

AutoJudge: Predicting Programming Problem Difficulty

1. Abstract

Online competitive programming platforms categorize problems into difficulty levels such as *Easy*, *Medium*, and *Hard*, along with numerical difficulty scores. These labels are typically assigned through human judgment and user feedback, which can be subjective and inconsistent.

This project proposes **AutoJudge**, a machine learning system that automatically predicts both the **difficulty class** and **difficulty score** of programming problems using only textual descriptions.

The system combines **NLP-based feature extraction** with classical machine learning models and provides predictions through an interactive web interface.

2. Introduction

Competitive programming platforms host thousands of problems with varying levels of difficulty. Accurate difficulty estimation is important for:

- learner progression
- contest design
- recommendation systems

Manual labeling does not scale well and may introduce bias.

The goal of this project is to design an **automated, data-driven approach** that infers difficulty using only the problem text, without relying on metadata such as user submissions or acceptance rates.

3. Dataset Description

Each data sample represents a programming problem and contains the following fields:

- `title`

- `description`
- `input_description`
- `output_description`
- `problem_class` (Easy / Medium / Hard)
- `problem_score` (continuous numerical value)

Some text fields contain missing values, which are handled during preprocessing.

4. Data Preprocessing

The preprocessing pipeline consists of the following steps:

1. Missing values in text fields are replaced with empty strings
2. All textual fields are merged into a single column (`combined_text`)
3. Text is converted to lowercase
4. Special characters are removed while **preserving mathematical symbols**

This ensures that the complete problem context is captured while maintaining information relevant to algorithmic complexity.

5. Feature Engineering

Two complementary feature sets are used:

5.1 TF-IDF Features

- Unigrams and bigrams
- Capture semantic information from problem text
- High-dimensional sparse representation

5.2 Hand-crafted Structural Features

- Text length
- Word count
- Mathematical symbol count
- Algorithmic keyword frequencies (e.g., `dp`, `graph`, `dfs`, `bfs`, `recursion`)

The final representation combines semantic and structural signals to better model problem complexity.

6. Methodology

The task is formulated as **two independent machine learning problems**:

6.1 Difficulty Classification

Predicts one of:

- Easy
- Medium
- Hard

6.2 Difficulty Score Regression

Predicts a continuous difficulty score.

Both tasks use the same feature extraction pipeline.

7. Models Evaluated

7.1 Classification Models

- Dummy baseline (always predicts “Hard”)
- Logistic Regression

- Random Forest Classifier
- **Linear Support Vector Machine (SVM)**

7.2 Regression Models

- Mean baseline predictor
 - Linear Regression
 - Random Forest Regressor
 - **Gradient Boosting Regressor**
-

8. Evaluation Metrics

Classification

- Accuracy
- Precision, Recall, F1-score
- Confusion Matrix

	precision	recall	f1-score	support
easy	0.30	0.71	0.42	153
hard	0.56	0.37	0.44	389
medium	0.33	0.25	0.28	281
accuracy			0.39	823
macro avg	0.40	0.44	0.38	823
weighted avg	0.43	0.39	0.38	823

Regression

- Mean Absolute Error (MAE)

- Root Mean Squared Error (RMSE)

```
Accuracy: 0.5164034021871203
[[ 31  98  24]
 [ 7 365  17]
 [ 13 239  29]]
      precision    recall   f1-score   support
  easy          0.61      0.20      0.30      153
  hard          0.52      0.94      0.67      389
 medium         0.41      0.10      0.17      281

   accuracy           0.52      823
  macro avg       0.51      0.41      0.38      823
weighted avg     0.50      0.52      0.43      823
```

```
Accuracy: 0.40097205346294046
[[ 92  40  21]
 [121 188  80]
 [102 129  50]]
      precision    recall   f1-score   support
  easy          0.29      0.60      0.39      153
  hard          0.53      0.48      0.50      389
 medium         0.33      0.18      0.23      281

   accuracy           0.40      823
  macro avg       0.38      0.42      0.38      823
weighted avg     0.42      0.40      0.39      823
```

9. Experimental Results

9.1 Classification Results

Model	Accuracy
Dummy Baseline (Always Hard)	~0.47
Logistic Regression	~0.41
Random Forest	~0.40
Linear SVM (Final)	~0.50+

The Linear SVM outperformed other models due to its suitability for high-dimensional sparse TF-IDF features.

9.2 Regression Results

Model	MAE (Lower is Better)
Mean Baseline	~1.3
Linear Regression	~1.9
Random Forest Regressor	~1.7
Gradient Boosting Regressor (Final)	~1.6

Gradient Boosting achieved the lowest error by modeling non-linear relationships between textual complexity and difficulty.

10. Error Analysis

- Medium difficulty problems are the hardest to classify due to overlap with both Easy and Hard problems
 - Structural features improve model robustness
 - Classification accuracy is bounded by inherent ambiguity in textual descriptions
-

11. Web Interface

A user-facing web application was developed using **Gradio**.

The interface allows users to:

- Paste problem description, input, and output
- Instantly receive predicted difficulty class and score

This demonstrates real-time usability of the trained models.

AutoJudge – Programming Problem Difficulty Predictor

Paste a problem statement to get difficulty class and score

Problem Description

"The construction of office buildings has become a very standardized task. Pre-fabricated modules are combined according to the customer's needs, shipped from a faraway factory, and assembled on the construction site. However, there are still some tasks that require careful planning, one example being the routing of pipes for the heating system. A modern office building is made up of square modules, one on each floor being a service module from which (among other things) hot water is pumped out to the other modules through the heating pipes. Each module (including the service module) will have heating pipes connecting it to exactly two of its two to four neighboring modules. Thus, the pipes have to run in a circuit, from the service module, visiting each module exactly once, before finally returning to the service module. Due to different properties of the modules, having pipes connecting a pair of adjacent modules comes at different costs. For example, some modules are separated by thick walls, increasing the cost of laying pipes. Your task is to, given a description of a floor of an office building, decide the cheapest way to route the heating pipes."

Input Description

"The first line of input contains a single Integer n , stating the number of floors to handle. Then follow n floor descriptions, each beginning on a new line with two integers, $s_1 \leq n \leq 10$ and $s_2 \leq c \leq 10$, defining the size of the floor $s_1 \times s_2$ ($s_1 \times s_2 \leq 10$) times c modules. Beginning with the next line follows a floor description in ASCII format, in total $s_2 + 1$ rows, each with $s_2 + 2$ characters (count includes the final newline). All floors are perfectly rectangular, and will always have an even number of modules. All interior walls are represented by numeric characters, '0' or '9', indicating the cost of routing pipes through the wall (see sample input)."

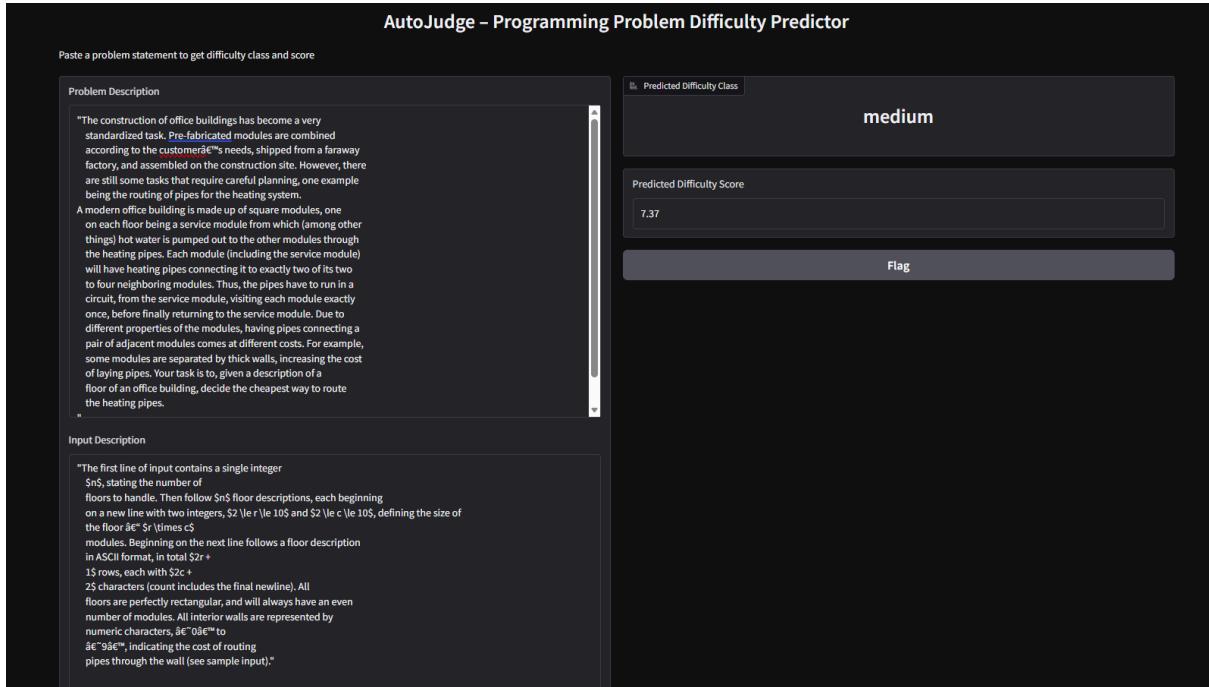
Predicted Difficulty Class

medium

Predicted Difficulty Score

7.37

Flag



AutoJudge – Programming Problem Difficulty Predictor

Paste a problem statement to get difficulty class and score

Problem Description

"In the menu to the right, you are able to download an English dictionary (dict.txt) containing a sorted list of words, one per line. Your task is to write a program that outputs a prefix of the dictionary that is as long as possible."

Input Description

no description

Output Description

"Output a number of lines, each containing a word from the dictionary. The words must form a prefix of the entire file."

Clear

Submit

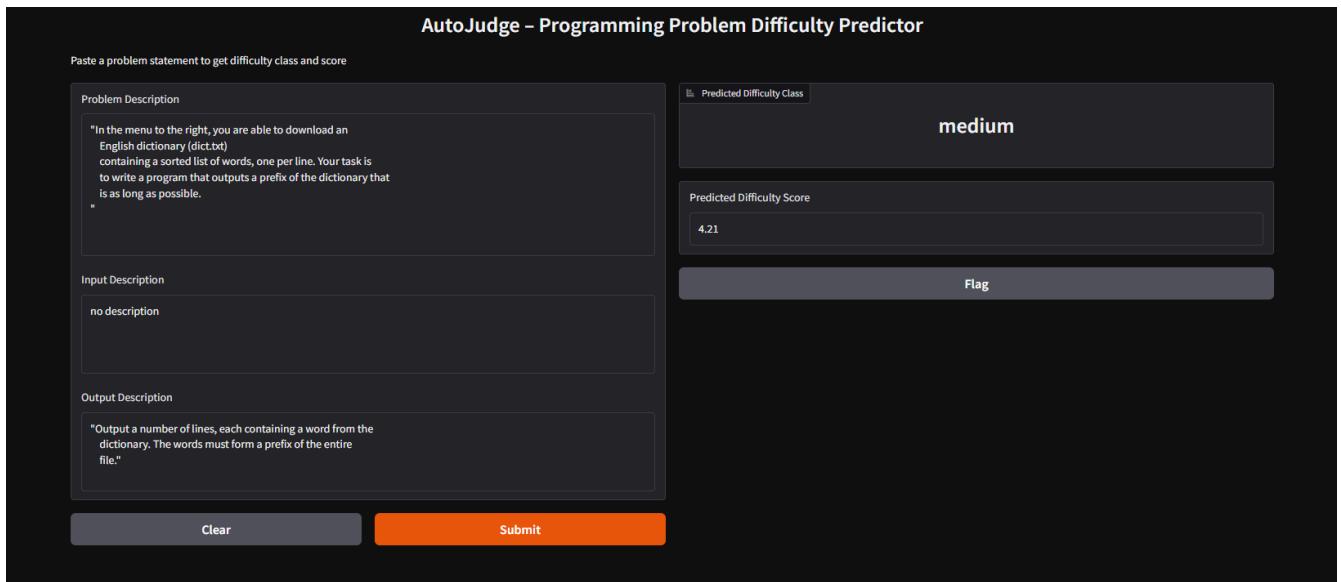
Predicted Difficulty Class

medium

Predicted Difficulty Score

4.21

Flag



12. Conclusion

This project demonstrates that programming problem difficulty can be reasonably predicted using only textual information.

By combining NLP techniques with classical machine learning models, AutoJudge achieves strong baseline performance and practical applicability.