AutoJudge: Automated Difficulty Prediction for Programming Problems Using Machine Learning

Abstract

Difficulty classification of programming problems is essential for organizing coding platforms, guiding learners, and designing competitive programming contests. Manual difficulty assignment is subjective, inconsistent, and time-consuming. This project proposes **AutoJudge**, a machine learning–based system that automatically predicts problem difficulty using textual descriptions. The system performs both **multi-class classification** to assign difficulty labels (*Easy, Medium, Hard*) and **regression** to estimate a continuous numerical difficulty score. Using natural language processing techniques such as TF-IDF and handcrafted linguistic features, the system achieves meaningful performance despite the inherent subjectivity of difficulty labeling. A web interface is also developed to demonstrate real-time predictions.

1. Introduction

Online programming platforms such as coding practice websites and competitive programming portals rely on difficulty labels to help users select appropriate problems and track learning progress. These labels also assist instructors in designing curricula and competitions. However, difficulty labeling is often performed manually by problem setters, making it subjective and prone to inconsistency. The same problem may be perceived differently by different individuals based on experience and familiarity with algorithms.

The goal of this project is to develop an **automated difficulty prediction system (AutoJudge)** that estimates the difficulty of programming problems directly from their textual descriptions. The system is designed to address two complementary objectives:

1. **Classification** – Predicting a categorical difficulty level (*Easy, Medium, Hard*).
2. **Regression** – Predicting a numerical difficulty score for fine-grained assessment.

By leveraging machine learning and natural language processing techniques, AutoJudge aims to provide a scalable and interpretable solution for difficulty estimation.

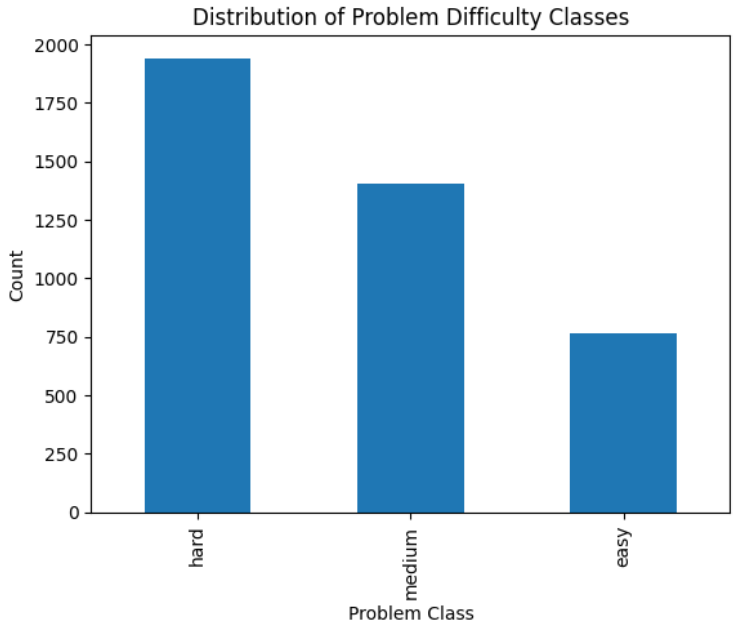
2. Dataset Description

The dataset used in this project consists of programming problems stored in **JSONL (JSON Lines) format**, where each line represents a single problem. The dataset contains both textual and numerical information.

2.1 Dataset Fields

- **Title** – Short description of the problem.
- **Problem Description** – Detailed explanation of the problem statement.
- **Input Description** – Description of input format and constraints.
- **Output Description** – Description of expected output.
- **Difficulty Class (problem_class)** – Categorical label (*Easy / Medium / Hard*).
- **Difficulty Score (problem_score)** – Continuous numerical score representing difficulty.

Exploratory analysis revealed that the dataset is **imbalanced**, with the *Hard* category containing more samples than *Easy* and *Medium* classes.



3. Data Preprocessing

Before applying machine learning models, several preprocessing steps were performed to ensure data consistency and quality.

3.1 Handling Missing Values

Some textual fields contained missing values. These were safely replaced with empty strings to prevent errors during text concatenation and feature extraction.

3.2 Text Consolidation

All textual fields (title, problem description, input description, output description) were combined into a single column called combined_text. This unified representation captures the complete problem context.

3.3 Text Cleaning

The following cleaning operations were applied:

- Conversion of text to lowercase.
- Removal of excessive whitespace.
- Retention of mathematical symbols and algorithmic terms, as they are relevant to problem difficulty.

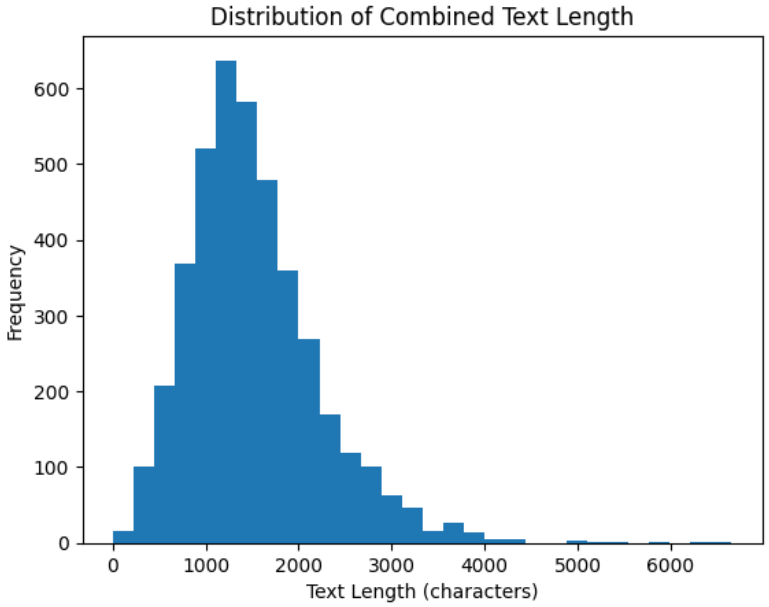
These steps reduce noise while preserving meaningful content required for difficulty prediction.

4. Exploratory Data Analysis (EDA)

Exploratory Data Analysis was conducted to understand the characteristics of the dataset and the relationship between textual features and problem difficulty.

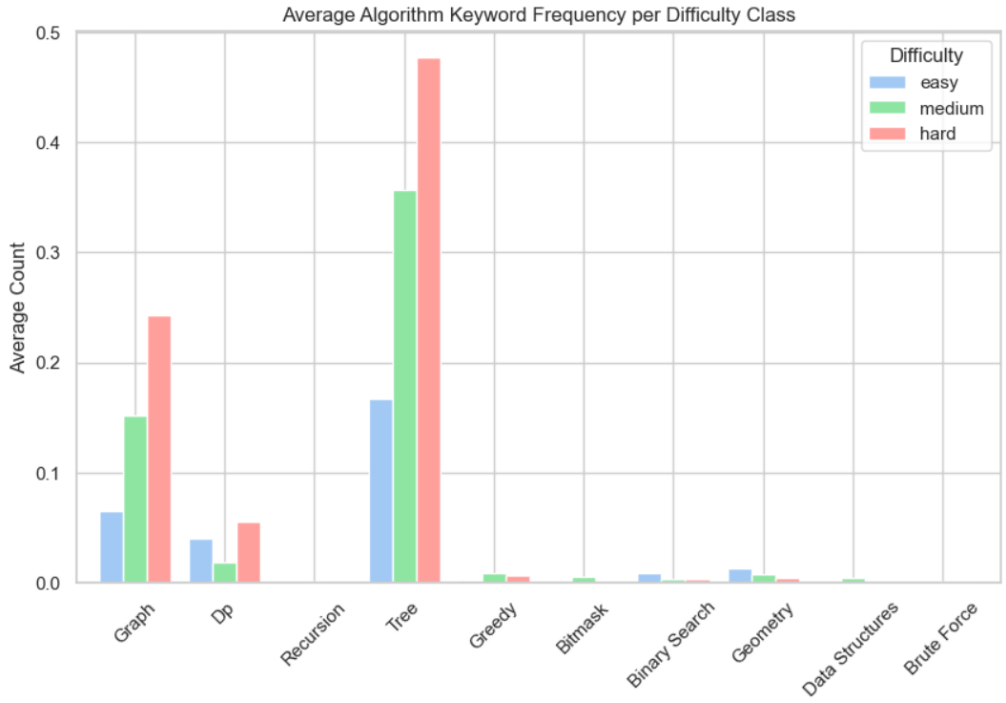
4.1 Text Length Analysis

Analysis showed that harder problems tend to have longer descriptions, indicating higher cognitive and informational load.



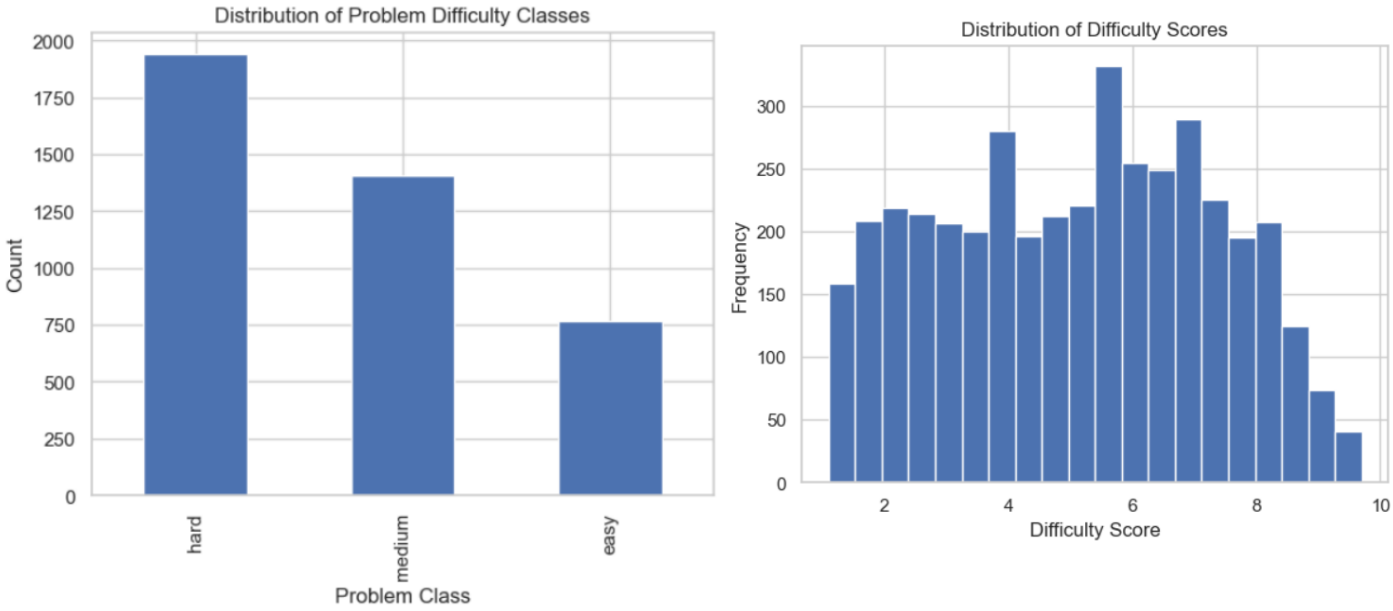
4.2 Algorithm Keyword Frequency

Algorithm-related keywords such as *graph*, *dynamic programming*, *recursion*, *tree*, and *greedy* appear more frequently in harder problems.



4.3 Difficulty Score Distribution

The difficulty score is continuously distributed rather than clustered around discrete values, supporting the need for regression modeling.



5. Feature Engineering

To capture both semantic and structural aspects of problem descriptions, the following features were engineered:

5.1 TF-IDF Features

Term Frequency–Inverse Document Frequency (TF-IDF) vectorization was applied using unigrams and bigrams. This captures both word importance and contextual phrases in the text.

5.2 Handcrafted Linguistic Features

Additional features were introduced to enhance interpretability:

- Text length (number of characters or words).
- Mathematical symbol count.
- Algorithm keyword frequencies.

All features were combined using sparse matrix concatenation to form the final feature representation for both classification and regression tasks.

6. Experimental Setup

- **Train–Test Split:** 80% training, 20% testing.
- **Feature Set:** Same engineered features used across all models.
- **Evaluation:** Performed on unseen test data.
- **Models:** Linear models chosen for interpretability and efficiency on high-dimensional sparse data.

This setup ensures fair comparison and reproducibility of results.

7. Classification Models and Results

7.1 Models Evaluated

- **Logistic Regression**
- **Balanced Logistic Regression**
- **Linear Support Vector Machine (Linear SVM)**

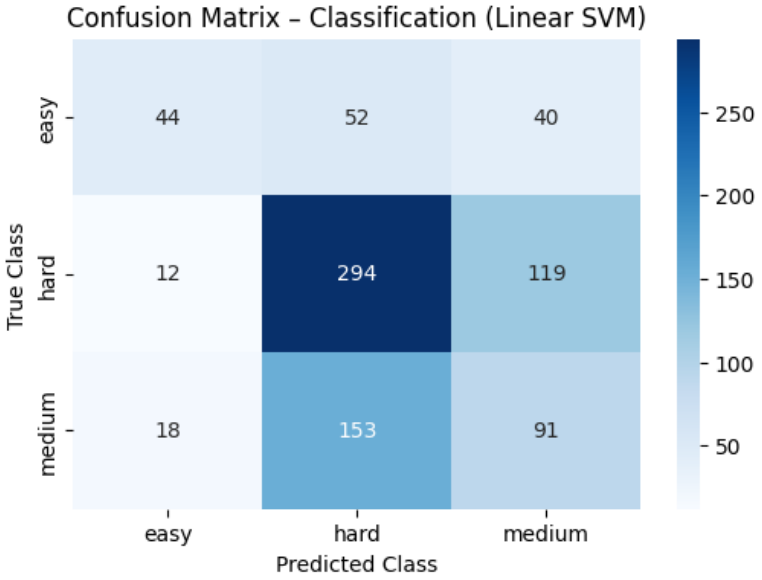
7.2 Model Comparison

Model	Accuracy	Macro F1-score
Logistic Regression	0.516	0.227
Balanced Logistic Regression	0.518	0.232
Linear SVM (Final)	0.521	0.470

Linear SVM was selected due to its superior balanced performance.

7.3 Final Classification Performance

- **Accuracy: 0.5577**
- **Macro F1-score: 0.50**



8. Regression Model and Results

8.1 Objective

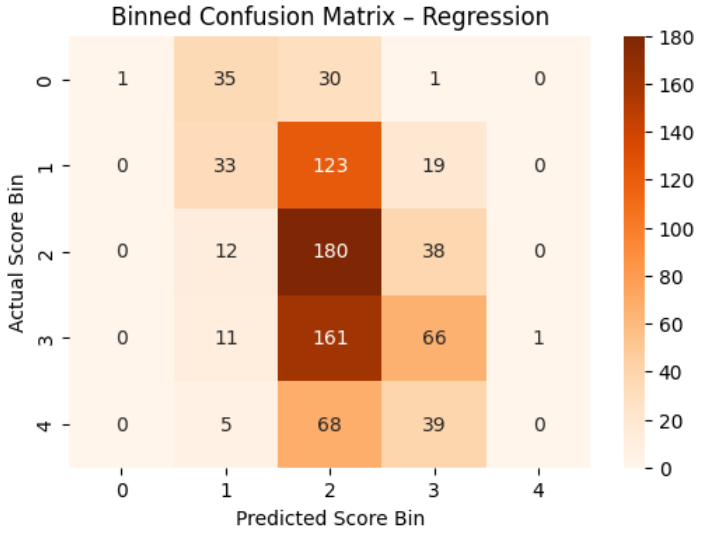
Estimate a continuous difficulty score to provide fine-grained difficulty assessment.

8.2 Model Used

- Ridge Regression

8.3 Performance Metrics

- **MAE:** 1.65
- **RMSE:** 1.98



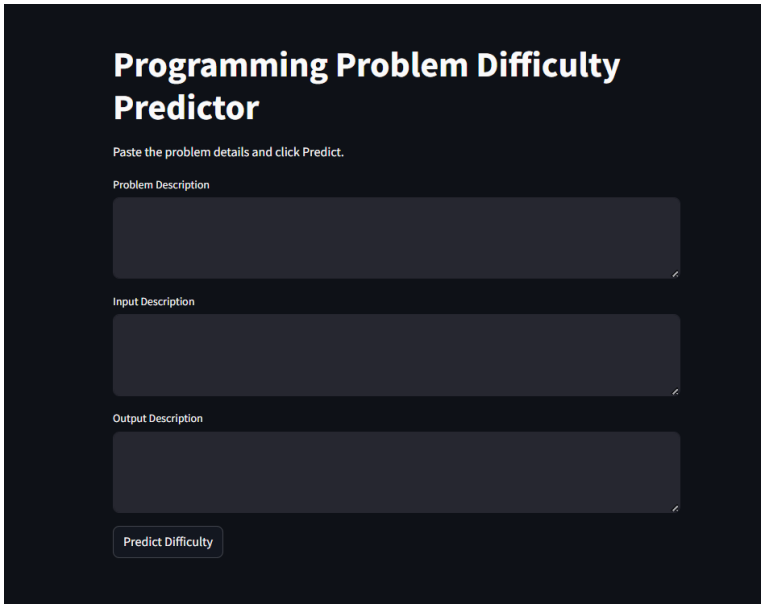
The regression model complements classification by enabling ranking and comparison of problems within the same difficulty class.

9. Web Interface Implementation

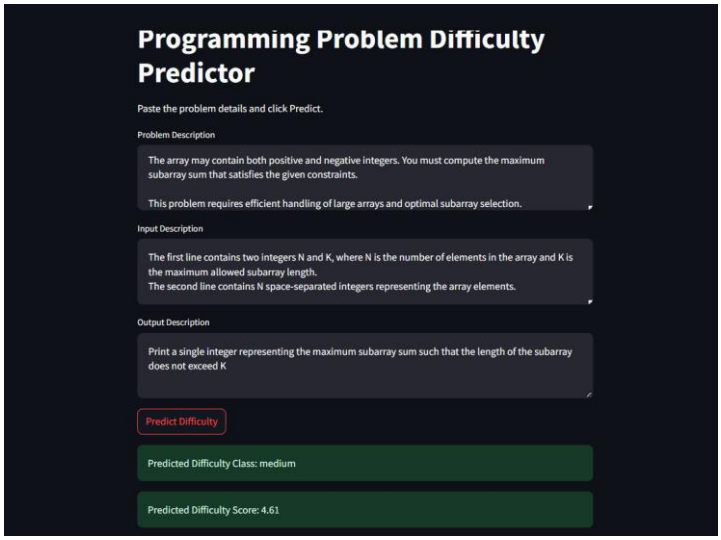
A web-based interface was developed using **Streamlit** to demonstrate the AutoJudge system.

Features

- Text input for problem description, input, and output.
- Displays predicted difficulty class and numerical score.
- Runs locally without deployment.



With problem attached



10. Conclusion

This project demonstrates that automated difficulty prediction for programming problems is feasible using text-based machine learning techniques. Despite the subjective nature of difficulty labeling, the AutoJudge system captures meaningful patterns using TF-IDF and linguistic features. The combination of classification and regression provides both interpretability and fine-grained analysis, making the system suitable for educational platforms and coding environments.

THANKING YOU,
Name- Jampa Sri Pramodhitha
Enroll No.- 23113075