

# AutoJudge

## Predicting Programming Problem Difficulty

**Name:** Anurag Kishor Patil

**Enrollment No.:** 24114015

**Department:** CSE

### 1. Problem Statement

Competitive programming platforms host a large number of problems that are manually labeled into difficulty levels such as easy, medium, and hard. This manual labeling process is subjective, time-consuming, and often inconsistent across platforms. The objective of this project is to design an automatic system that predicts the difficulty of a programming problem based solely on its textual description and related metadata.

The task is formulated in two complementary ways:

- **Regression:** Predict a continuous difficulty score representing the relative hardness of a problem.
- **Classification:** Predict a discrete difficulty label (easy / medium / hard) using the learned representations.

The final system integrates machine learning models with a web-based interface to allow real-time predictions.

### 2. Dataset Description

Two datasets were used to train the models.

#### 1. Kattis Problems Dataset

- Link: [data\\_kt](#)
- Number of Entries: 4112
- Columns:

#	Column	Non-Null Count	Dtype
0	title	4112 non-null	object
1	description	4112 non-null	object
2	input_description	4112 non-null	object
3	output_description	4112 non-null	object
4	sample_io	4112 non-null	object
5	problem_class	4112 non-null	object
6	problem_score	4112 non-null	float64
7	url	4112 non-null	object

## 2. Codeforces Problems Dataset

- Link: [data\\_cf](#)
- Number of Entries: 10024
- Irrelevant columns (like contest\_id, contest\_name, tags, time\_limit etc) were removed. Only the following columns were kept.

#	Column	Non-Null Count	Dtype
0	title	10024 non-null	object
1	description	9981 non-null	object
2	input_format	9862 non-null	object
3	output_format	9742 non-null	object
4	examples	9895 non-null	object
5	rating	9767 non-null	float64

## 3. Data Preprocessing

Data preprocessing is critical for ensuring meaningful feature extraction. The following steps were applied consistently during training and inference:

1. For the Codeforces dataset, the rows with null values in the rating column are dropped. In the remaining columns, null fields are replaced with empty strings.
2. Normalization of ‘rating’ column (in Codeforces dataset) and ‘problem\_score’ column (in Kattis dataset)
3. The Codeforces dataset does not provide explicit difficulty class labels. Therefore, difficulty classes for Codeforces problems were derived by analyzing the relationship between problem scores and difficulty classes in the Kattis dataset. Based on the observed score distributions, score thresholds were

defined to map normalized problem scores to the classes easy, medium, and hard.

4. Conversion to strings of 'examples' column (in Codeforces dataset) and 'sample\_io' column (in Kattis dataset)
5. Aggregation of all textual fields into a single string to represent the complete problem context:  
Combined Text = Title + Description + Input + Output + Sample\_io/examples
6. Text Cleaning through removal of extra whitespace and lowercasing
7. Concatenation of both the datasets into a single dataframe. Final dataframe:

#	Column	Non-Null Count	Dtype
---	---	-----	-----
0	title	13879 non-null	object
1	description	13879 non-null	object
2	input_description	13879 non-null	object
3	output_description	13879 non-null	object
4	sample_io	13879 non-null	object
5	problem_class	13879 non-null	object
6	problem_score	13879 non-null	float64
7	sample_io_text	13879 non-null	object
8	score_norm	13879 non-null	float64
9	combined_text	13879 non-null	object

## 4. Feature Engineering

The system uses a hybrid feature representation combining **textual** and **numeric** features.

### 4.1 TF-IDF Features

- TF-IDF vectorization is applied to the combined problem text
- Captures keyword importance and semantic structure
- Produces a high-dimensional sparse feature matrix

### 4.2 Numeric Features

Four handcrafted numeric features were engineered:

1. **Constraint Magnitude Feature:** Extracts numerical constraints such as  $10^5$ ,  $10^9$  and converts it to a logarithmic scale.

2. **Mathematical Density:** Counts LaTeX expressions, operators, and mathematical symbols
3. **Algorithmic Keyword Count:** Counts occurrences of algorithm-related terms such as dp, graph, dfs, binary search, etc.
4. **Text Length:** Character length of the main problem description

## 5. Models Used and Results

### 5.1 Regression Models (Difficulty Score Prediction)

The following regression models were evaluated:

Evaluation	Ridge Regression	Random Forest Regressor	XGBoost Regressor
MAE	0.1750	0.1777	0.1708
R2	0.3311	0.3216	0.3659

As observed from the table, the model with least MAE and highest R2 value is XGBoost Regressor. Hence it was used for predicting difficulty scores.

### 5.2 Classification Models (Difficulty Class Prediction)

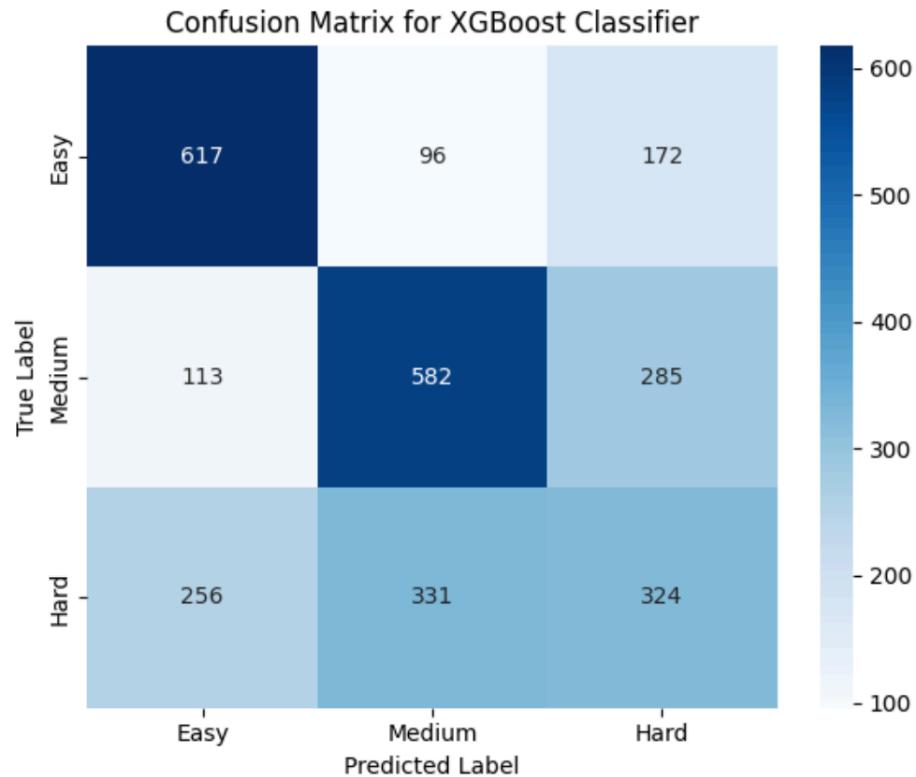
The following classification models were evaluated:

Evaluation	Logistic Regression	SVM	XGBoost Classifier
Accuracy	0.5345	0.5075	0.5486
F1	0.5295	0.5039	0.5426
Precision	0.5262	0.5012	0.5391
Recall	0.5347	0.5075	0.5489

The XGBoost classifier achieves the highest overall accuracy (0.5486) and macro-averaged F1 score (0.5426), indicating improved class balance and better

modeling of non-linear relationships in the feature space. Consequently, XGBoost was selected as the final classification model for deployment.

Confusion Matrix of XGBoost Classifier:



## 6. Experimental Setup

- Train–test split: 80% training, 20% testing
- Feature pipeline: TF-IDF + numeric features (stacked)
- Evaluation metrics:
  - Regression: MAE, R<sup>2</sup>
  - Classification: Accuracy, Precision, Recall, F1-score, Confusion Matrix

All experiments were conducted using the same preprocessing and feature extraction pipeline to avoid train–serve skew.

## 8. Web Interface And Browser Extension

## 1. Web UI:

To demonstrate real-time difficulty prediction in a practical setting, the system was integrated with both a **web-based interface** and a **browser extension**. This allows users to obtain predictions either by manually entering problem details or by directly analyzing problems on competitive programming platforms.

- **Architecture**

- Backend: Flask REST API
- Frontend: HTML + JavaScript
- Models loaded using joblib

- **Functionality**

- User inputs problem details via a form
- Backend extracts features and runs ML models
- Output displayed: Predicted difficulty class and Difficulty score

## 2. Browser Extension

It was observed that copying and pasting the problem can be tedious. So tan extension was added for Codeforces website which predicted problem class without manual input.

- **Architecture**

- Content Script: DOM extraction from problem webpages
- Frontend: HTML + JavaScript (Extension popup)
- Backend Communication: REST API calls to Flask server

- **Functionality**

- Automatically extracts problem details from the active webpage
- Sends extracted data to the Flask /predict API endpoint
- Receives prediction results from the backend
- Output displayed in extension popup

## 3. Sample Prediction

- **Web UI:**

Rating of problem shown in the web ui screenshot = 800  
Predicted Result = Easy

# Problem Difficulty Predictor

Paste a programming problem to estimate its difficulty using ML.

## Problem Basics

---

**Problem Title**

e.g., Two Sum, Longest Substring...

## Problem Details

---

**Input Description**

Describe the input format and constraints...

**Output Description**

Describe the expected output...

**Problem Description**

Full problem statement...

**Sample Input/Output**

Examples of input and output...

**Predict Difficulty**

127.0.0.1:5000

# Problem Difficulty Predictor

Paste a programming problem to estimate its difficulty using ML.

## Problem Basics

---

**Problem Title**

New World, New Me, New Array

## Problem Details

---

**Input Description**

(1≤t≤1000) — the number of test cases.  
The only line of each test case contains three integers n, k, p (1≤n≤50, -2500≤k≤2500, 1≤p≤50) — the length of the array, the required sum, and the boundary of the segment from which numbers can be replaced.

**Output Description**

For each test case, output the minimum number of operations to achieve the final sum k in the array, or -1 if it is impossible to achieve the sum k.

**Problem Description**

Natsume Akito has just woken up in a new world and immediately received his first quest! The system provided him with an array a of n zeros, an integer k, and an integer p. In one operation, Akito chooses two integers i and x such that 1≤i≤n and  $a[i] \neq 0$ , and performs the assignment  $a[i] \leftarrow x$ . Akito is still not fully accustomed to controlling his new body, so help him calculate the minimum number of operations

**Sample Input/Output**

```
0
21 100 10
9 -420 42
5 -7 2
13 37 7
```

**Predict Difficulty**

**Prediction Result**

Difficulty Level	Difficulty Score
EASY	2.7 / 100

- **Extension:**

Rating of problem shown in the web ui screenshot = 1400  
Predicted Result = Medium

**C2. The Cunning Seller (hard version)**

time limit per test: 2 seconds  
memory limit per test: 256 megabytes

This is the hard version of the problem. The easy version differs from the hard one in that it requires determining the minimum cost with the least number of deals, while the hard version requires determining the minimum cost with a limited number of deals.

After the cunning seller sold three watermelons instead of one, he decided to increase his profit — namely, he bought even more watermelons. Now he can sell  $3^x$  watermelons for  $3^{x+1} + x \cdot 3^{x-1}$  coins, where  $x$  is a non-negative integer. Such a sale is called a deal.

A calculating buyer came to him, but he has little time, so the buyer can make no more than  $k$  deals and plans to buy exactly  $n$  watermelons.

The buyer is in a hurry and has therefore turned to you to determine the minimum number of coins he must pay the seller for  $n$  watermelons if he makes no more than  $k$  deals. If it is impossible to buy exactly  $n$  watermelons while making no more than  $k$  deals, output  $-1$ .

**Input**  
The first line contains an integer  $t$  ( $1 \leq t \leq 10^4$ ) — the number of test cases. The description of each test case follows.

In a single line of each test case, there are two integers  $n$  and  $k$  ( $1 \leq n, k \leq 10^9$ ) — how many watermelons need to be bought and how many deals can be made.

**Output**  
For each test case, output a single integer — the minimum cost of the watermelons or  $-1$  if it is impossible to buy the watermelons while meeting all the conditions.

**Example**

input	copy
8 1 1 3 3 8 2 2 4 10 10 28 14	

**CF Difficulty Predictor**  
Analyze this Codeforces problem to estimate its difficulty

**Predict Difficulty**

**Virtual participation**  
Virtual contest is a way to take part in past contests, as close as possible to participation on time. It is supported only by ICPC mode for virtual contests. If you've seen these problems, a virtual contest is not for you - solve these problems in the archive. If you just want to have some practice before a real contest, a virtual contest is not for you - solve this problem in the archive. Never use someone else's code, read the tutorials or communicate with other person during a virtual contest.

**Start virtual contest**

**Clone Contest to Mashup**  
You can clone this contest to a mashup.

**Clone Contest**

**Submit?**  
Language: GNU G++23 14.2 (64 bit, ms)  
Choose file: Choose File No file chosen  
Submit

**C2. The Cunning Seller (hard version)**

time limit per test: 2 seconds  
memory limit per test: 256 megabytes

This is the hard version of the problem. The easy version differs from the hard one in that it requires determining the minimum cost with the least number of deals, while the hard version requires determining the minimum cost with a limited number of deals.

After the cunning seller sold three watermelons instead of one, he decided to increase his profit — namely, he bought even more watermelons. Now he can sell  $3^x$  watermelons for  $3^{x+1} + x \cdot 3^{x-1}$  coins, where  $x$  is a non-negative integer. Such a sale is called a deal.

A calculating buyer came to him, but he has little time, so the buyer can make no more than  $k$  deals and plans to buy exactly  $n$  watermelons.

The buyer is in a hurry and has therefore turned to you to determine the minimum number of coins he must pay the seller for  $n$  watermelons if he makes no more than  $k$  deals. If it is impossible to buy exactly  $n$  watermelons while making no more than  $k$  deals, output  $-1$ .

**Input**  
The first line contains an integer  $t$  ( $1 \leq t \leq 10^4$ ) — the number of test cases. The description of each test case follows.

In a single line of each test case, there are two integers  $n$  and  $k$  ( $1 \leq n, k \leq 10^9$ ) — how many watermelons need to be bought and how many deals can be made.

**Output**  
For each test case, output a single integer — the minimum cost of the watermelons or  $-1$  if it is impossible to buy the watermelons while meeting all the conditions.

**Example**

input	copy
8 1 1 3 3 8 2 2 4 10 10 28 14	

**CF Difficulty Predictor**  
Analyze this Codeforces problem to estimate its difficulty

**MEDIUM** Difficulty Score  
**3.20**

**Virtual participation**  
Virtual contest is a way to take part in past contests, as close as possible to participation on time. It is supported only by ICPC mode for virtual contests. If you've seen these problems, a virtual contest is not for you - solve these problems in the archive. If you just want to have some practice before a real contest, a virtual contest is not for you - solve this problem in the archive. Never use someone else's code, read the tutorials or communicate with other person during a virtual contest.

**Start virtual contest**

**Clone Contest to Mashup**  
You can clone this contest to a mashup.

**Clone Contest**

**Submit?**  
Language: GNU G++23 14.2 (64 bit, ms)  
Choose file: Choose File No file chosen  
Submit

## 9. Conclusion

This project demonstrates that text analysis can fairly predict how hard a problem is, using machine learning. Among all models tested, XGBoost did the best job against linear and tree-based models for both regression and classification tasks.

Key takeaways:

- Hybrid feature representations are crucial
- Text is best represented in models that handle sparse data
- The system generalizes well despite the subjective nature of difficulty labels

Future work may include:

- Add code submissions and editorial data
- Utilize transformer-based text representations.
- Extend to normalize difficulty across different platforms