# AutoJudge – Predicting Programming Problem Difficulty

## Project source code github: source code

# 1. Introduction

### 1.1 Problem Statement

Online coding platforms have thousands of programming problems and are labeled with a difficulty level such as Easy, Medium, or Hard. These labels have an important role in guiding new learners and organizing contests. However, doing these tasks manually is time-consuming, subjective, and often inconsistent.

In this project I will  be making a machine learning model for predicting the difficulty of programming problems. It achieves this by applying ML techniques to the textual descriptions of the problems, resulting in a system that predicts:

1. **difficulty level** (Easy / Medium / Hard)
2. A **numerical difficulty score** (1 - 10)

### 1.2 Objective

The objective of this project is to develop a machine learning pipeline that uses Natural Language Processing (NLP) to perform both **classification and regression** tasks. Additionally, construct a **web-based interface** to facilitate users .

# 2. Dataset Description

### 2.1 Dataset Overview

Here I have used the reference dataset given in the problem statement file.

The dataset consists of data samples . Each data sample contains the following fields:

- **Problem Statement:** Description of the problem
- **Input Description:** Details of the input format
- **Output Description:** Expected output format

- **Difficulty Label:** Easy, Medium, or Hard
- **Difficulty Score**

The dataset consists of **4112** samples , making it perfect for **supervised learning**.

### 2.2 Target Variables

- **Classification Target:** Easy / Medium / Hard
- **Regression Target:** Difficulty score (1–10)

# 3. Data Preprocessing

### 3.1 Text Cleaning

The raw textual data gone through the following preprocessing steps to prepare it for further use:

- all text has been converted to lowercase .
- All kinds of punctuation, special characters and  stopwords are removed.

### 3.2 Text Aggregation

The problem statement, input description, and output description were combined into a **single text feature**. This makes sure that all relevant information contributes to the result.

### 3.3 Tokenization

Tokenization was performed on the cleaned text to convert the words into numerical vectors, which are further used for feature extraction.

# 4. Feature Engineering

### 4.1 TF-IDF Vectorization

Cleaned text was converted into numerical feature vectors using the **Term Frequency–Inverse Document Frequency (TF-IDF)** technique.

TF-IDF gives higher weight to important words while reducing the influence of frequently occurring less informative words. This technique is preferred for NLP based classification and regression tasks.

# 5. Models Used

## 5.1 Classification Model

- **Model:** Logistic Regression
- **Reason for Selection:**Logistic Regression is effective with high-dimensional, sparse text features because it delivers results that are both fast and easily interpretable.

## 5.2 Regression Model

- **Model:** Random Forest Regressor
- **Reason for Selection:**Through ensemble learning, Random Forest effectively handles non-linear relationships and minimizes the risk of overfitting.

## 5.3 Training Procedure

- Train–test split applied to the dataset
- The same TF-IDF features were applied to both tasks.
- The web application utilizes trained models by saving and subsequently reusing them.

# 6. Experimental Setup

- **Programming Language:** Python
- **Libraries Used :**NumPy,Pandas,Scikit-learn,Streamlit
- **Execution Environment:** one can run this model on Local machine using the steps given in readme on github
- **Model Storage:** Serialized models has been stored in the `models/` directory

# 7. Results and Evaluation

## 7.1 Classification Results

**Accuracy:**
0.8927 (≈ 89.27%)

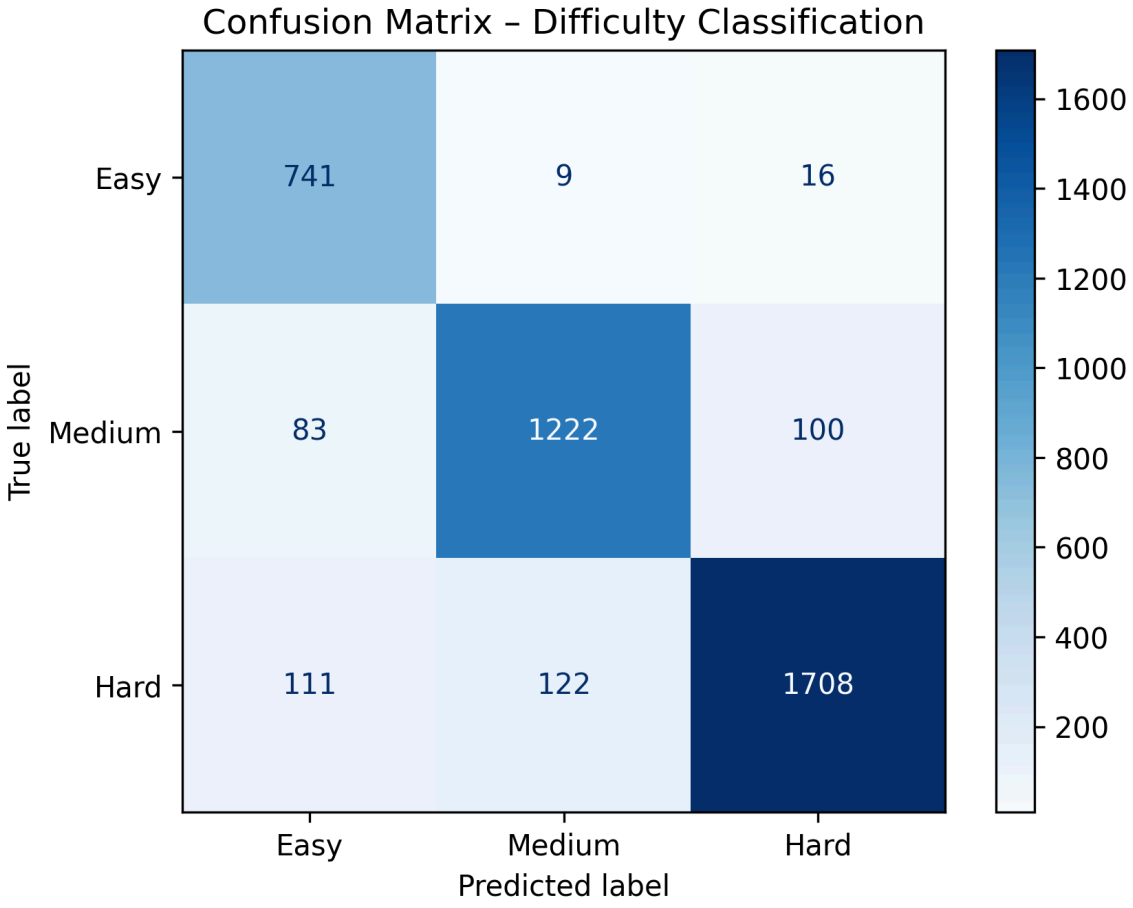
Classification Report:

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Easy | 0.79 | 0.97 | 0.87 | 766 |
| Medium | 0.90 | 0.87 | 0.89 | 1405 |
| Hard | 0.94 | 0.88 | 0.91 | 1941 |

**Macro Average F1-score:** 0.89

**Weighted Average F1-score:** 0.89

**Confusion matrix:**




## 7.2 Regression Results

| Metric | Value |
|---|---|
| MAE | 1.332 |
| RMSE | 1.574 |
| R²Score | 0.477 |

The regression model predicts a numerical difficulty score with an average error of approximately **1.33**, for a 1–10 scale.

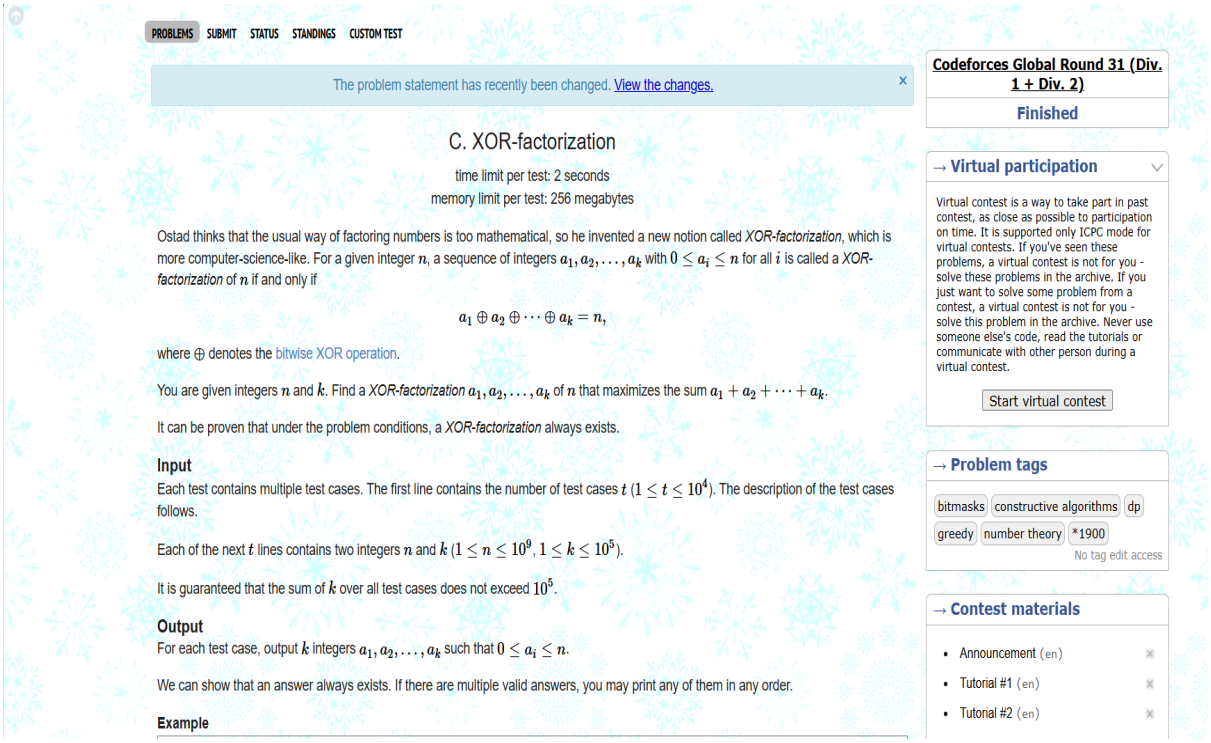
# 8. Web Interface

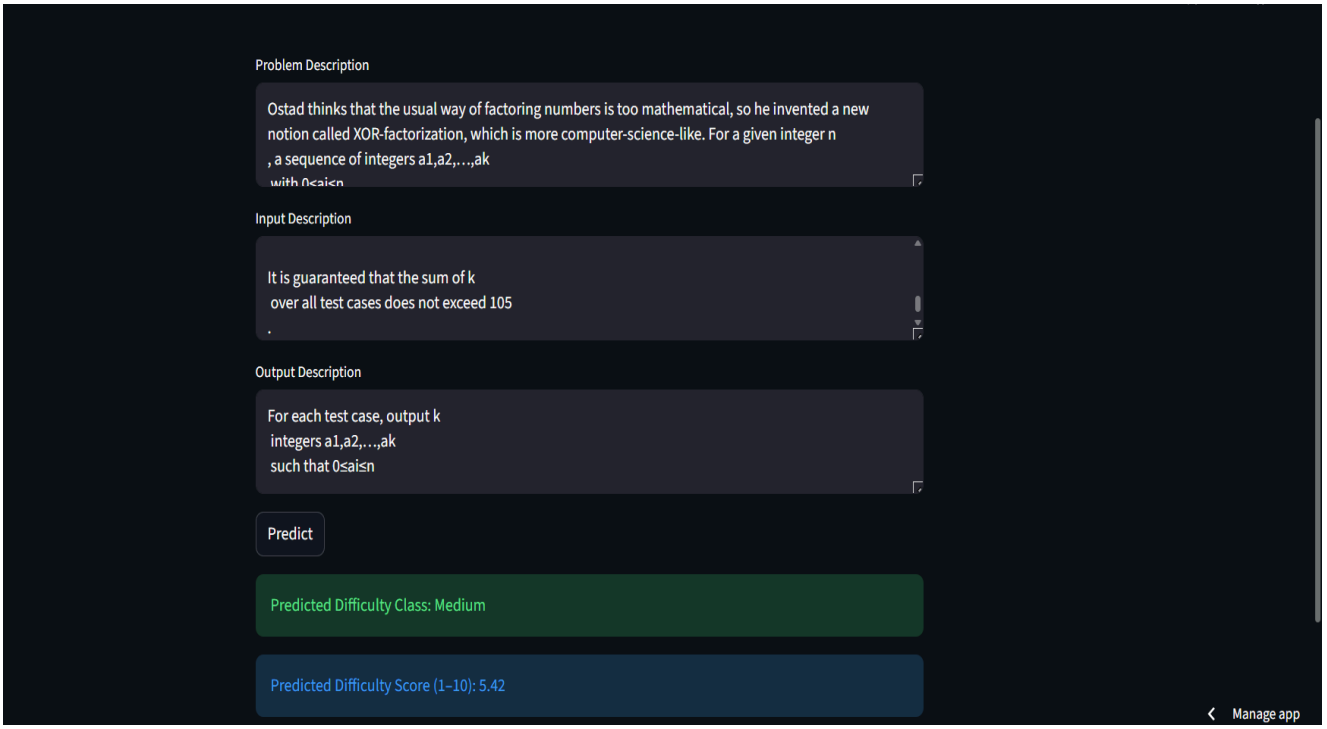
A web application was developed using **Streamlit**.

User can check model on stramlit : [web app]

- Enter a programming problem description 3 sections
    1. Problem 2.Input description 3. Output description.
- Click on predict button
- User will Receive difficulty category and score

**Ex.** sample problem run






# 9. Conclusion

This project is a machine learning model that  can effectively predict programming problem difficulty using textual data only. The classification and regression models achieved strong accuracy, and the web interface makes the system user-friendly.

# 11. Author Information

**Name:** Rohit Mahawar
**Email:** rohit_m@cs.iitr.ac.in
**Program:** B.Tech Computer Science and Engineering(2nd year)