REPORT

AUTOJUDGE: PROGRAMMING PROBLEM DIFFICULTY PREDICTION USING MACHINE LEARNING

Name: Mohit sharma

Enrollment no: 23112062

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, which makes them subjective and sometimes inconsistent.

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

1. **Classification** – Predicting the difficulty class (Easy, Medium, or 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.

Dataset Description

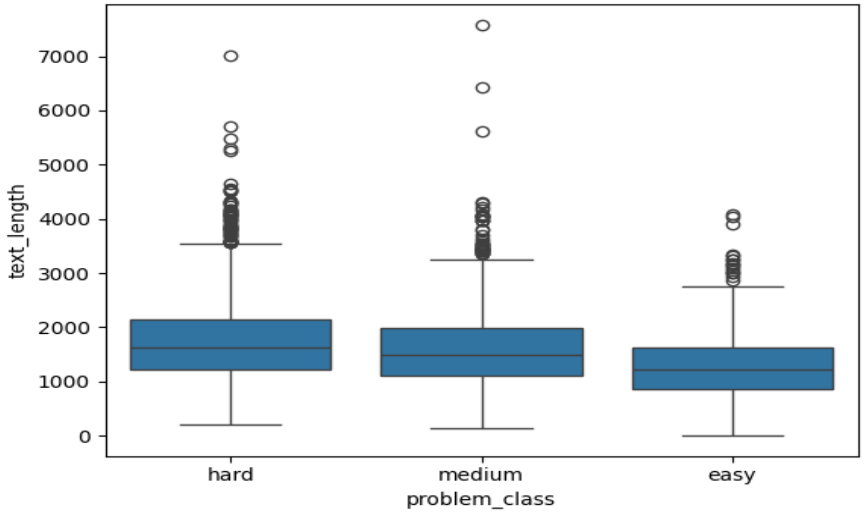
The dataset consists of programming problems collected from online coding platforms.

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

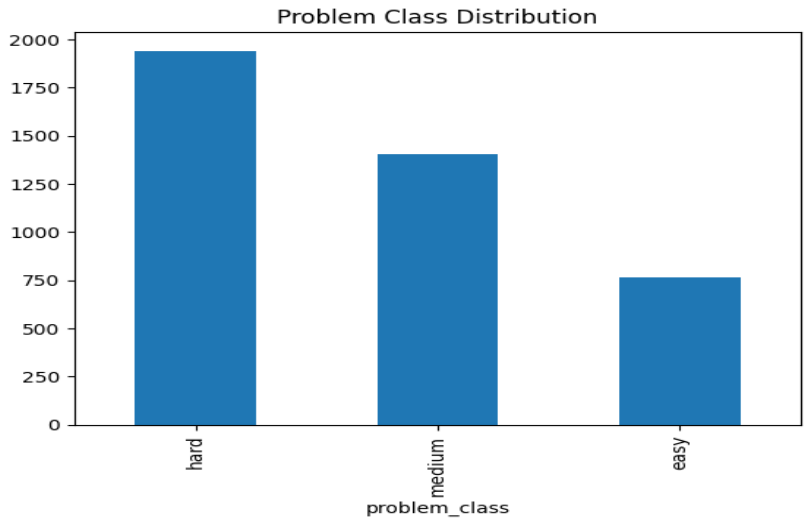
Exploratory Data Analysis

Text Length vs Problem Class



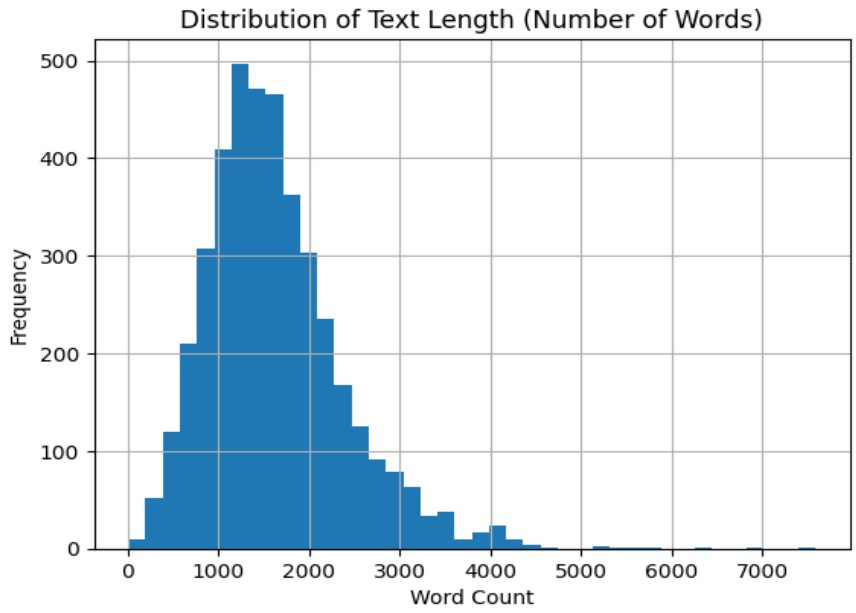
The comparison of text length across difficulty classes reveals a clear upward trend from Easy to Hard problems. **Hard problems generally have longer descriptions**, reflecting more complex constraints, explanations, and edge cases, while Easy problems tend to be more concise.

Problem Class Distribution



The class distribution plot shows a clear imbalance in the dataset, with **Hard problems appearing most frequently, followed by Medium and then Easy**. This imbalance reflects real-world competitive programming datasets, where harder problems are often more numerous.

Distribution of Text Length (Word Count)



The histogram of text length exhibits a **right-skewed distribution**, where most problems fall within a moderate word count range, while a smaller number of problems have very long descriptions.

Keyword Frequency by Difficulty Class

problem_class	count	mean	std	min	25%	50%	75%	max
easy	766.0	1.970888	0.433289	1.1	1.6	2.0	2.3	2.8
hard	1941.0	7.071149	1.049729	5.5	6.2	7.0	7.9	9.7
medium	1405.0	4.125836	0.774216	2.8	3.5	4.1	4.8	5.5

The keyword frequency analysis demonstrates a strong relationship between algorithmic terminology and problem difficulty. **Advanced terms such as graph, tree, recursion, and modulo appear far more frequently in Hard problems compared to Easy ones**, where their presence is minimal or absent.

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.

Feature Engineering

To convert raw textual problem statements into machine-learning-ready inputs, a combination of **statistical text representations and**

handcrafted features was used. The primary textual representation was obtained using **TF-IDF vectorization**.

In addition to TF-IDF features, several **auxiliary features derived directly from the text** were introduced to better capture structural and algorithmic complexity:

- **Text length**, representing the total number of characters in the combined problem description.
- **Mathematical symbol count**, capturing the density of mathematical expressions and constraints.
- **Keyword frequency features** for common algorithmic terms such as *dp, graph, tree, recursion, greedy,* and *modulo*.

Experimental setup

- Train-test split of 80% train, 20% test.
- Models trained on TF-IDF for reproducibility.
- Feature engineering to get some relevant features.

Model Architecture

Two separate models were trained to address the classification and regression tasks.

Classification Model

Algorithm: Random forest classifier
Objective: Predict difficulty class

The classification task uses a rf. This model achieved a **test accuracy of approximately 57.7%**, with stable cross-validation results, indicating consistent generalization.

Regression Model

Algorithm: XGBoost Regressor
Objective: Predict numerical difficulty score

The regression task employs XGBoost to model non-linear relationships. The difficulty score was log-transformed during training to improve stability. The **MAE and RMSE obtained were 0.29 and 0.35 respectively.**

Results and Evaluation

Classification Results

The classification model was evaluated using accuracy and a confusion matrix. The model achieved an overall **accuracy of 57.7%** on the test dataset. **Five-fold cross-validation produced a mean accuracy of 52.5%**.

Regression Results

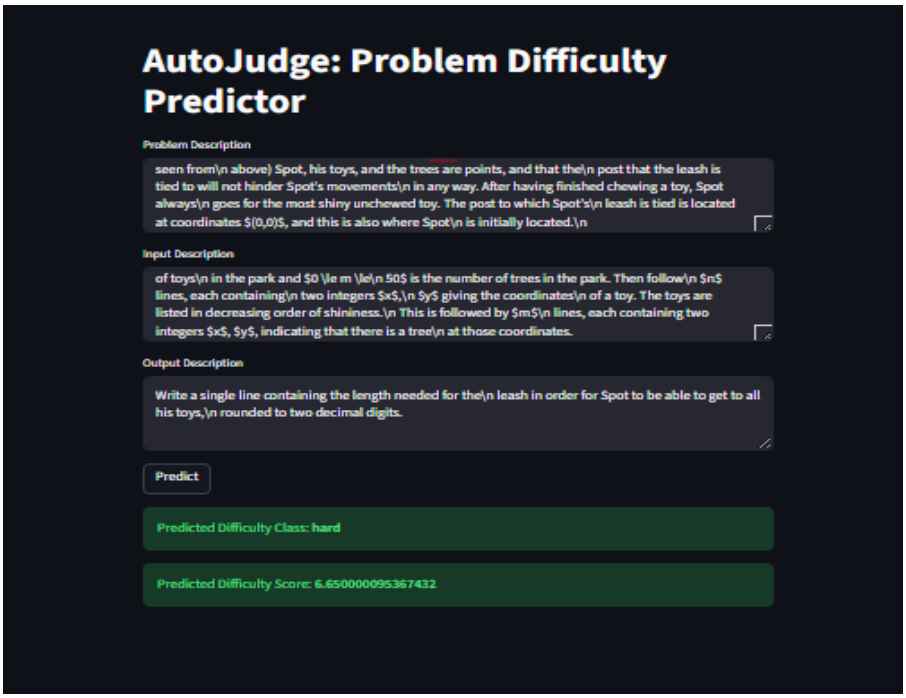
The regression model was evaluated using Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) on log-transformed difficulty scores. The model achieved an **MAE of 0.294** and an **RMSE of 0.357**.

Confusion matrix on test:

55	54	27
23	364	38
24	182	56

Web application interface:

Below is the streamlit based web application:



Conclusion

This project demonstrated that the difficulty of programming problems can be effectively estimated using only their textual descriptions. By combining TF-IDF representations with engineered textual features and machine learning models, the system was able to predict both difficulty classes and numerical scores with reasonable accuracy.