# Project Report

## AUTOJUDGE: PROGRAMMING PROBLEM DIFFICULTY PREDICTION USING MACHINE LEARNING

**NAME:** NAIMISH MEHTA
**ENROLLMENT NO. :** 23112059

## 1. Introduction

Online coding platforms categorize programming problems into difficulty levels such as Easy, Medium, and Hard, often accompanied by a numerical difficulty score. These labels are usually assigned manually or inferred from user performance, making them subjective and inconsistent.

This project, **AutoJudge**, aims to **automatically predict the difficulty of programming problems using only their textual descriptions**. The system performs two tasks:

1. **Classification** – Predicting difficulty class (Easy/Medium/Hard)
2. **Regression** – Predicting a numerical difficulty score

The complete pipeline includes data preprocessing, feature extraction, machine learning models, and a web-based interface for real-time predictions.

## 2. Dataset Description

The dataset consists of programming problems collected from online coding platforms. Each data point contains both textual and numerical information.

### 2.1 Features

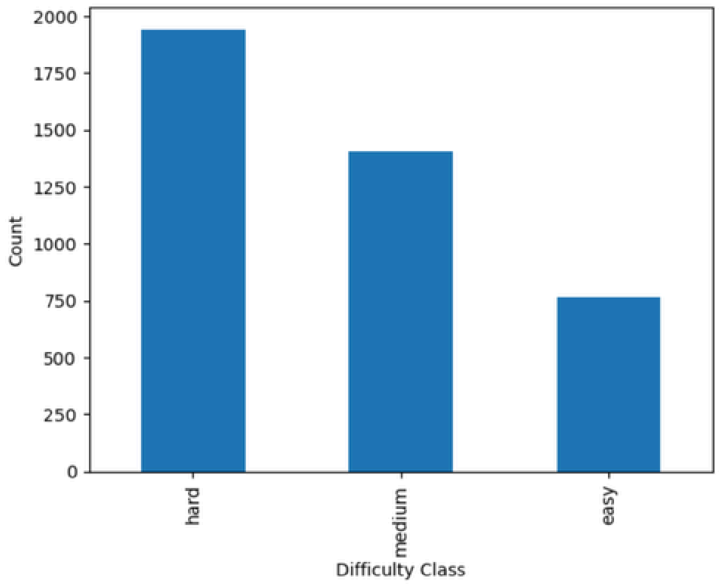
- **title:** Problem title
- **description:** Problem statement
- **input_description:** Input format
- **output_description:** Output format
- **problem_class:** Difficulty label (Easy / Medium / Hard)
- **problem_score:** Numerical difficulty score

The dataset was originally provided in **JSONL format** and converted into CSV for preprocessing and model training.

# 3. Exploratory Data Analysis (EDA)

Exploratory Data Analysis was performed to understand data distribution, class balance, and target behavior.

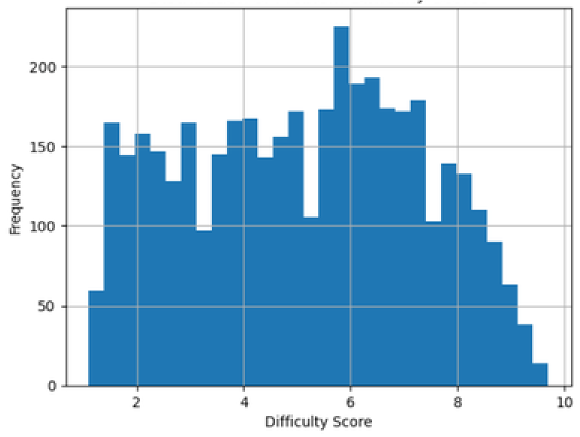
## 3.1 Difficulty Class Distribution



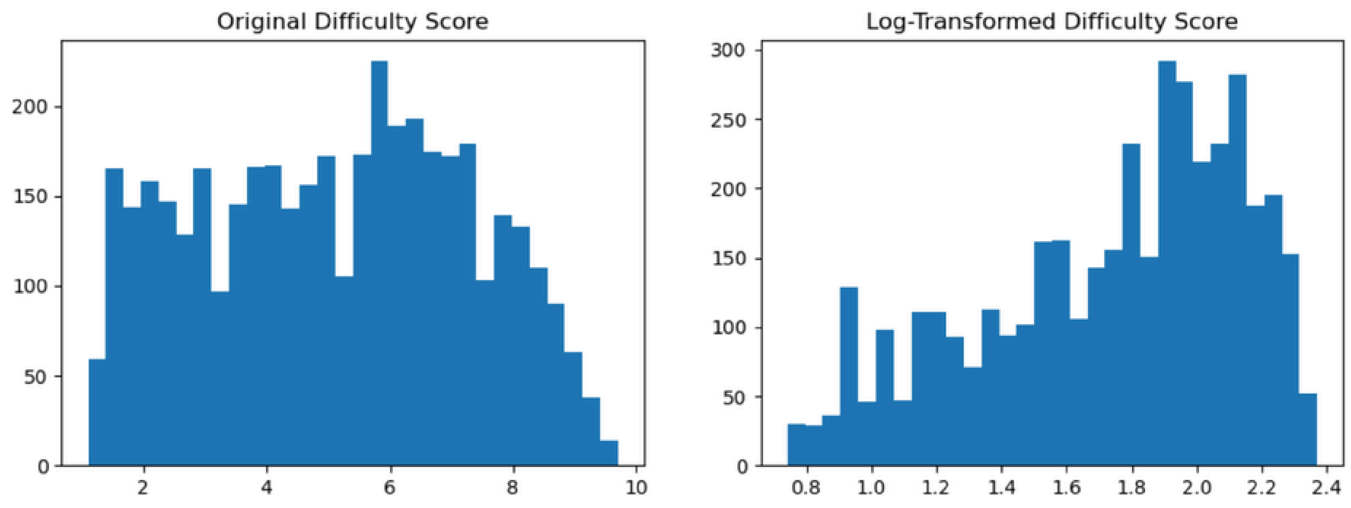
This plot shows the distribution of Easy, Medium, and Hard problems. The visualization helps identify any class imbalance, which is important for classification performance evaluation.

## 3.2 Difficulty score Distribution

The difficulty scores show a skewed distribution, indicating the need for transformation before regression modeling.



## 3.2 Difficulty score Distribution



A log transformation (log1p) was applied to the difficulty score to reduce skewness and stabilize variance, leading to improved regression performance.

## 4. Data Preprocessing

The following preprocessing steps were applied:

- Missing text values were replaced with empty strings
- All textual fields were concatenated into a single feature
- Numerical difficulty score was log-transformed

**Final input text:**
title + description + input_description + output_description

These steps ensured consistency and reduced noise in the input data.

## 5. Feature Engineering

Textual data was converted into numerical features using **TF-IDF Vectorization** with the following configuration:

- Unigrams, bigrams, and trigrams
- Stopword removal
- Sublinear term frequency scaling
- Maximum features limited to **15,000**

TF-IDF effectively captures term importance while maintaining sparsity for large vocabularies.

## 6. Model Architecture

Two separate models were trained for the two tasks.

### 6.1 Classification Model

- **Algorithm: Random Forest Classifier**
- **Objective:** Predict difficulty class (Easy / Medium / Hard)

Random Forest was chosen for its robustness, ability to model non-linear relationships, and resistance to overfitting.

### 6.2 Regression Model

- **Algorithm: XGBoost Regressor**
- **Objective:** Predict numerical difficulty score

XGBoost was selected due to its strong performance on structured data and its ability to capture complex feature interactions.

# 7. Experimental Setup

- Train-test split: 80% training, 20% testing
- Random seed fixed for reproducibility
- Models trained on TF-IDF features only
- No user statistics or submission data used

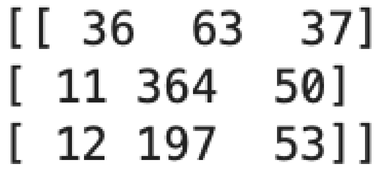
# 8. Results and Evaluation

### 7.1 Classification Results

Metrics Used:

- Accuracy
- Confusion Matrix

The **Random Forest classifier** achieved an o**verall accuracy** of **55.04%** on the test dataset.



```
[[ 36  63  37]
 [ 11 364  50]
 [ 12 197  53]]
```

The confusion matrix shows how well the model distinguishes between difficulty classes and highlights common misclassification patterns.

### 7.2 Regression Results

Metrics Used

- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)

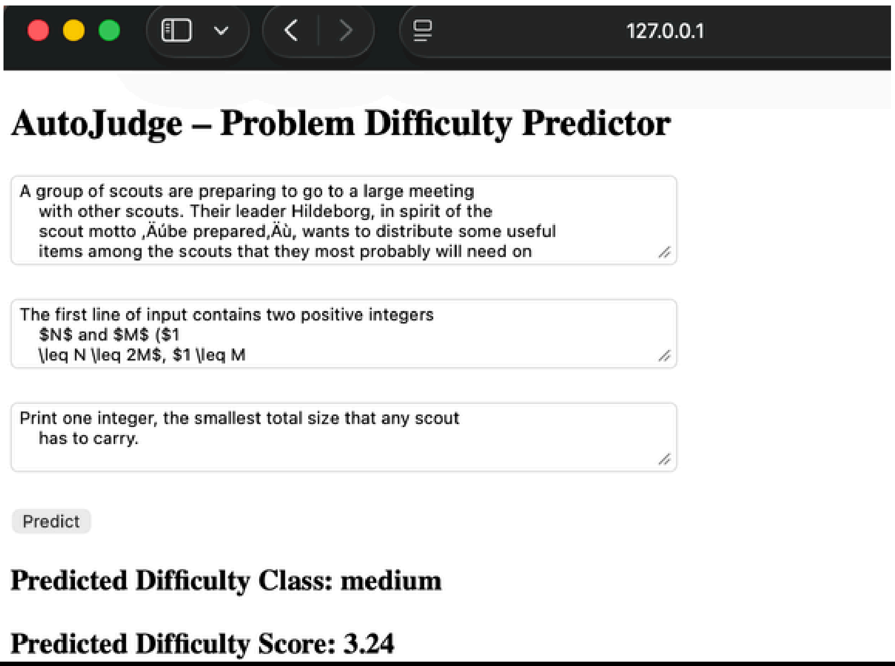
The XGBoost regression model achieved a **Mean Absolute Error (MAE)** of **0.299** and a **Root Mean Squared Error (RMSE)** of **0.360** on the test dataset.

These results indicate that the model is able to predict difficulty scores with reasonable accuracy, considering the subjective nature of problem difficulty and the use of textual features alone.

## 9. Web Application Interface

A Flask-based web application was developed to demonstrate real-time predictions.

The application runs locally and integrates the trained models using joblib.



AutoJudge – Problem Difficulty Predictor

A group of scouts are preparing to go to a large meeting
with other scouts. Their leader Hildeborg, in spirit of the
scout motto ,Äúbe prepared,Äù, wants to distribute some useful
items among the scouts that they most probably will need on

The first line of input contains two positive integers
$N$ and $M$ ($1
\leq N \leq 2M$, $1 \leq M

Print one integer, the smallest total size that any scout
has to carry.

Predicted Difficulty Class: medium

Predicted Difficulty Score: 3.24

## 10. Conclusion

This project demonstrates an end-to-end machine learning pipeline for predicting programming problem difficulty using textual data alone. The system successfully integrates preprocessing, feature extraction, classification, regression, and deployment into a single application.

Despite the subjective nature of difficulty scoring, the models achieve reasonable performance and provide meaningful predictions.

## 11. Future Work

- Sentence embeddings (BERT / SBERT)
- Ordinal regression techniques
- Cloud deployment
- Dataset expansion

## 12. References

- Scikit-learn Documentation
- XGBoost Documentation
- TF-IDF Vectorization (Salton & Buckley)