AutoJudge

AI-Powered Programming Problem Difficulty Analyzer

Submitted by: Sanchita Gupta
Enrollment No.: 23119040

Abstract

Programming platforms often struggle to accurately assess the difficulty of coding problems due to the subjective and time-consuming nature of manual labeling. **AutoJudge** addresses this challenge by using machine learning to automatically predict problem difficulty from textual descriptions.

The system classifies problems into **Easy, Medium, and Hard** categories and also predicts a continuous difficulty score using **TF-IDF features** and **structural indicators**. Logistic Regression is used for classification, while Ridge Regression predicts the difficulty score. A web interface built with **FastAPI, HTML, CSS, and JavaScript** enables real-time interaction.

Experimental results show that AutoJudge achieves about **51% classification accuracy** and a **mean absolute error of 1.7** for difficulty score prediction, demonstrating the effectiveness of the proposed approach.

---

1. Introduction

1.1 Objectives

The objectives of this project are:

1. To develop a system that automatically predicts:
    - Difficulty class: Easy / Medium / Hard
    - Difficulty score: Continuous numeric value
2. To design an explainable system that highlights what makes a problem difficult
3. To build a user-friendly web interface
4. To ensure the system runs locally without dependency on external services

---

2. Dataset Description

2.1 Dataset Source

The dataset is stored in: data/raw/problems_data.jsonl

It contains programming problems with the following fields:

- Problem description
- Input description
- Output description
- Difficulty label
- Difficulty score

After preprocessing, the cleaned dataset is saved as :
data/processed/problems_clean.csv

2.2 Dataset Characteristics

Attribute	Description
Total problems	~1000+
Difficulty classes	Easy, Medium, Hard
Features	Textual + structural
Labels	Categorical + numerical

---

3. Data Preprocessing

3.1 Preprocessing Steps

1. Conversion to lowercase
2. Removal of punctuation and special characters
3. Stopword elimination
4. Token normalization
5. Cleaning irrelevant symbols
6. Handling missing values

The cleaned dataset is saved and reused for both training and inference.

---

4. Feature Engineering

4.1 Textual Features

Text data is transformed using **TF-IDF vectorization**. The following fields are used:

- Problem description
- Input description
- Output description

TF-IDF captures:

- Important keywords
- Term frequency relevance
- Document-level importance

4.2 Structural Features

Structural indicators are extracted such as:

- Length of description
- Presence of algorithmic keywords
- Constraint complexity
- Indicators like recursion, graph, dp, etc.

4.3 Final Feature Vector

The final representation is created by concatenating:

Final Features = TF-IDF vectors + Structural features

This hybrid approach improves the model’s understanding of both language and complexity patterns.

---

5. Models Used

5.1 Classification Model

Objective: Predict difficulty class
Algorithms tested:

- Logistic Regression
- Support Vector Machine

Final model: Logistic Regression
Reason:

- Better F1-score
- Faster convergence
- Easier interpretability

5.2 Regression Model

Objective: Predict numeric difficulty score
Algorithms tested:

- Ridge Regression
- Random Forest Regressor

Final model: Ridge Regression
Reason:

- Lower MAE
- Better generalization
- Stability on sparse features

---

6. Experimental Setup

6.1 Tools & Environment

- Python 3.12
- Libraries:
  - pandas
  - numpy
  - scikit-learn
  - fastapi
  - joblib

6.2 Training Configuration

- Train-test split: 80% / 20%
- Stratified sampling for classification
- Separate pipelines for classification and regression

---

7. Results and Evaluation

7.1 Classification Results

Metric	Value
Accuracy	0.501
F1-Score	0.490

Confusion Matrix :

Actual \ Predicted	Easy	Medium	Hard
Easy	93	26	34
Medium	60	221	108
Hard	54	129	98

The confusion matrix shows that:

- Medium class is predicted more accurately
- Some confusion exists between Medium and Hard
- Easy problems occasionally overlap with Medium

7.2 Regression Results

Metric	Value
MAE	1.696
RMSE	2.023
R^2	0.157

The regression model provides reasonable score estimates, though improvements are possible with more data and advanced embeddings.

---

8. Web Interface

8.1 Architecture

The system follows a client-server architecture:

User → Web UI → FastAPI Server → ML Models → Prediction → UI Display

8.2 Interface Features

The web interface allows users to:

- Enter problem description
- Enter input description
- Enter output description
- Click **Analyze Difficulty**

The system displays:

- Difficulty badge
- Predicted score
- Confidence bar
- Explanation highlights
- Similar problems

8.3 Technology Stack

Layer	Technology
Backend	FastAPI
Frontend	HTML, CSS, JavaScript
ML Models	scikit-learn
Storage	Joblib models

---

9. Sample Predictions

Example 1 — Easy Problem

Problem Description
Add two integers and print their sum.

Input Description
Add two integers and print their sum.

Output Description
Print a + b.

System Output

- **Difficulty:** EASY
- **Predicted Score:** 2.25
- **Confidence:** 82%

What makes this hard?
Problem appears straightforward

Similar Problems

- Add Two Numbers — Difficulty: Easy • Score: 1.2
- Two-sum — Difficulty: Easy • Score: 1.3
- N-sum — Difficulty: Easy • Score: 1.3

Example 2 — Medium Problem

Problem Description
Find shortest paths in a graph that may contain negative edge weights.

Input Description
Edges may contain negative weights.

Output Description
Edges may contain negative weights.

System Output

- **Difficulty:** MEDIUM
- **Predicted Score:** 3.81
- **Confidence:** 41%

What makes this hard?
shortest path, graph

Similar Problems

- Red/Blue Spanning Tree — Difficulty: Medium • Score: 5.3
- Kth Subtree — Difficulty: Hard • Score: 6.8
- Forest Construction — Difficulty: Hard • Score: 6.2

Example 3 — Hard Problem

Problem Description
Optimize range queries using segment trees with lazy propagation.

Input Description
You are given an array and multiple range update and query operations.

Output Description
Return results of all range queries efficiently.

System Output

- **Difficulty:** HARD
- **Predicted Score:** 4.91
- **Confidence:** 38%

What makes this hard?
lazy propagation, segment tree, optimize

Similar Problems

- Red/Blue Spanning Tree — Difficulty: Medium • Score: 5.3
- Kth Subtree — Difficulty: Hard • Score: 6.8
- Forest Construction — Difficulty: Hard • Score: 6.2

---

10. Future Scope

Potential improvements include:

- Using BERT or SBERT embeddings
- Applying ensemble learning
- Adding analytics dashboard to UI
