AutoJudge: Predicting Programming Problem Difficulty Using Machine Learning

Vishal Meena

vishal_m@ch.iitr.ac.in

1 Introduction

Competitive programming platforms host thousands of problems with varying difficulty levels. Accurate difficulty labeling is essential for guiding learners, organizing contests, and ensuring fair assessment. Traditionally, difficulty levels are manually assigned by experts, making the process time-consuming and subjective.

This project, **AutoJudge**, aims to automatically predict the **difficulty class (Easy / Medium / Hard)** and a **numerical difficulty score (1-10)** for programming problems using their textual descriptions. By leveraging **Natural Language Processing (NLP)** and machine learning models, the system learns patterns from historical problem data and provides consistent, data-driven difficulty estimation.

2 Dataset Description

The dataset consists of programming problems collected from online competitive programming platforms. Each problem record contains the following attributes:

- **Title** – Short name of the problem
- **Description** – Full problem statement
- **Input Description** – Input format
- **Output Description** – Output format
- **Sample I/O** – Example inputs and outputs
- **Problem Class** – Easy / Medium / Hard
- **Problem Score** – Numerical difficulty score (1–10)
- **URL** – Source link

A preliminary inspection confirmed that the dataset contains **no missing values**, making it suitable for direct preprocessing.

3 Data Preprocessing

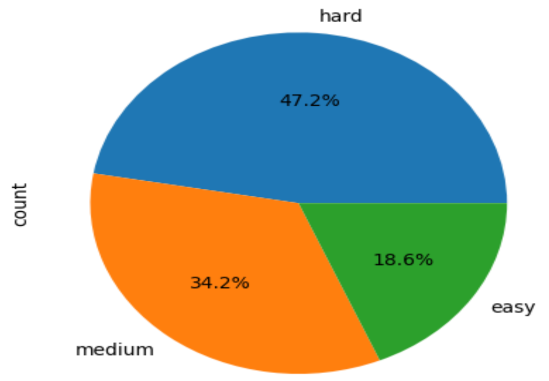
All text-related fields were concatenated into a **single textual feature** to represent the full problem statement. Preprocessing steps included *lowercasing, removal of extra spaces*, and normalization of line breaks. No missing value imputation was required.

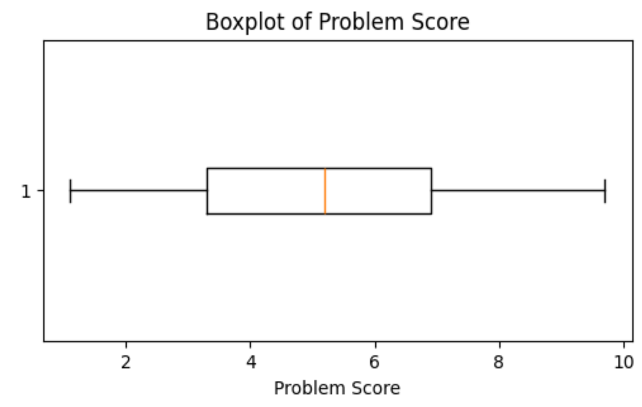
For both classification and regression, a unified preprocessing pipeline was adopted. Categorical difficulty labels were encoded using label encoding for classification.

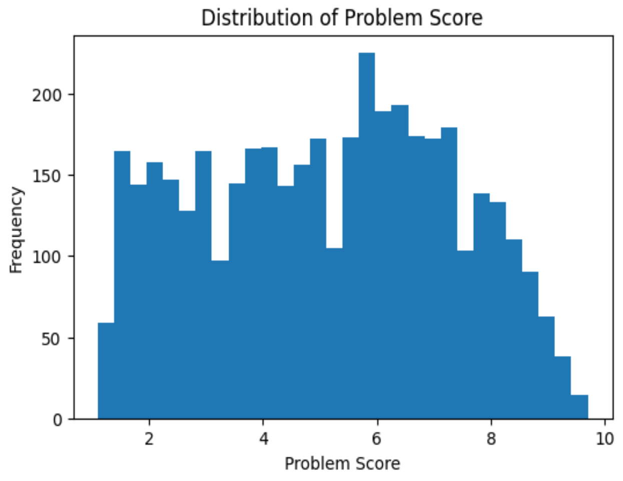
For regression, the numerical difficulty score was ultimately modeled **directly on the original 1–10 scale** to preserve interpretability in real-world usage. This ensured that predicted scores correspond directly to human-interpretable difficulty levels.

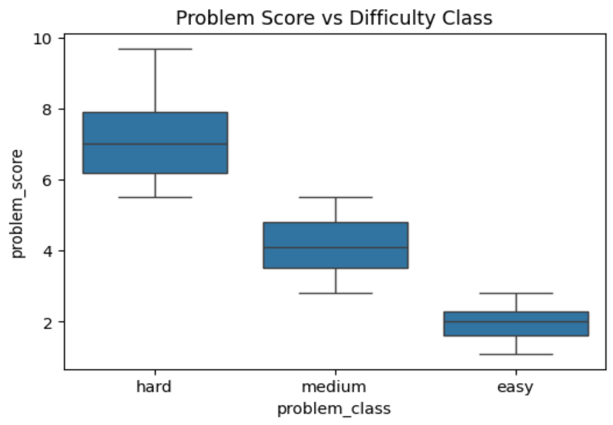
4. Exploratory Data Analysis

Exploratory Data Analysis (EDA) revealed that the dataset is moderately **imbalanced**, with hard problems appearing more frequently than easy ones.









The **distribution of problem scores** shows a wide spread from low to high values, with most scores concentrated in the medium range and **no extreme outliers**. The **boxplot** confirms a fairly balanced score distribution with moderate variance. Analysis of text length demonstrated that **harder problems generally have longer and more complex descriptions**.

The **boxplot of Problem Score vs Difficulty Class** shows that easy problems generally have lower scores, while **hard problems have higher scores**. This confirms that the difficulty class labels are consistent with the numerical problem scores and validates using the same dataset for both classification and regression.

These observations motivated the use of balanced evaluation metrics and the inclusion of structural text features such as word count and text length.

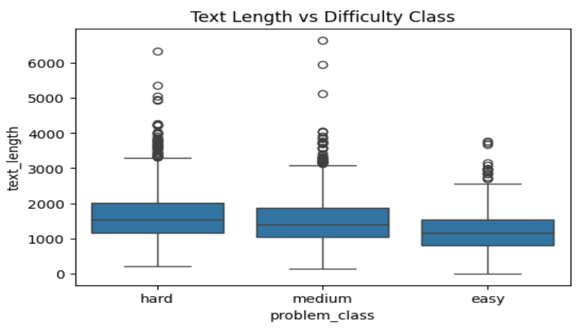
5 Feature Engineering

5.1 Text-Based Features

- TF-IDF vectors : To convert textual data into numerical representations, Term Frequency–Inverse Document Frequency (TF-IDF) was employed. Both unigrams and bigrams were considered. Initially, a high-dimensional TF-IDF space was generated.

- Vocabulary size initially set to 50,000.

5.2 Numeric Features



- Text length

- Word count

- Maximum numeric constraint extracted from text

The analysis shows that **hard problems generally have longer tex**t descriptions compared to easy problems, indicating higher complexity and detailed explanations. Medium problems lie between easy and hard in terms of text length.

in addition to TF-IDF features, **keyword-based difficulty signals** were engineered using domain knowledge from competitive programming.

- **Easy keywords:** array, loop, print, sum, basic, integer, simple

- **Medium keywords:** binary search, DFS, BFS, sorting, greedy, stack, queue, two pointers

- **Hard keywords:** DP, dynamic programming, bitmask, segment tree, union find, flow, graph theory

For each problem, the frequency of Easy, Medium, and Hard keywords was computed and used as numeric features. These features provide **explicit semantic cues** about problem complexity that complement TF-IDF representations.

5.3 Feature Selection

Due to the very high dimensionality of TF-IDF features, **SelectKBest** was used:

- **Chi-square test** for classification
- **F-regression test** for regression

Feature selection was applied **only on training data**, preventing data leakage and improving model generalization.

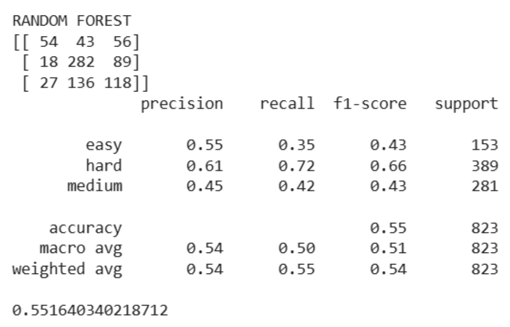
6 Classification Models

The classification task aims to predict the **difficulty class** of a problem.

6.1 Models Evaluated

i. Logistic Regression
ii. Random Forest Classifier
iii. Support Vector Machine (SVM)
iv. XGBoost

Among these, Random Forest Classifer consistently outperformed other models due to its effectiveness on high-dimensional sparse text data.



```
RANDOM FOREST
[[ 54  43  56]
 [ 18 282  89]
 [ 27 136 118]]
              precision    recall  f1-score   support

        easy       0.55      0.35      0.43       153
        hard       0.61      0.72      0.66       389
      medium       0.45      0.42      0.43       281

    accuracy                           0.55       823
   macro avg       0.54      0.50      0.51       823
weighted avg       0.54      0.55      0.54       823

0.551640340218712
```

We used **SelectKBest(chi2)** to reduce the **high-dimensional TF-IDF feature space** by selecting only the most informative words related to the target classes. The chi-square test measures the statistical dependency between each word and the class labels, allowing the model to focus on features that contribute most to distinguishing problem difficulty levels. Selecting the top features reduces noise, improves generalization, and **significantly enhances model performance**.

6.2 Model Selection

RF consistently outperformed other classifiers due to its ability to handle sparse, high-dimensional text data effectively.

Model	**Accuracy**	**Macro F1-score**
Logistic Regression	0.48	0.47
Random Forest	**0.5516**	**0.50**
SVM	0.49	0.46

7. Regression Models

The regression task aims to predict a continuous numerical difficulty score for programming problems.
To ensure consistency, **the same TF-IDF and numeric feature pipeline used for classification was reused for regression**, with only the target variable and feature selection criterion changed. This design eliminated redundant preprocessing and ensured fair model comparison.

7.1 Models Evaluated

6.3.1 Linear Regression

6.3.2 Ridge Regression

6.3.3 Random Forest Regression

6.3.4 Gradient Boosting Regression

6.3.5 XGBoost Regression

7.2 Regression Model Comparison

Model	**MAE**	**RMSE**	**R²**	**Remarks**
Linear Regression	2.906	4.134	-2.521	Underfits high dimensional data
Ridge Regression	1.704	2.034	0.146	stable and interpretable baseline
Random Forest Regression	1.704	2.043	0.14	Overfits mid-range scores
Gradient Boosting Regression	1.69	2.026	0.154	Strong non-linear performance
XGBoost Regression	**1.681**	**2.025**	**0.15**	**Best overall regression model**

7.3 Final Model Selection

Based on the final evaluation on the test dataset, **Gradient Boosting Regression (GBR)** and **XGBoost Regression** demonstrated very similar performance in predicting the numerical difficulty score. The results show that both models achieved comparable error values, with **XGBoost Regression obtaining a slightly lower MAE (1.6549)** and **GBR achieving a nearly identical RMSE (1.9976)**.

Although the difference between the two models is marginal, **XGBoost Regression achieved the highest R² score (0.1783)**, indicating a slightly better ability to explain variance in the difficulty scores compared to Gradient Boosting Regression (R² = 0.1777). This suggests that XGBoost captures non-linear relationships in the data marginally more effectively.

Given its **lowest MAE**, **competitive RMSE**, and **highest R² score**, **XGBoost Regression was selected as the final regression model** for difficulty score prediction. Gradient Boosting Regression was retained as a strong comparative baseline due to its stable and consistent performance.

Model	**MAE**	**RMSE**	**R²**
Gradient Boosting Regression (GBR)	1.6665	1.9976	0.1777
XGBoost Regression	**1.6549**	**1.9969**	**0.1783**

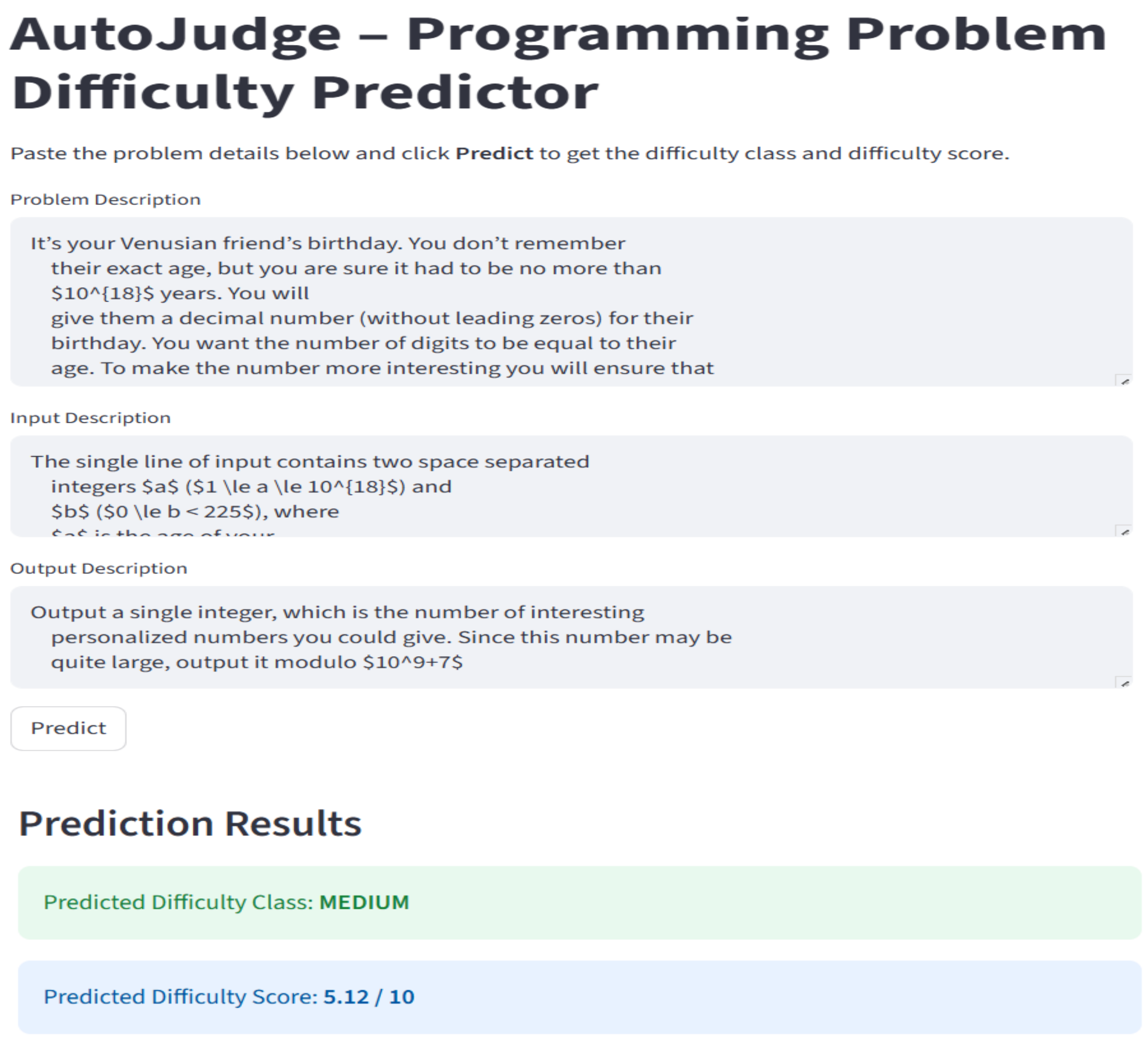
8 Results and Evaluation

The AutoJudge system was evaluated on both classification and regression tasks using a test dataset. For classification, the Random Forest model achieved the best performance with an accuracy of **0.55** and a Macro F1-score of **0.50**, indicating balanced prediction across difficulty classes. For regression, **XGBoost Regression** performed best with a **MAE of 1.65**, **RMSE of 1.99**, and **R^2 of 0.18** on the 1–10 scale. These results demonstrate that combining textual, numeric, and keyword-based features enables reliable difficulty estimation.

9 Web Application

A web application was developed using Streamlit to demonstrate the trained models. Users can input a programming problem description, and the system predicts both the difficulty class and the difficulty score in real time. All trained models and preprocessing components were saved using joblib and loaded during inference.

Sample Example (screenshot with prediction):



AutoJudge – Programming Problem Difficulty Predictor

Paste the problem details below and click **Predict** to get the difficulty class and difficulty score.

Problem Description

It's your Venusian friend's birthday. You don't remember their exact age, but you are sure it had to be no more than \$10^{18}\$ years. You will give them a decimal number (without leading zeros) for their birthday. You want the number of digits to be equal to their age. To make the number more interesting you will ensure that

Input Description

The single line of input contains two space separated integers \$a\$ (\$1 \le a \le 10^{18}\$) and \$b\$ (\$0 \le b < 225\$), where

Output Description

Output a single integer, which is the number of interesting personalized numbers you could give. Since this number may be quite large, output it modulo \$10^9+7\$

Prediction Results

Predicted Difficulty Class: **MEDIUM**

Predicted Difficulty Score: **5.12 / 10**

Problem description : It's your Venusian friend's birthday. You don't remember their exact age, but you are sure it had to be no more than 10^{18} years. You will give them a decimal number (without leading zeros) for their birthday. You want the number of digits to be equal to their age. To make the number more interesting you will ensure that no adjacent pairs of digits will be identical.
Their exact day of birth is represented as an integer in the range 0 to 224 (since Venus has 225 days in a year). To make their gift more personal you want the given number to have the same remainder as their birthday when divided by 225.
There are potentially a lot of possible gifts that you could give. You may decide to give more than one gift. Determine the number of possible gifts modulo 10^9+7.

Input: The single line of input contains two space separated integers a ($1 \le a \le 10^{18}$ and b ($0 \le b < 225$), where a is the age of your friend and b is the birthdate of your friend.

Output: a single integer, which is the number of interesting personalized numbers you could give. Since this number may be quite large, output it modulo 10^9+7.

Actual	Prediction by our model
Class: medium	Class: MEDIUM
Score: 5.4	Score: 5.12

10 Conclusion

This project demonstrates that machine learning combined with NLP techniques can effectively predict programming problem difficulty. Feature engineering and feature selection played a crucial role in improving model performance. The system is interpretable, efficient, and suitable for real-world deployment.

Future work may explore transformer-based embeddings and larger datasets to further improve prediction accuracy.