

# AutoJudge: Predicting Programming Problem Difficulty Using NLP and Machine Learning

---

## 1. Introduction

Competitive programming platforms host a large number of programming problems that are typically categorised by difficulty levels such as Easy, Medium, and Hard. These difficulty labels and numerical scores are generally assigned through manual curation and user feedback, which makes the process subjective and dependent on user participation.

The objective of this project is to develop **AutoJudge**, an automated system that predicts the difficulty of a programming problem using only its textual description. The system performs two predictive tasks: classification of problems into Easy, Medium, or Hard, and regression to estimate a numerical difficulty score. The project demonstrates how classical Natural Language Processing (NLP) techniques combined with machine learning models can approximate problem difficulty without relying on solution code or user statistics.

---

## 2. Dataset Description

The dataset used in this project consists of approximately 4,100 programming problems collected from online competitive programming platforms. Each problem entry contains multiple textual fields describing the problem as well as pre-assigned difficulty labels. The main attributes in the dataset include the problem title, detailed problem description, input specification, output specification, sample input-output examples, difficulty class, and a numerical difficulty score.

The dataset is pre-labelled and was provided for use in this project. No manual annotation or labelling was performed.

---

## 3. Data Preprocessing

Before feature extraction, the dataset underwent several preprocessing steps to ensure consistency and robustness. Missing textual fields were handled by replacing null values with empty strings. This prevented errors during text processing and ensured uniform input across all samples.

The title, problem description, input description, and output description were concatenated into a single textual field referred to as `combined_text`. This allowed the model to learn from the complete problem statement rather than isolated components. Finally, all text was converted to lowercase to avoid duplication of features caused by case sensitivity.

---

## 4. Feature Engineering and NLP Techniques

### 4.1 TF-IDF Representation

To convert textual data into numerical form, the **TF-IDF (Term Frequency–Inverse Document Frequency)** technique was employed. TF-IDF captures the importance of words relative to the entire dataset by penalizing frequently occurring words and emphasizing rare but informative terms.

The vectorizer was limited to the top **5,000 features**, and English stop words were removed to reduce noise. This representation forms the core semantic feature set used by both classification and regression models.

---

### 4.2 Engineered Features

In addition to TF-IDF features, several manually engineered features were introduced to capture structural and algorithmic complexity that may not be fully represented by word frequencies alone.

Text length was used as a basic indicator of problem complexity, as harder problems often require longer explanations. Mathematical symbol density was computed to measure the proportion of mathematical and arithmetic symbols in the problem statement, reflecting mathematically intensive problems. Algorithm keyword frequency was calculated by counting occurrences of terms such as graph, dp, tree, bfs, and dfs, which are commonly associated with higher difficulty problems.

Constraint-related features were also engineered. These include the number of large numerical constraints mentioned in the problem, the maximum constraint value, and the frequency of constraint-related words such as constraint, bound, and limit. Additionally, the length of sample input-output examples was used as a proxy for structural complexity.

Redundant and highly correlated features were deliberately removed during feature refinement to improve model stability and interpretability.

---

## 5. Machine Learning Models

### 5.1 Classification Model

For difficulty classification, **Logistic Regression** was used. This model is well-suited for high-dimensional sparse feature spaces such as TF-IDF representations. The classifier predicts one of three difficulty classes: Easy, Medium, or Hard.

## 5.2 Regression Model

For predicting numerical difficulty scores, **Ridge Regression** was employed. Ridge Regression introduces L2 regularisation, which helps prevent numerical instability and overfitting when dealing with high-dimensional feature vectors. This makes it more reliable than standard linear regression for this task.

---

## 6. Experimental Setup and Evaluation

The dataset was split into training and testing sets using an **80–20 split**, with a fixed random seed to ensure reproducibility.

### 6.1 Classification Results

The classification model achieved an accuracy of approximately **51%**. Given the subjective nature of difficulty labelling and overlap between Medium and Hard problems, this performance is considered reasonable.

A confusion matrix was used to analyse misclassifications, revealing that most errors occurred between adjacent difficulty levels.

```
Confusion Matrix:  
[[ 46  64  43]  
 [ 22 290  77]  
 [ 21 177  83]]
```

### 6.2 Regression Results

The regression model achieved a **Mean Absolute Error (MAE)** of approximately **1.7** and a **Root Mean Squared Error (RMSE)** of approximately **2.1**. These results indicate that the predicted difficulty scores are generally close to the true values.

---

## 7. Web Interface Implementation

A web interface was developed using **Streamlit** to allow users to interact with the system. The interface provides text input fields for the problem description, input format, and output format. Upon submission, the trained models generate predictions in real time.

The web interface serves as a practical demonstration of how the system can be used by problem setters or platform administrators.

Deploy

## AutoJudge: Problem Difficulty Predictor

Paste a programming problem description to predict its difficulty.

Problem Description

Input Description

Output Description

Predict

---

## 8. Sample Predictions and Observations

The system was tested on problems of varying complexity. Simple loop-based problems were consistently predicted as Easy with low difficulty scores. Problems involving dynamic programming, graphs, or heavy constraints were predicted as Medium or Hard with higher scores. These observations align well with expected difficulty levels.

---

## 9. Conclusion and Future Work

This project demonstrates that programming problem difficulty can be estimated using classical NLP techniques combined with machine learning models. Although difficulty prediction is inherently subjective, AutoJudge is able to capture relative complexity trends using textual information alone.

Future improvements could include the use of word embeddings or transformer-based models, explicit modelling of time and memory constraints, and continuous refinement using user feedback.

---

## Author Details

**Name:** Anmol Yadav

**Program:** B.Tech

**Department:** Electrical Engineering

**Year:** 2nd Year

**Enrollment Number:** 24117016

---