AutoJudge: Predicting Programming Problem Difficulty Using Machine Learning

Name: Aayushi Sinha

Enrolment No.: 24115004

[B. Tech UG-2-Year Electrical Engg**.]**

---

1. Introduction

Online competitive programming platforms such as Codeforces, CodeChef, and Kattis categorize programming problems based on difficulty levels such as *Easy*, *Medium* and *Hard*, often accompanied by a numerical difficulty score. These difficulty labels are typically assigned through expert judgment or user feedback, which may be subjective and time-consuming.

The objective of this project is to build **AutoJudge**, an intelligent machine learning-based system that automatically predicts:

1. The **difficulty class** of a programming problem (Easy / Medium / Hard)
2. The **difficulty score** of the problem (numerical value on a scale of 0-10)

The prediction is **solely based on the textual content** of the problem which includes its description, input format and output format. The system provides a **web-based interface** that allows users to input new programming problem statements and obtain predictions in real time.

2. Problem Statement

Given the textual description of a programming problem, the task is to:

- Perform **classification** to predict the difficulty class (Easy / Medium / Hard)
- Perform **regression** to predict a numerical difficulty score (0-10 scale)

Constraints:

- Only textual information can be used
- No human feedback or historical solve data is allowed
- The system must work locally and provide predictions through a web-interface

3. Dataset Description

The dataset used in this project consists of programming problems collected from online coding platforms. Each data sample contains the following fields:

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

- **Problem score** (numerical difficulty value)

The dataset is stored in **JSONL (JSON Lines)** format in which each line represents a single problem entry. The dataset already contains labelled difficulty classes and scores, eliminating the need for manual labelling or annotation. It is available in a GitHub repository.
Dataset source (reference):
https://github.com/AREEG94FAHAD/TaskComplexityEval-24

4. Data Preprocessing

Before training the models, several preprocessing steps were applied to clean and standardize the textual data:

4.1 Text Cleaning

- Converted all text to lowercase
- Removed special characters except mathematical symbols (+ - * / = < >)
- Removed unnecessary whitespace
- Combined problem description, input description and output description into a single text field

4.2 Handling Missing Values

- Missing textual fields were handled by ignoring incomplete records
- Only valid problem entries were used for training

5. Feature Engineering

Feature engineering plays a crucial role in transforming raw text into meaningful numerical representations.

5.1 TF-IDF Features

- Used **TF-IDF (Term Frequency–Inverse Document Frequency)** to convert text into numerical vectors
- It captures the importance of words relative to the entire dataset

5.2 Keyword Frequency Features

- A predefined list of programming-related keywords (e.g., *graph*, *dp*, *recursion*, *array*) was used
- The frequency of each keyword in the problem text was counted

5.3 Numeric Features

Additional numeric features were extracted:

- Length of the combined text
- Number of digits in the text
- Count of mathematical symbols

5.4 Final Feature Vector

All features were concatenated into a single vector:

Final Features = TF-IDF + Keyword Counts + Numeric Features

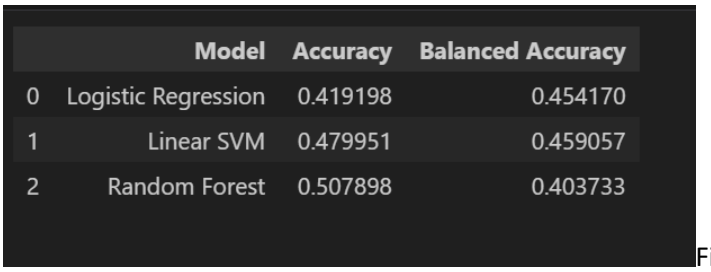
6. Machine Learning Models

Two separate models were trained:

6.1 Classification Model

- **Objective:** Predict difficulty class of programming problem (Easy / Medium / Hard)
- **Algorithm Used:** Random Forest Classifier
- **Reason for Selection:**
    - It demonstrated the best classification accuracy among the evaluated models.
    - It is robust to noisy and imbalanced labels in the dataset, which is important here given the subjective nature of problem difficulty.
    - It can capture non-linear relationships in TF-IDF based text features without extensive hyperparameter tuning.
    - It is computationally heavier than linear models, but the training is performed offline, this makes the trade-off acceptable for a better predictive performance.

  Logistic regression and Linear SVM were also evaluated as baseline models for comparison.



	Model	Accuracy	Balanced Accuracy
0	Logistic Regression	0.419198	0.454170
1	Linear SVM	0.479951	0.459057
2	Random Forest	0.507898	0.403733

Figure 1: Accuracy Comparison

6.2 Regression Model

- **Objective:** Predict numerical difficulty score for a programming problem (out of 10)
- **Algorithm Used:** Ridge Regression (L2-regularized Linear Regression)
- **Reason for Selection:**
    - Linear regression was used as the baseline regression model.
    - Ridge regression significantly reduced MAE and RMSE compared to standard Linear regression.
    - Regularization improves stability and generalization on high-dimensional and sparse TF-IDF features.
    - It is computationally efficient and even well suited for text-based regression tasks.

Tree-based regression models such as Random Forest Regressor and Gradient Boosting Regressor were tried too but excluded due to poor scalability and excessive computation time on sparse TF-IDF representations.

	Model	MAE	RMSE
0	Linear Regression	1.793677	2.199446
1	Ridge Regression	1.770190	2.106931

Figure 2: Regression Model comparison

7. Experimental Setup

- Programming Language: Python
- Libraries Used:
  - scikit-learn
  - NumPy
  - SciPy
  - Streamlit
  - Joblib
  - pandas
- Dataset split into training and testing sets.
- Same feature extraction pipeline used for both models.

8. Evaluation Metrics

8.1 Classification Evaluation

- **Accuracy** was used to measure classification performance (Figure 1)
- A **confusion matrix** was analysed to understand misclassifications between Easy, Medium, and Hard categories

	Pred Easy	Pred Medium	Pred Hard
Actual Easy	23	28	102
Actual Medium	8	46	227
Actual Hard	9	31	349

Figure 3: Confusion matrix

8.2 Regression Evaluation

- **Mean Absolute Error (MAE)**

- **Root Mean Squared Error (RMSE)**

These metrics quantify how close the predicted difficulty score is to the actual score. (Figure 2)

9. Web Interface

A web-based interface was developed using **Streamlit** to allow real time predictions of entered programming problem description.

Features:

- Text boxes for:
    - Problem description
    - Input description
    - Output description
- A “Predict” button to generate results
- Displays:
    - Predicted difficulty class
    - Predicted difficulty score (out of 10)

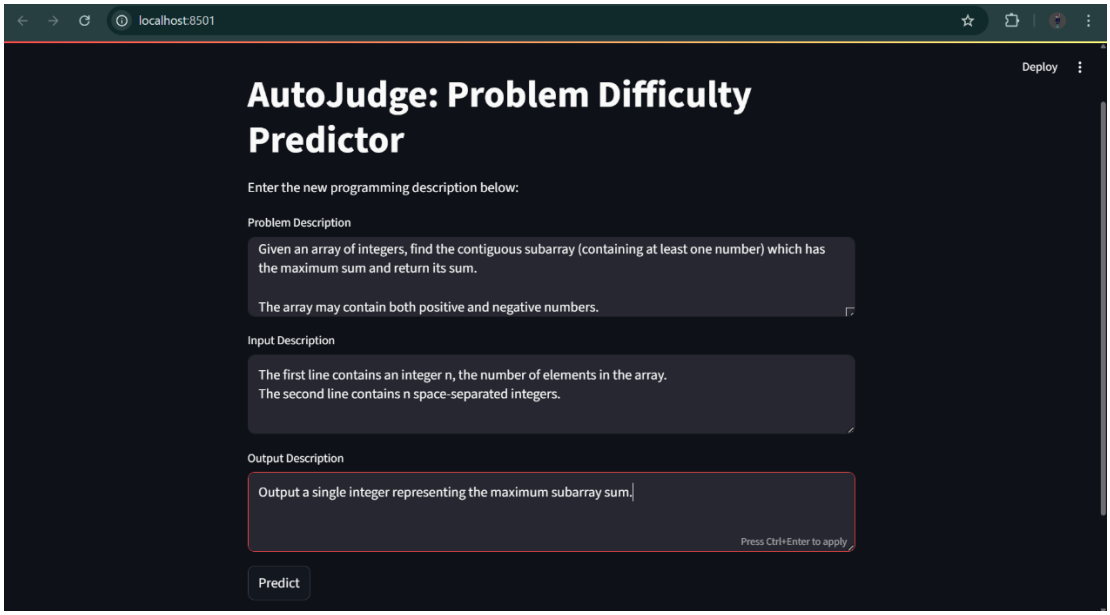
The web application runs locally and loads the trained models to provide instant predictions.

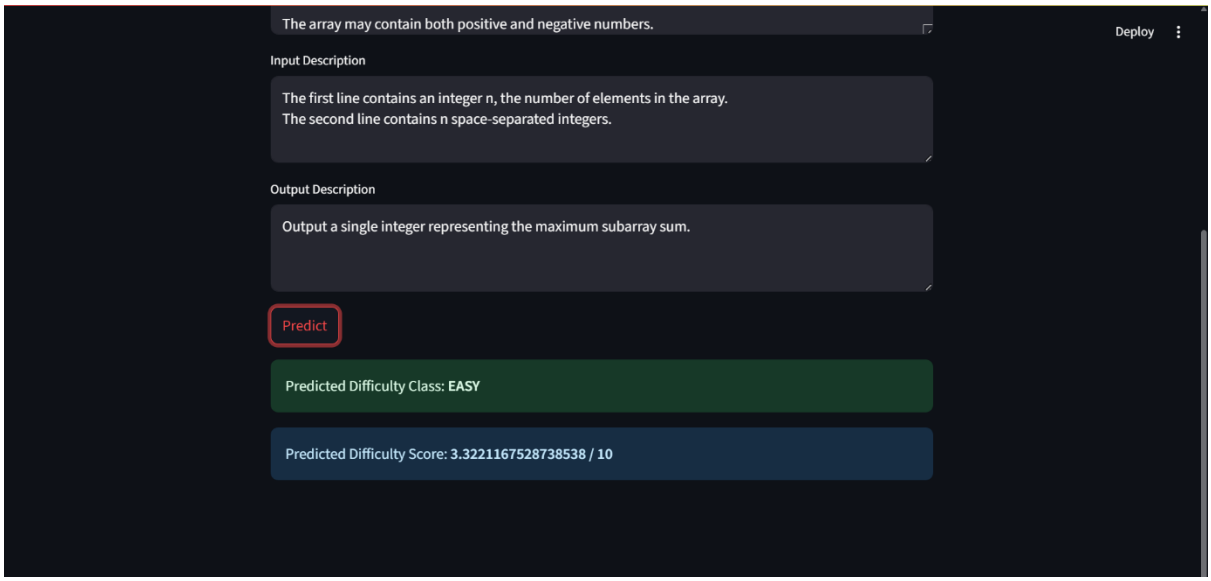
10. Results and Sample Predictions

Sample predictions demonstrate that:

- Problems with simple logic and minimal constraints are classified as **Easy.**
- Problems involving advanced algorithms or complex constraints are classified as **Hard.**
- Problems that are neither Easy nor Hard but fall in between them are classified as Medium.
- The predicted scores closely align with the expected difficulty levels.

Screenshots of the web interface and prediction outputs are included to validate system functionality.





11. Conclusion

This project successfully demonstrates that programming problem difficulty can be predicted using only textual information. The system achieves reliable classification and regression performance by combining TF-IDF features, keyword analysis, and numeric indicators with classical machine learning models.

The **AutoJudge** system provides a scalable and automated solution that can help educators, competitive programming platforms, and content creators in evaluating problem difficulty objectively.

12. Future Work

Possible improvements include:

- Using advanced models such as Gradient Boosting or Transformers.
- Expanding the keyword list dynamically.

- Training on larger and more diverse datasets.
- Deploying the system as a hosted web service.

13. References

1. Scikit-learn Documentation
2. Streamlit Documentation
3. TF-IDF Feature Extraction Techniques