# ACM-IITR

# AutoJudge Predicting Programming Problem Difficulty

**Laksh Alawadhi** **24113076**

**Problem Statement:**

It helps in predicting the difficulty of problem(primarily DSA/CP problem) with the difficulty score as well. The difficulty level is classified into three categories easy, medium and hard and score ranges from 1 to 10 and for prediction it uses

- Problem description
- Input description
- Output description

This has been made with the help of text preprocessing, feature extraction, testing and evaluating various models on the basis of their metrics and with the help of various scatterplots and heatmaps

**Dataset used:**

I used the dataset which was initially provided to us in JSON format but then I converted it into CSV format, and it has 4112 rows and 8 columns

- Title (short name of the problem)
- Description (full PS explaining the task to be solved)
- Input Description (expected input format)
- Output Description (expected output format)
- Sample input (example test cases to illustrate problem requirements)
- Problem class (represents categorical difficulty)
- Problem score (represents numerical difficulty)
- URL (link of the problem)

Description has 81 null values, input_description has 120 null values and output_description has 131 null values

It has 1 duplicate value (the last row with index number 4111 is duplicate of row with index number 1233 or vice-versa) and the features are not correlated because most of them are having texts.

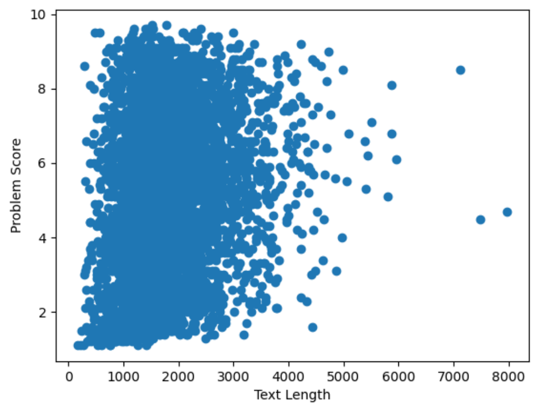
So, it can be concluded that I was provided with the dataset full of texts and having two types of result one was classification (easy/medium/hard) and other was regression (numerical data)

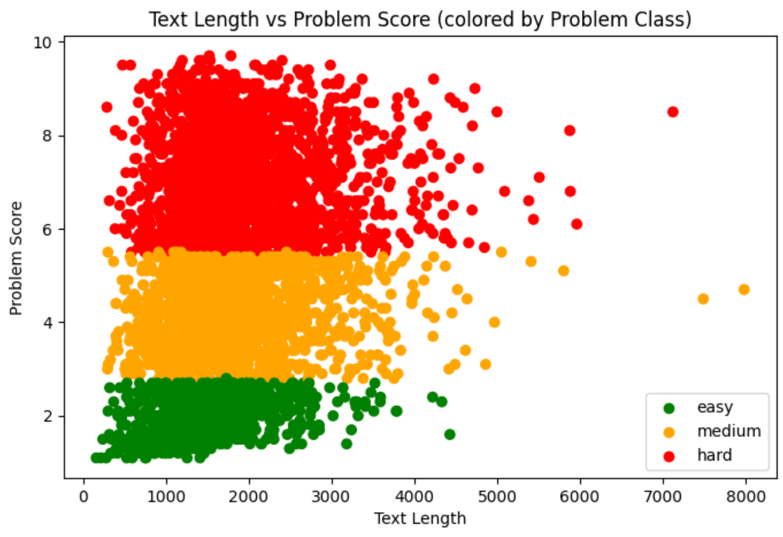
## Data pre-processing:

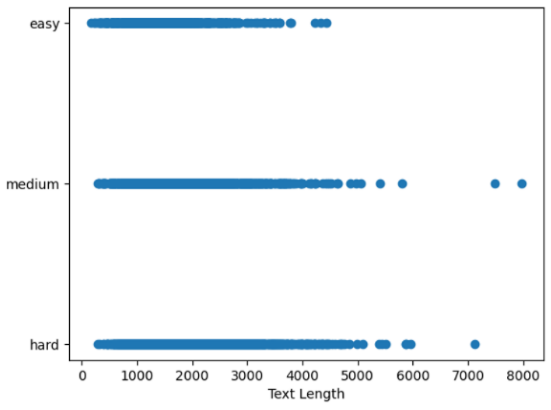
- Conversion of provided raw JSON file to CSV and then found null values and then replaced with " "
- I found the correlation between columns, and it does not exist because they are text rich columns and only one has numerical data
- Finding if any duplicate value exists or not and I tried to drop this, but it decreased the accuracy of the model hence I did not execute this, the possible reason for accuracy drop could be
  (1) Reduction in effective training information
  (2) Sample frequency of important patterns decreases
  (3) Removing duplicates increases data sparsity
- I combined all the text columns and tried to drop URL column, but it also reduced evaluation metrices hence I did not perform this
- Text Cleaning

## Feature Engineering:

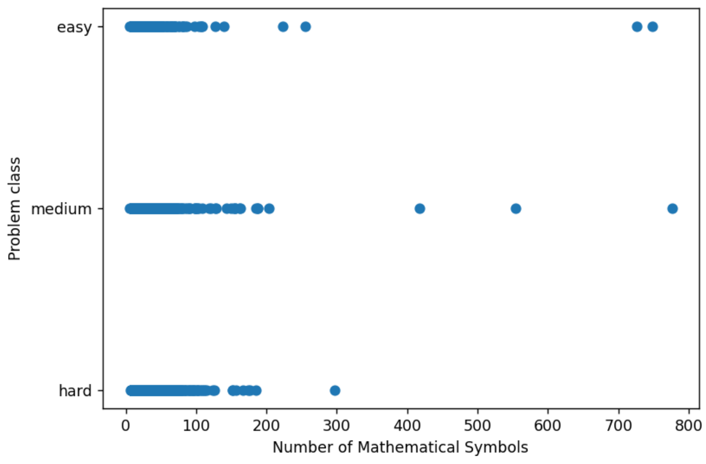
- I found text_length and then made scatterplot of this with both problem_score and problem_class and then individually with both

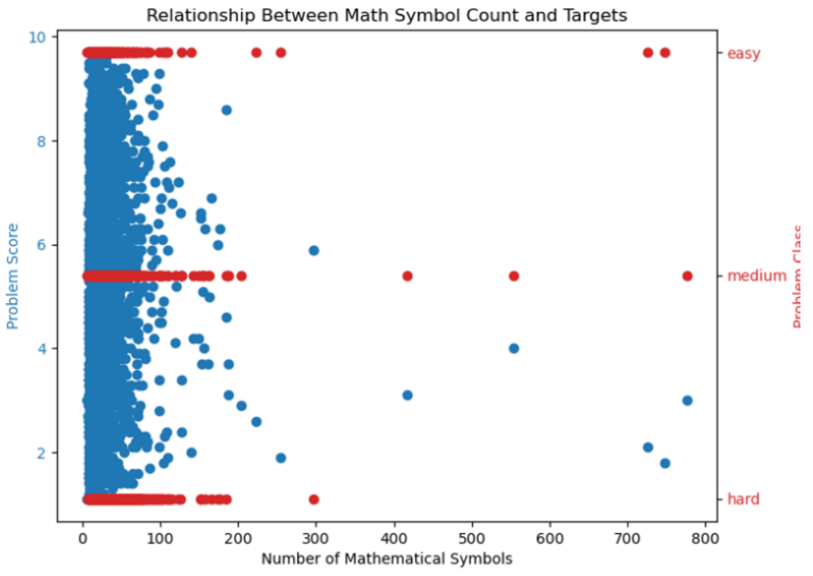


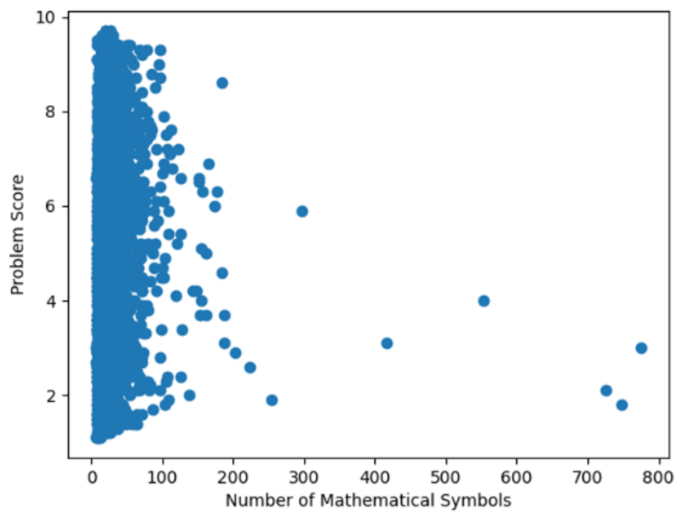




- Cleaned the text and then found
    - Number of mathematical symbols and then made its graph with problem_score, problem_class and then with both

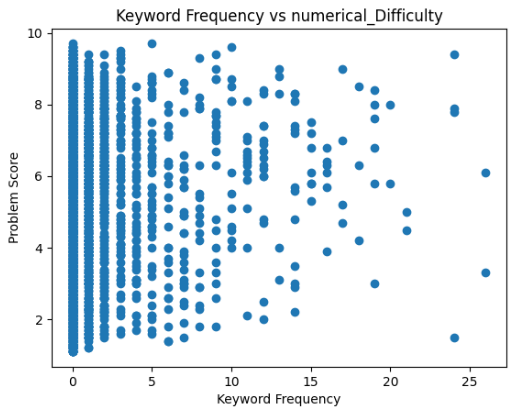


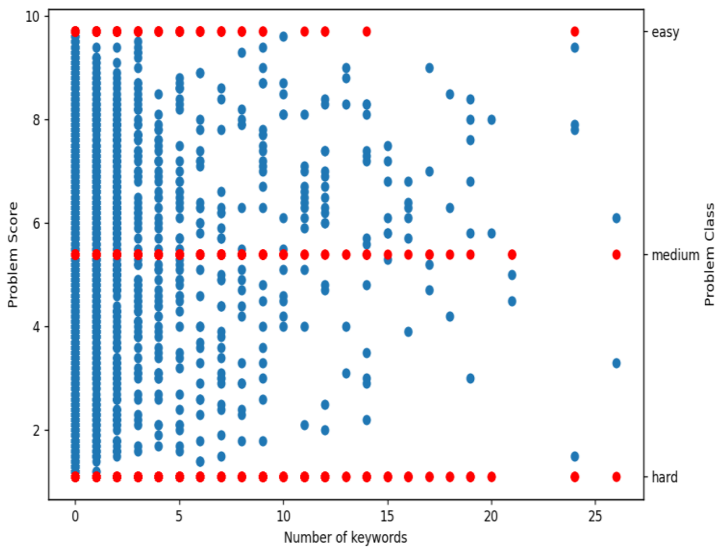


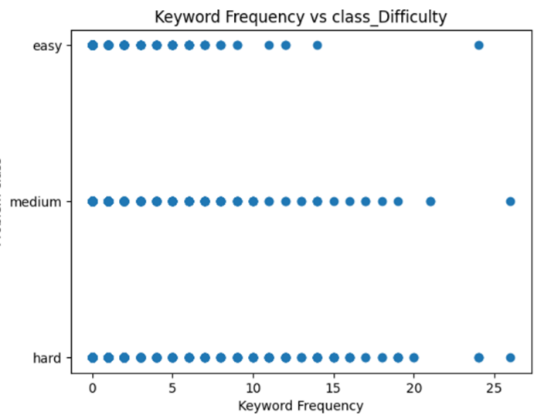


b. Number of sentences and length of those sentences
c. Created constraints (constraints, 1 ≤ n, upper-bound expressions, time constraints, memory constraints) and then found its count
d. Calculated io_complexity which gives the idea that how much time it was taken to read or understand that text
e. Found the frequency a word (keyword_count), types of word getting repeated(keyword_diversity) and then assigned weights to various types of word based on their capability of increasing the toughness of problem

f. Plotted graphs of keyword frequency with problem_score, problem_class and then with both







g. I used TF-IDF vectors because this dataset is full of texts and ML models can't understand texts hence these were used to convert into scores
**TF:** How often a word appears
**IDF:** How rare that word is in a document
And then finally they both are multiplied together

h. I took the ngram range to be (1,2) it is little computationally expensive, but it was required to consider for words like dynamic programming, etc.

Finally, I scaled the numeric features and then combined that with TF-IDF vectors and labelled data into two y_reg (for regression) and y_clf (for classification)

## Train-test split:

- Training size --> 80%
- Test size --> 20%

**<u>Regression model:</u>**

- Imported all the necessary libraries from scikit learn and then tried with Gradient Boosting, Linear Regression and Random Forest
- It was evident from the scatterplots what Linear Regression won't be a good fit for this, its metrices are as follows,
  ***R2_score = -6.57*** ***RMSE = 6.041***
- The data has many numbers of tf-idf features and all the decision trees will be trained independently so, it will also not be the best choice, its metrices are
  ***R2_score = 0.129*** ***RMSE = 2.049***
- I finally tried with Gradient Boosting to be precise Hist gradient boosting it was truly the best model to be used among all three and hence its metrices are
  ***R2_score = 0.175*** ***RMSE = 1.99***

**<u>Classification model:</u>**

- From scatter plot again it was intuitive that I should not use logistic regression because the data does not seem to be linearly separable, hence for logistic regression
  - Classification accuracy = 50.42%
  - Confusion matrix=
    ```
    [[43   51   42]
     [19  297  109]
     [21  166   75]]
    ```
- As this is not linearly separable hence Linear SVM can't be used and non-linear SVM will be computationally expensive and would be difficult to deal with sparse data, so it is also not a good choice, for this metrices are as follows:
  - Classification accuracy = 47.14%
  - Confusion matrix=
    ```
    [[63   36   37]
     [52  239  134]
     [47  129   86]]
    ```
- Random forest will be the best choice for this type of data; hence it is which can be seen by its metrices:
  - Classification accuracy = 57.35%
  - Confusion matrix=
    ```
    [[49   65   22]
     [27  365   33 ]
     [23  181   58]]
    ```

**Web Interface:**

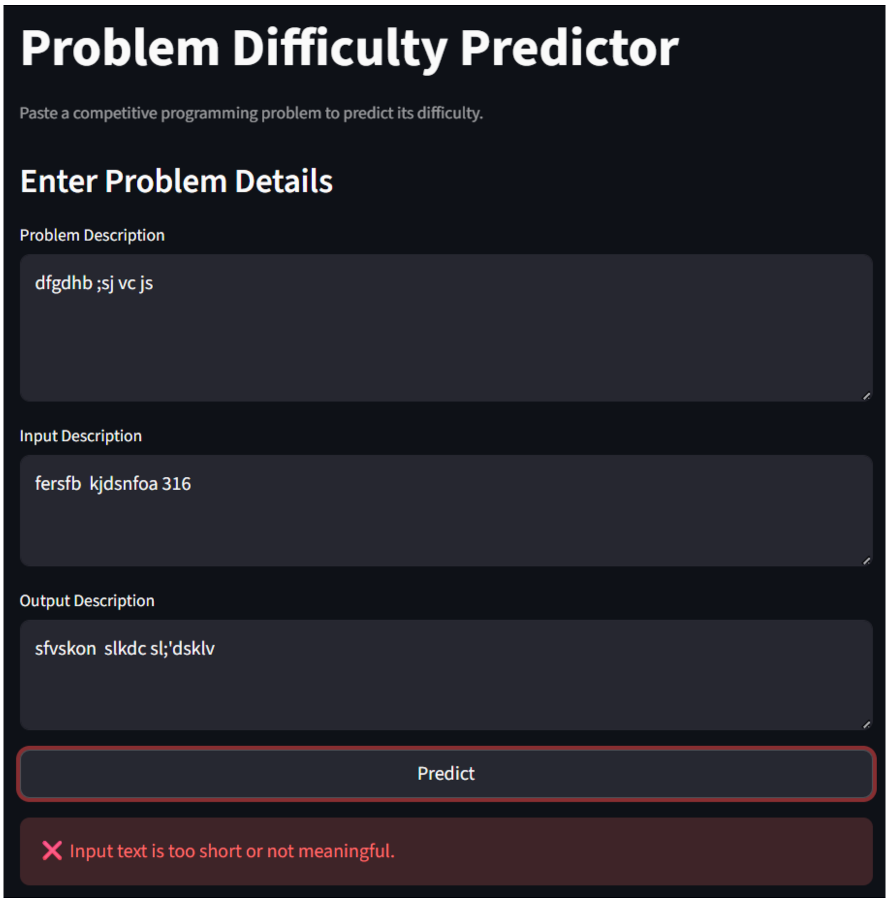
- The models were trained and all required files (ifidf.pkl, scaler.pkl, reg_model.pkl, clf_model.pkl and label_encoder.pkl) were created from the code given in the file **model_code.py**
- Then the web interface was created using Streamlit and its code is in **Web_UI_code.py**
- Then in the terminal a command **python -m streamlit run Web_UI_code.py** was run (I used python in start of command to resolve the issue of file location)
- Then this code was connected to Streamlit using my GitHub repo
- For app creation go to create app and then enter all the details and click deploy
- A validation mechanism was incorporated into the codebase to restrict random or unstructured input descriptions.

There were some compatibility issues with the Python and scikit-learn versions which were fixed by adding files requirements.txt and runtime.txt.

**User Input → Text Preprocessing → Feature Engineering → TF-IDF Vectorization → Feature Scaling → Feature Fusion → Regression & Classification Inference → Difficulty Score and Class Output**

**Predictions:**

The deployed app was tested using sample problem statements from the dataset to ensure it was working correctly.


Problem Difficulty Predictor

Paste a competitive programming problem to predict its difficulty.

Enter Problem Details

Problem Description: dfgdhb ;sj vc js

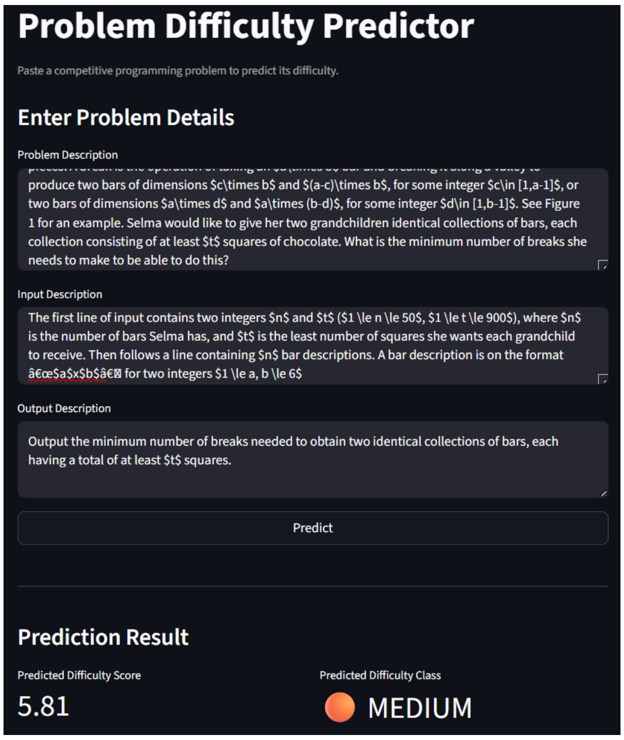
Input Description: fersfb kjdsnfoa 316

Output Description: sfvskon slkdc sl;'dsklv

Predict

✕ Input text is too short or not meaningful.

This shows that this app does not give any output if random entries are filled


Problem Difficulty Predictor

Paste a competitive programming problem to predict its difficulty.

Enter Problem Details

Problem Description

produce two bars of dimensions $c\times b$ and $(a-c)\times b$, for some integer $c\in [1,a-1]$, or two bars of dimensions $a\times d$ and $a\times (b-d)$, for some integer $d\in [1,b-1]$. See Figure 1 for an example. Selma would like to give her two grandchildren identical collections of bars, each collection consisting of at least $t$ squares of chocolate. What is the minimum number of breaks she needs to make to be able to do this?

Input Description

The first line of input contains two integers $n$ and $t$ ($1 \le n \le 50$, $1 \le t \le 900$), where $n$ is the number of bars Selma has, and $t$ is the least number of squares she wants each grandchild to receive. Then follows a line containing $n$ bar descriptions. A bar description is on the format â€œ$a$x$b$â€ for two integers $1 \le a, b \le 6$

Output Description

Output the minimum number of breaks needed to obtain two identical collections of bars, each having a total of at least $t$ squares.

Predict

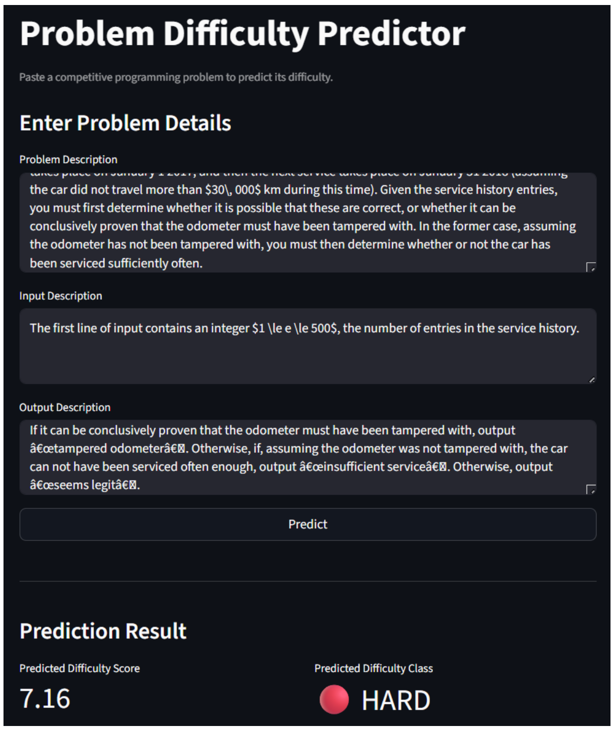
Prediction Result

Predicted Difficulty Score

5.81

Predicted Difficulty Class

MEDIUM


Problem Difficulty Predictor

Paste a competitive programming problem to predict its difficulty.

Enter Problem Details

Problem Description

the car did not travel more than $30\, 000$ km during this time). Given the service history entries, you must first determine whether it is possible that these are correct, or whether it can be conclusively proven that the odometer must have been tampered with. In the former case, assuming the odometer has not been tampered with, you must then determine whether or not the car has been serviced sufficiently often.

Input Description

The first line of input contains an integer $1 \le e \le 500$, the number of entries in the service history.

Output Description

If it can be conclusively proven that the odometer must have been tampered with, output â€œtampered odometerâ€. Otherwise, if, assuming the odometer was not tampered with, the car can not have been serviced often enough, output â€œinsufficient serviceâ€. Otherwise, output â€œseems legitâ€.

Predict

Prediction Result

Predicted Difficulty Score

7.16

Predicted Difficulty Class

HARD

## Conclusions:

- This project presents an automated system to predict the difficulty level and score of programming problems using their textual descriptions.
- Text preprocessing, TF-IDF vectorisation, and numerical features were used to convert text into a suitable form for machine learning.
- Gradient Boosting performed best for score prediction, while Random Forest gave the highest accuracy for difficulty classification.
- The entries to be filled while using the Web App was improved by preventing random entries.

Overall, the deployed AutoJudge application provides a simple and effective way to estimate problem difficulty.

**App URL**: https://autojudge-predicting-programming-problem-difficulty-vgep9awfjp.streamlit.app/

**Github URL**: laksh7218 (aandhiizz)