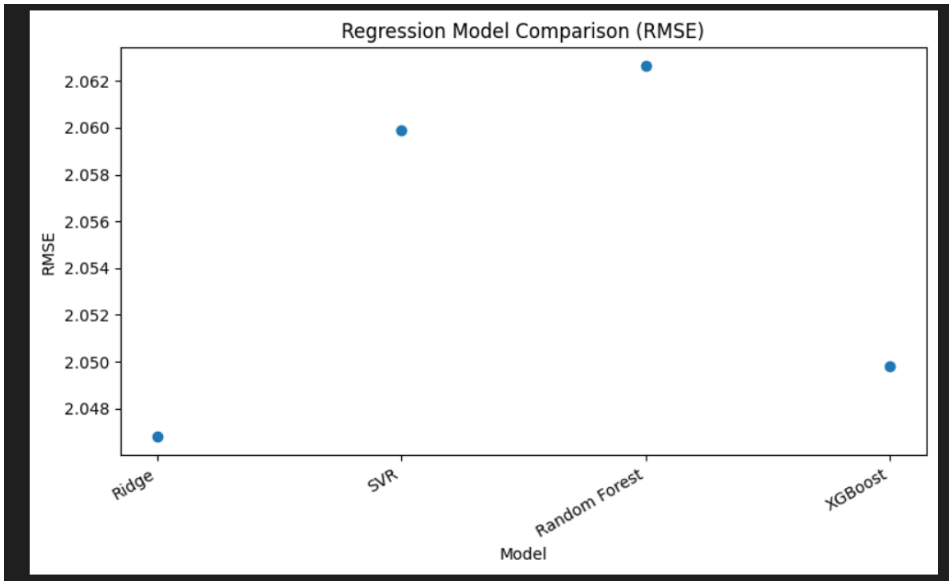


5.2 Regression Models

The following models were trained to predict **difficulty score**:

Model	MAE	RMSE
Ridge Regression	**1.7154**	**2.0468**
XGBoost	1.6895	2.0498
SVR	1.7279	2.0599
Random Forest	1.7262	2.0626

✅ **Best Regression Model: Ridge Regression**



---

6. Experimental Setup

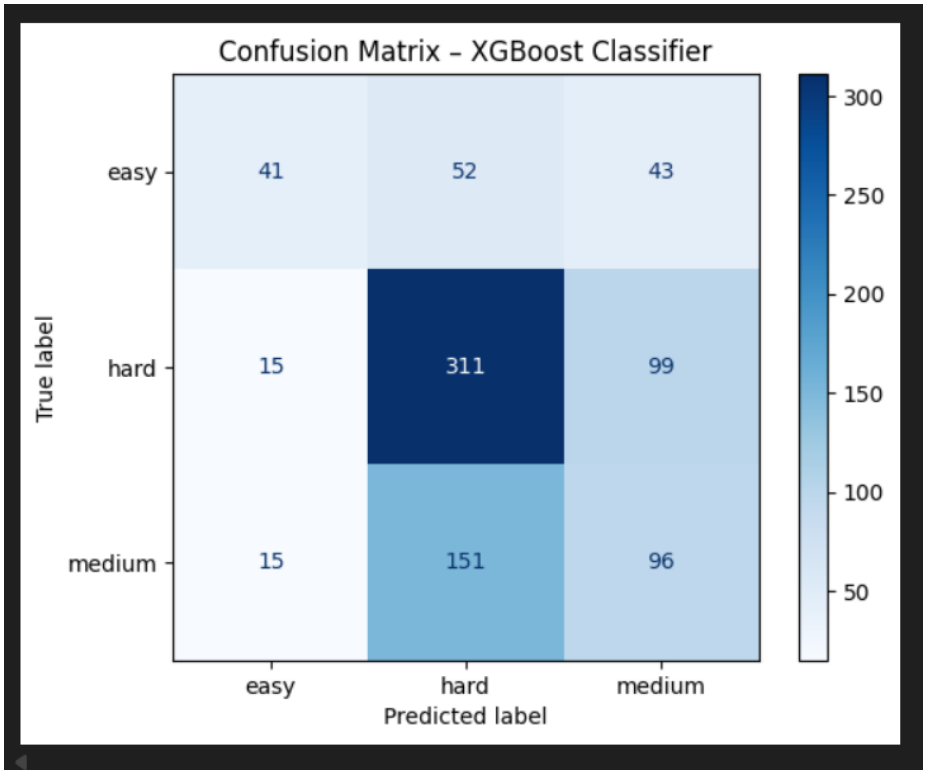
- Train-test split: **80% training, 20% testing**
- Random seed: 42
- Evaluation Metrics:
    - Classification → Accuracy, confusion matrix
    - Regression → MAE, RMSE

---

7. Results & Evaluation

7.1 Classification Performance

- XGBoost achieved the highest accuracy of **54.43%**
- Performs well on noisy textual data



7.2 Regression Performance

- Ridge Regression provided the **lowest RMSE**
- More stable predictions compared to tree-based regressors

7.3 Sample Prediction

Input:

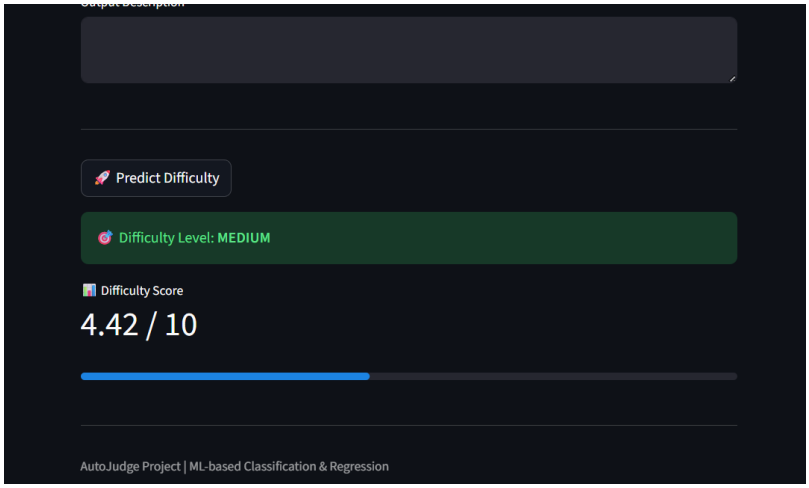
Title: Shortest Path

Description: Find shortest path using graph

Output:

Difficulty Class: Medium

Difficulty Score: 4.42



---

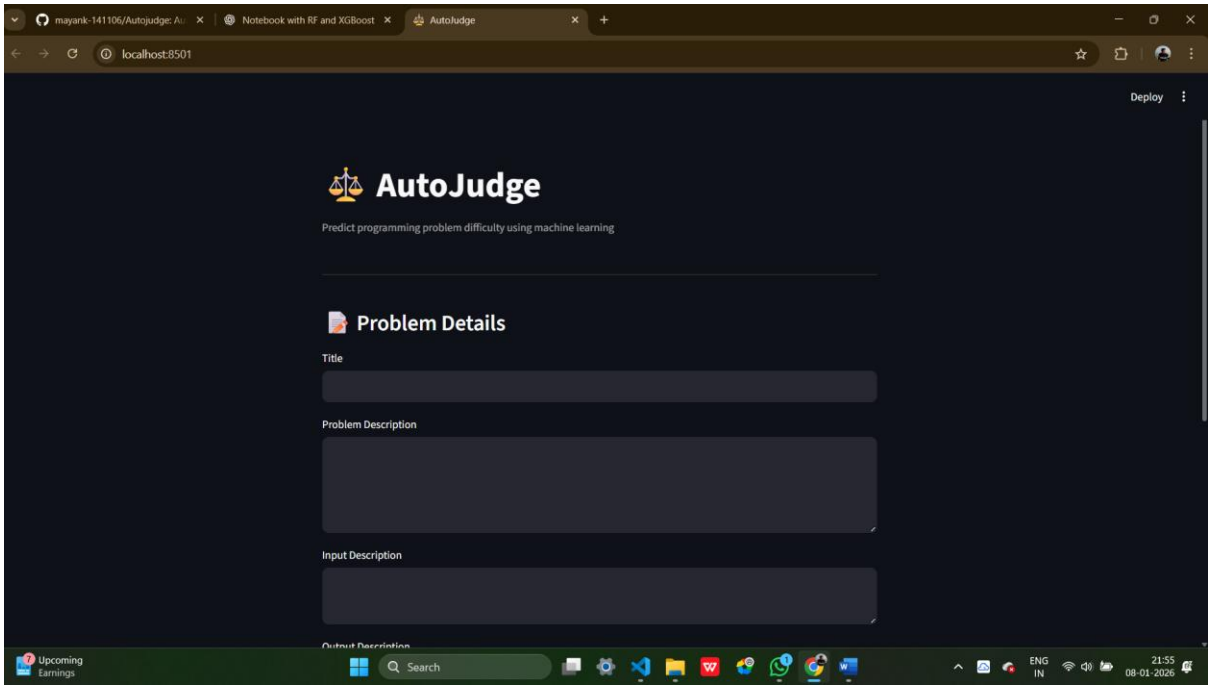
8. Web Interface

8.1 Technology Used

- **Streamlit** for web UI
- **Joblib** for model loading

8.2 Interface Features

- User inputs problem details
- Predicts difficulty instantly
- Displays score with progress bar
- Clean and responsive UI



---

9. Conclusion

This project demonstrates an effective **end-to-end ML pipeline** for predicting programming problem difficulty using textual data. By combining NLP, machine learning, and web deployment, AutoJudge provides a scalable and automated solution suitable for coding platforms.

Future Improvements:

- Deep learning (BERT-based models)
- Larger datasets
- Multilingual support
- Confidence-based predictions