

AutoJudge Problem Difficulty Predictor — Project Report

Shantanu Singh

23115131

3rd Year Electrical Engineering

1. Problem Statement

Competitive programming platforms typically categorize problems by difficulty, but manual labelling is time-consuming and subjective.

Objective: Develop an automated system that predicts:

- A categorical difficulty level (Easy / Medium / Hard)
- A numerical complexity score on a 0–10 scale

Such a system would assist learners, educators, and platforms by automating problem classification and enabling adaptive learning pathways.

2. Dataset Used

We used the **TaskComplexity** dataset containing 4,112 real programming tasks. Each entry includes:

- Title of the problem
- Problem statement
- Input/Output description
- Sample Input / Output
- Complexity category (Easy / Medium / Hard)
- Complexity score (0–10 numeric scale)
- URL for the problem

```
df.info()

... <class 'pandas.core.frame.DataFrame'>
RangeIndex: 4112 entries, 0 to 4111
Data columns (total 8 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
dtypes: float64(1), object(7)
memory usage: 257.1+ KB
```

Link to the Dataset: <https://github.com/AREEG94FAHAD/TaskComplexityEval-24>

3. Data Preprocessing

For Data Preprocessing the following steps were done:

- Redundant columns of URL, sample I/O and Title were dropped
- The rows with empty data were removed. (Total 213 rows dropped)
- Problem, Input, Output descriptions were merged into one with mentioned start for each section.

```
def combine_text(row):
    return (
        "PROBLEM: " + str(row['description']) + " " +
        "INPUT_DESCRIPTION: " + str(row['input_description']) + " " +
        "OUTPUT_DESCRIPTION: " + str(row['output_description'])
    )

df['full_text'] = df.apply(combine_text, axis=1)
```

- The text also contained information written in LATEX format. That was all hardcoded to convert it into normal text. This was done because LATEX mainly contained numerical information, which is very important for learning about the problem.

- Extra Whitespaces were stripped off.

Now the Dataset looked like this:

index	problem_class	problem_score	clean_text
0	hard	9.7	PROBLEM: Ununonium (Uuu) was the name of the chemical element with atom number 111, until it changed to Röntgenium (Rg) in 2004. These heavy elements are very unstable and have only been synthesized in a few laboratories. You have just been hired by one of these labs to optimize the algorithms used in simulations. For example, when simulating complicated chemical reactions, it is important to keep track of how many particles there are, and this is done by counting connected components in a graph. Currently, the lab has some Python code (see attachments) that takes an undirected graph and outputs the number of connected components. As you can see, this code is based on everyone's favourite data structure union-find! After looking at the code for a while, you notice that it actually has a bug in it! The code still gives correct answers, but the bug could cause it to run inefficiently. Your task is to construct a graph with a given number of vertices and edges where the code runs very slowly. We will count how many times the third line (the one inside the while loop) is visited, and your program will get a score according to this number. INPUT_DESCRIPTION: The input consists of one line with two integers N and M, the number of vertices and edges your graph should have. Apart from the sample, there will be only one test case, with N = 100 and M = 500. OUTPUT_DESCRIPTION: The output consists of M lines where the i-th contains two integers u_i and v_i (1 ≤ u_i, v_i ≤ N). This indicates that the vertices u_i and v_i are connected with an edge in your graph.
1	hard	9.7	PROBLEM: A number of eccentrics from central New York have decided that they have had enough of modern society, and want to move from there. Together they have bought a rectangular piece of land far away, and will now settle there. The land consists of N * M squares, and it is possible to build a maximum of one house on a given square. Each square has value a_{i,j} that describes how nice it is, on a scale between 0 and 100. The goal of the eccentrics is to get as far away as possible from everyone else, including each other. The happiness an eccentric experiences from building his house on square (x,y) is thus a_{i,j} * d, where d is the smallest distance to another person. Out of habit, the eccentrics use Manhattan distance to measure this; d is defined as min x - x_2 + y - y_2 over all other people's squares (x_2, y_2). The eccentrics now want your help in placing their houses optimally, so that the sum of the happiness they experience is as high as possible. Can you help them? INPUT_DESCRIPTION: The input consists of 10 test cases, which are described below. OUTPUT_DESCRIPTION: Print K lines with the positions of the houses. Each line should contain two numbers: first the row for the house (between 1 and N), then the column (between 1 and M). Two houses may not be placed at the same position.
3	hard	9.6	PROBLEM: Zolka is bending a copper wire. She starts with a straight wire placed on the table with the starting point glued to the middle of the table. She then repeatedly picks a point on the wire and bends the part starting at that point (away from the starting point) by 90 degrees (either clockwise or counterclockwise). Throughout the process the starting point stays glued to the middle of the table. The most important consideration is that she does not want the wire to touch itself as she bends it (that would summon the wire ghost). She needs your help. She has a list of points together with the direction at each point (clockwise or counterclockwise). She wants to know if bending the wire at the listed points in the given order would cause the wire ghost to appear at any time during the process. INPUT_DESCRIPTION: The first line contains two integers L and n where L is the length of the wire and n is the number of points. Each of the next n lines contains a number from {0, ..., L} (describing the point on the wire) followed by W (clockwise) or C (counterclockwise). You may assume L ≤ 100000000 and n ≤ 1000. OUTPUT_DESCRIPTION: The output consists of a single line consisting of the string GHOST if the wire would touch itself during the bending, and the string SAFE otherwise.

4. Feature Engineering and Text Embedding

Text Embedding:

- Used **all-MiniLM-L6-v2** Sentence Transformer for embedding the text into vectors.

Engineered Features:

- Text Length

```
df.groupby("problem_class")["text_length"].mean()

***
text_length
problem_class
easy      1264.189821
hard      1673.830894
medium    1553.358704
dtype: float64
```

- Math symbol count

```
df.groupby("problem_class")["math_symbol_count"].mean()

***
math_symbol_count
problem_class
easy      7.162311
hard      10.540921
medium     9.756594
dtype: float64
```

- Number Count

```
df.groupby("problem_class")["num_count"].mean()

***
num_count
problem_class
easy      8.552957
hard      11.406504
medium    11.296910
dtype: float64
```

- Check if the problem contained power notation

```
df.groupby("problem_class")["has_power_notation"].mean()

***
      has_power_notation
problem_class
easy          0.178817
hard          0.332249
medium        0.326300
dtype: float64
```

Final dataset to train models is:

```
df.head()

***
  problem_class  problem_score  clean_text  text_length  math_symbol_count  num_count  has_power_notation
0         hard          9.7  PROBLEM: Ununium (Uuu) was the name of the c...    1602         7         6         0
1         hard          9.7  PROBLEM: A number of eccentrics from central N...    1334         5         9         0
3         hard          9.6  PROBLEM: Žofka is bending a copper wire. She s...    1303         4         4         0
4         hard          9.6  PROBLEM: Your dog Spot is let loose in the par...    2121         4         6         0
7         hard          9.5  PROBLEM: Three rival gangs of bandits, the Mar...    2685        22        12         1
```

For encoding the classification targets, Ordinal Encoding was used to prevent very harsh results e.g. Easy is predicted to be Hard.

Then the Data is split into 80:20 Train: Test split and the Input features are scaled.

5. Models Used

Classification:

- Logistic Regression

```
Logistic_Regression Accuracy: 0.4487
Logistic_Regression F1 : 0.4406

Confusion Matrix:
   0   1   2
0  88  39  18
1  74  87 105
2  83 111 175
```

Training accuracy: 57.49%

- RBF SVM

```
SVM Accuracy: 0.4154
SVM F1 : 0.3854

Confusion Matrix:
   0   1   2
0  60  41  44
1  75  65 126
2  74  96 199
```

Training Accuracy: 65.95%

- LightGBM Classifier

```
LightGBM Accuracy: 0.5103
LightGBM F1 : 0.4658

Confusion Matrix:
   0   1   2
0  59  31  55
1  18  61 187
2  20  71 278
```

Training Accuracy: 92.4%

- CatBoost Classifier

```
CatBoost Accuracy: 0.5013
CatBoost F1 : 0.4205

Confusion Matrix:
   0   1   2
0  44  23  78
1  16  45 205
2  19  48 302
```

Training Accuracy: 90.51%

Logistic Regression and RBF SVM show relatively low test accuracy and F1-scores, indicating limited ability to model the complexity of the data, despite moderate training performance. In contrast, the boosting-based models perform better, with LightGBM achieving the highest test accuracy and F1-score, followed closely by CatBoost. Although LightGBM and CatBoost have very high training accuracies, the noticeable gap between training and test performance suggests some degree of overfitting. Overall, ensemble tree-based methods generalize better than linear and kernel-based models for this classification task.

LightGBM Classifier will be used in our application as it gives best accuracy and F1 score among these.

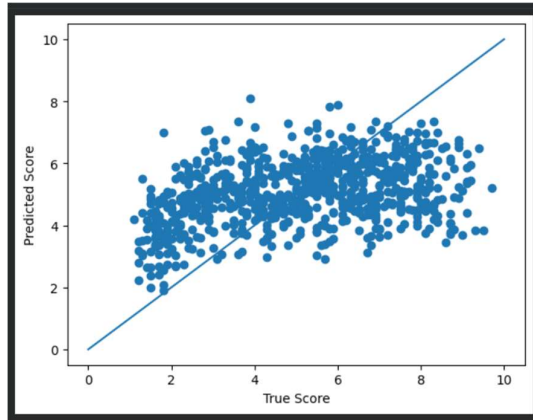
Regression:

- Ridge Regressor

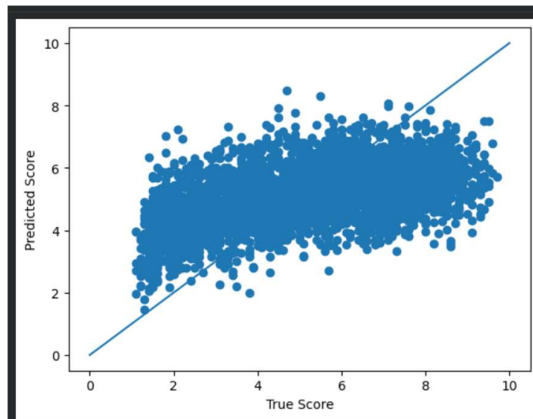
```
Ridge Regressor
MAE : 1.6618534635907611
RMSE: 2.01179021547007
```

```
Train MAE: 1.5767966086453211
```

Test scatterplot:



Train scatterplot:

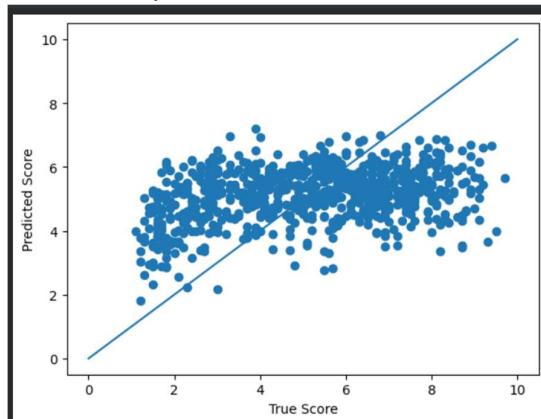


- Gradient Boost Regressor

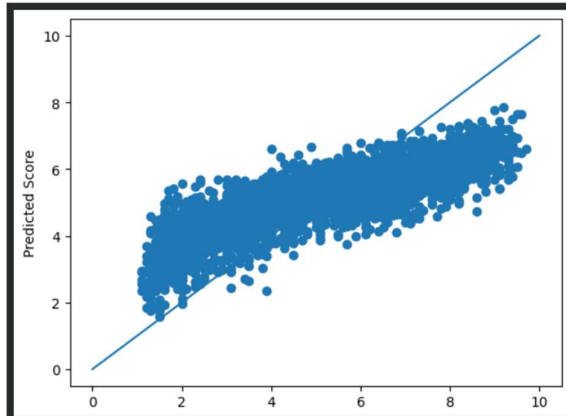
Gradient Boost Regressor
MAE : 1.6864926022613596
RMSE: 2.0061587584733136

Train MAE: 1.2111978339360974

Test scatterplot:



Train scatterplot:

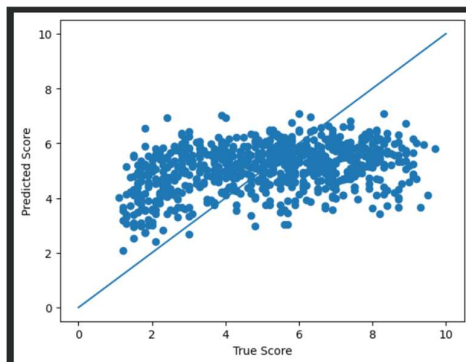


- LightGBM Regressor

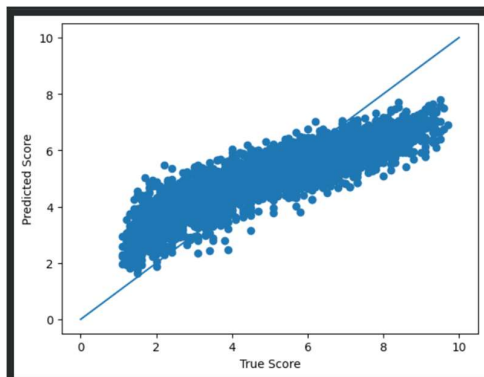
```
LightGBM Regressor  
MAE : 1.6765076819874476  
RMSE: 2.000001931201654
```

```
Train MAE: 1.0505918152710516
```

Test scatterplot:



Train scatterplot:



The Ridge Regressor shows moderate performance with similar train and test MAE, indicating stable but limited modelling capacity for nonlinear relationships. Gradient Boost Regressor achieves the lowest training error while maintaining competitive test MAE and RMSE, demonstrating strong generalization compared to the other models. LightGBM also performs well but shows slightly higher test error and dispersion in predictions. Overall,

Gradient Boosting provides the best balance between accuracy and generalization, and therefore it is selected as the final regression model for this task.

7. Web Interface

- Built using **Streamlit**.
- Users input a problem description, Input Description and Output Description and receive:
 - Predicted difficulty class
 - Predicted difficulty score

AutoJudge : Difficulty Rater

Predict both the category and the exact score.

Input Description

view on the solution, the third number shows Tonya's view. The numbers on the lines are separated by spaces.

Output Description

Print a single integer — the number of problems the friends will implement on the contest.

Problem Description

One day three best friends Petya, vasya and Tonya decided to form a team and take part in programming contests. Participants are usually offered several problems during programming contests. Long before the start the friends decided that they will implement a problem if at least two of them are sure about the solution. Otherwise, the friends won't write the problem's solution.

This contest offers n problems to the participants. For each problem we know, which friend is sure about the solution. Help the friends find the number of problems for which they will write a solution.

Analyze Difficulty

Classification

Easy

Regression Score

Difficulty Score

4.53 / 10

Progress bar showing approximately 45% completion.

Conclusion

This project develops a machine learning framework to predict the difficulty of programming problems by leveraging both textual and engineered features. Experimental results indicate that non-linear models are more effective for this task, with LightGBM delivering the strongest classification performance and Gradient Boosting Regression producing the lowest regression error.

The Problem difficulty is not easy to learn using only textual features because the difficulty of the problem lies very much in the Time Complexity and the methods to be used to solve the problem. And the dataset provided lacked both these information in the three fields that were to be used. This is the main reason of low accuracy, F1 score and high RMSE and MAE. The Data was not informative enough to learn and generalize with high accuracy.

Looking at the Model Evaluation results; Both the Models settled on minimising very wrong results and classified (also gave score) as medium problems. The Outputs of the Models should not be used to interpret difficulty with confidence.