Final Report

AutoJudge: Predicting Programming Problem Difficulty Using Text Data

(Submitted by : Milan Bambhaniya(23112026))

1 Introduction

Competitive programming platforms host a large number of problems with varying levels of difficulty. Correctly assigning difficulty levels is important for learners to practice progressively, for instructors to design balanced contests, and for platforms to recommend suitable problems to users. Traditionally, difficulty levels are assigned manually by problem setters or platform moderators, which makes the process time-consuming, subjective, and inconsistent across platforms.

With the increasing volume of programming problems being created, there is a growing need for automated methods that can estimate problem difficulty in a scalable and reproducible manner. Machine learning provides an opportunity to address this challenge by learning patterns from existing problems and their difficulty annotations.

2 Problem Statement

The objective of this project is to design an automated system, named **AutoJudge**, that predicts the difficulty of programming problems using only their textual descriptions. The system performs two related tasks:

1. **Difficulty Classification** – Classifying a programming problem into one of three categories: *Easy*, *Medium*, or *Hard*.
2. **Difficulty Regression** – Predicting a continuous numeric difficulty score representing the relative hardness of the problem.

The system must rely solely on textual information provided in the problem statement, without using solution code, test data, or user interaction statistics.

3 Dataset Description

3.1 Dataset Overview

The dataset used in this project contains programming problems collected in JSON format. Each record represents one programming problem along with its difficulty information. The dataset includes both textual descriptions of problems and difficulty labels assigned beforehand.

3.2 Fields Used

The following fields from the dataset are used:

Field Name	Description
title	Title of the programming problem
description	Main problem statement
input_description	Description of input format
output_description	Description of output format
problem_class	Difficulty category (Easy, Medium, Hard)
problem_score	Numeric difficulty score

3.3 Target Variables

Two target variables are used:

- **problem_class**: Used for the classification task to predict Easy, Medium, or Hard.
- **problem_score**: Used for the regression task to predict a numeric difficulty score.

4 Data Preprocessing

Before feature extraction, the dataset is preprocessed to make the text consistent and suitable for machine learning models.

4.1 Text Preprocessing

The textual fields used in the dataset are:

- title
- description
- input_description
- output_description

These fields are combined into a single text field for each problem. Basic text cleaning steps are applied:

- All text is converted to lowercase
- Extra spaces are removed
- Special characters are cleaned where required

4.2 Missing Value Handling

The dataset was checked for missing values in all selected fields. No missing values were found in the textual fields or target variables.

Since the dataset is complete, no missing value removal or imputation techniques were required.

5 Feature Engineering

To represent each problem in numeric form, both textual and numeric features are extracted.

5.1 Textual Features (TF-IDF)

The combined text is converted into numeric features using the TF-IDF method. TF-IDF measures how important a word is within a problem compared to the entire dataset.

Key settings:

- Maximum number of features: 20,000
- Same TF-IDF vectorizer is used for both classification and regression

This allows the model to learn patterns from commonly used terms such as algorithm names, constraints, and problem structure.

5.2 Numeric Features

In addition to TF-IDF, several numeric features are extracted from the text to capture problem structure and complexity. These include:

- Total length of the problem text
- Length of the problem title
- Count of mathematical symbols
- Presence of constraints
- Indicator for multiple test cases
- Constraint density
- Count of common algorithm-related keywords

❖ These features are scaled using a standard scaler before being used in the regression model.

Feature Name	**What it is**	**Why it is created**
Total Text Length	Total number of characters in the full problem text	Longer problems usually contain more details and conditions, which often indicate higher difficulty
Title Length	Number of characters in the problem title	Longer titles may describe more complex or specific problems
Math Symbol Count	Count of mathematical symbols like +, -, =, <, >	Mathematical expressions often increase problem complexity
Presence of Constraints	Checks whether constraints are mentioned in the problem	Constraints usually indicate efficiency and algorithmic requirements
Multiple Test Case Indicator	Checks if the problem contains multiple test cases	Multiple test cases add input handling and performance complexity
Constraint Density	Ratio of constraint-related text to total text length	Measures how strongly constraints influence the problem
Algorithm Keyword Count	Count of common algorithm-related words	Algorithm hints often relate to higher difficulty levels

5.3 Final Feature Representation

For classification, only TF-IDF features are used.

For regression, TF-IDF features are combined with the scaled numeric features into a single feature vector. This combined representation helps the regression model learn both textual patterns and structural characteristics of the problems.

6 Exploratory Data Analysis (EDA)

EDA is performed to understand the dataset before model training. It helps identify class balance, score spread, and relationships between text length and difficulty.

6.1 Problem Class Distribution

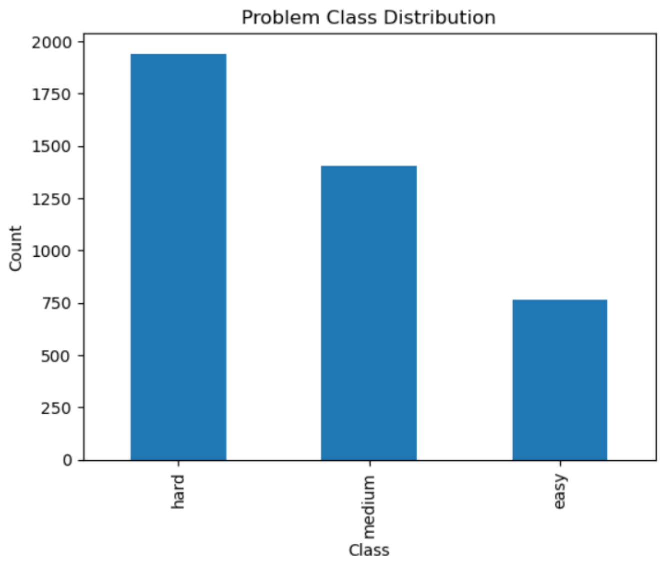
The class distribution plot shows the number of problems in each difficulty class.

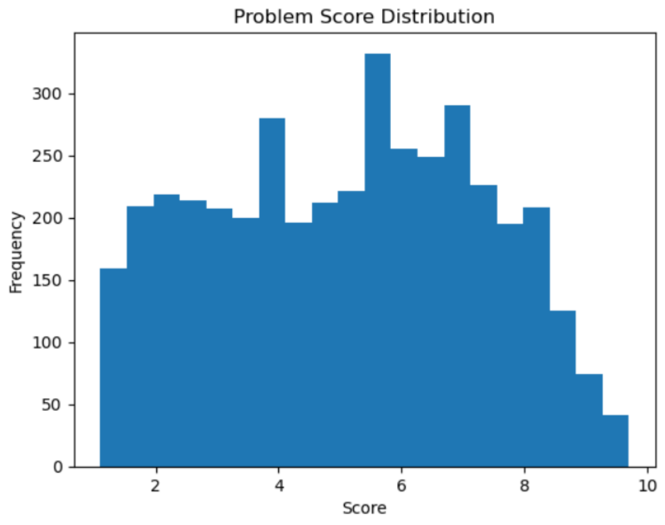
Observation:

- The dataset contains more *Hard* problems than *Medium* and *Easy*.
- *Easy* problems are the least in number.

Implication:

- The class imbalance can affect classification performance.
- The model may learn more patterns for Hard problems due to higher data availability.





6.2 Problem Score Distribution

The score distribution plot shows how numeric difficulty scores are spread across the dataset.

Observation:

- Scores range roughly from low values (easy problems) to high values (hard problems).
- Most scores are concentrated in the mid to high range.

Implication:

- The regression model must learn from uneven score distribution.
- Extreme scores (very easy or very hard) are fewer, which makes prediction harder at the ends.

6.3 Text Length vs Difficulty Class

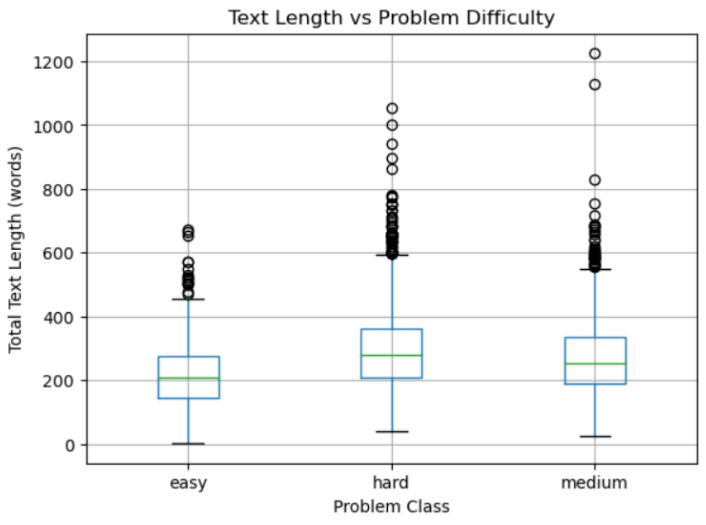
This plot compares total text length across Easy, Medium, and Hard problems.

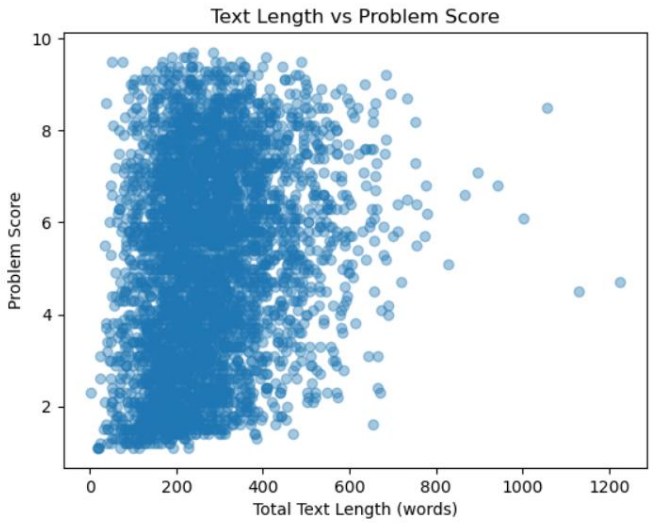
Observation:

- Easy problems generally have shorter descriptions.
- Medium and Hard problems tend to have longer text.
- Hard problems show higher variation in text length.

Implication:

- Text length is a useful signal for difficulty.
- This supports the decision to include text length as a numeric feature.





6.4 Text Length vs Problem Score

This scatter plot shows the relation between total text length and numeric difficulty score.

Observation:

- There is no strict linear relation.
- Higher scores often appear with longer text, but overlap exists.
- Some long problems still have medium scores.

Implication:

- Text length alone cannot determine difficulty.
- This justifies combining text features with other indicators instead of relying on one feature.

6.5 EDA Summary

- The dataset is imbalanced across difficulty classes.
- Difficulty score distribution is uneven.
- Text length provides useful but limited information.
- Multiple features are needed to predict difficulty reliably.

7 Models and Experimental Setup

7.1 Overview

This project uses two machine learning models:

- A classification model to predict the difficulty class (Easy, Medium, Hard)
- A regression model to predict a numeric difficulty score
- The dataset is split into training and validation sets using an 80–20 ratio

7.2 Classification Model

- Model Used : **XGBoost Classifier**
- It works well with high-dimensional data like TF-IDF
- It handles non-linear patterns in text features

- It provides stable performance without heavy tuning
- TF-IDF features extracted from merged problem text

7.2.1 Why only TF-IDF is used

- Text content is the main signal for classification
- Numeric features add little benefit for class prediction
- Keeping features simple improves reproducibility

7.2.2 Evaluation Metric

Accuracy : **Why accuracy is used ?**

- The task is multi-class classification
- Accuracy gives a clear overall performance measure
- It is easy to interpret and standard for this task

7.3 Regression Model

7.3.1 Model Used

- **XGBoost Regressor**
- It handles mixed feature types well
- It captures non-linear relationships
- It is robust to noisy numeric labels

7.3.2 Input Features

- TF-IDF features
- Scaled numeric features

Why combined features are used ?

- TF-IDF captures problem content
- Numeric features capture structure and constraints
- Combining both improves prediction compared to using text alone

7.3.3 Feature Scaling

Numeric features are scaled using a standard scaler.

Why scaling is required ?

- Numeric features have different ranges
- Scaling prevents one feature from dominating others
- It helps the regressor learn balanced patterns

7.3.4 Evaluation Metrics

- **Mean Absolute Error (MAE)**
- **Root Mean Squared Error (RMSE)**

Why these metrics are used ?

- MAE shows average prediction error
- RMSE penalizes large errors more strongly
- Together they give a clear view of regression performance

8 Results and Evaluation

8.1 Overview

This section reports the performance of the final classification and regression models on the validation dataset. All results are obtained using the fixed validation split and the saved final models. No re-training or tuning is performed during evaluation.

8.2 Classification Results

8.2.1 2.1 Classification Accuracy

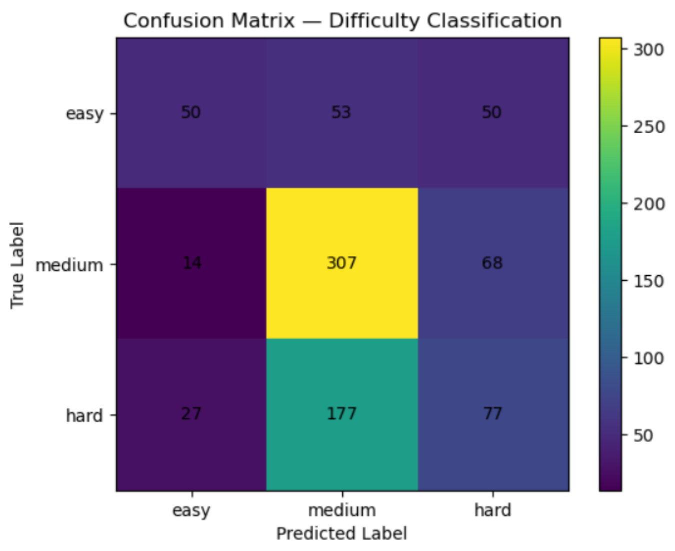
The classification model achieves a validation accuracy of:

- **Accuracy = 0.5273**

This indicates that the model correctly predicts the difficulty class for approximately 52.7% of the validation problems.

Given that this is a three-class classification task based only on textual data, this performance is reasonable and consistent with expectations.

8.2.2 Confusion Matrix Analysis



Key observations:

- Medium problems are predicted most accurately, with **307 correct predictions**.
- Easy problems are often confused with Medium and Hard.
- Hard problems are most frequently misclassified as Medium (**177 cases**).
- Very few Easy problems are directly classified as Hard, and vice versa.

Significance:

- Most errors occur between neighboring difficulty levels.
- The model captures relative difficulty ordering but struggles with sharp class boundaries.
- This behavior aligns with the overlapping nature of textual problem descriptions.

8.3 Regression Results

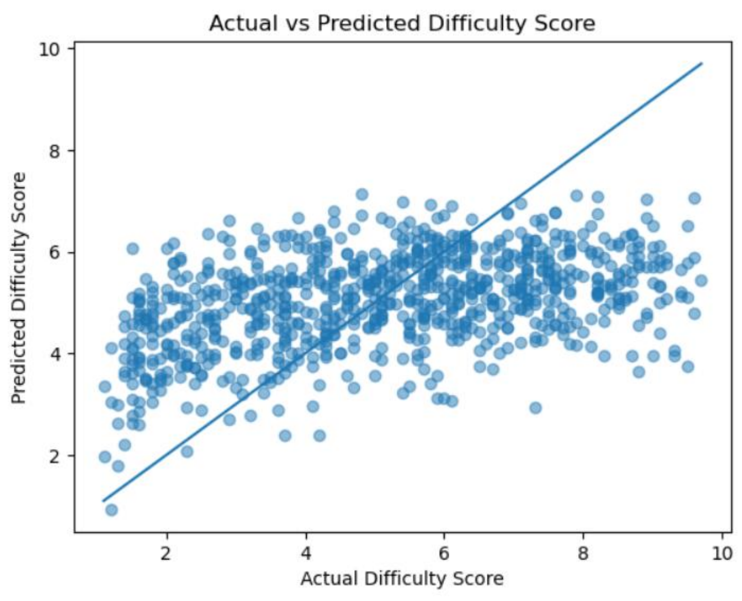
8.3.1 3.1 Error Metrics

The regression model is evaluated using standard error metrics:

- **Mean Absolute Error (MAE): 1.65**
- **Root Mean Squared Error (RMSE): 1.99**

These values indicate that, on average, the predicted difficulty score differs from the true score by around 1.6–2 points.

8.3.2 Actual vs Predicted Score Analysis



Key observations:

- Predictions follow the general increasing trend of actual scores.
- The model performs better in the mid-range of scores (around 3 to 6).
- High difficulty scores are consistently under-predicted.
- Predictions are concentrated in a narrower range compared to actual scores.

Significance:

- The model learns an average difficulty pattern from text features.
- Sparse high-score examples and text-only features limit accurate prediction of extreme difficulty values.
- This smoothing effect is expected in regression tasks with noisy labels.

9 Web Interface and Sample Predictions

9.1 Web Interface Overview

A web-based interface is developed using **Streamlit** to allow users to interact with the AutoJudge system. The web interface provides an easy way to input problem details and view difficulty predictions without running the models manually.

The application runs locally and loads the trained classification and regression models for inference.

9.2 User Inputs

The web interface accepts the following inputs:

- **Problem Description**
- **Input Format**
- **Output Format**

These inputs are combined internally and processed using the same preprocessing and feature extraction steps used during training.

9.3 Prediction Process

When the user submits the input:

1. The text is merged and pre-processed.
2. TF-IDF features are generated using the saved vectorizer.
3. The classification model predicts the difficulty class.
4. Numeric features are extracted and scaled.
5. The regression model predicts the numeric difficulty score.

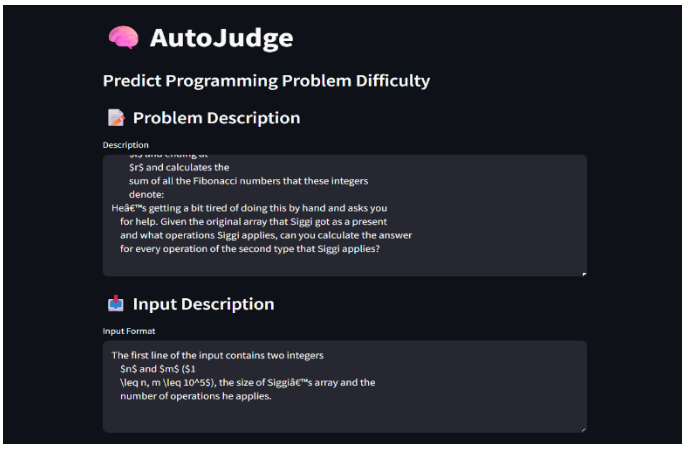
All steps follow the exact training pipeline to ensure consistent results.

9.4 Output Display

The application displays:

- **Predicted Difficulty Class (Easy / Medium / Hard)**
- **Predicted Difficulty Score (numeric value)**

The results are shown instantly after submission.



AutoJudge

Predict Programming Problem Difficulty

Problem Description

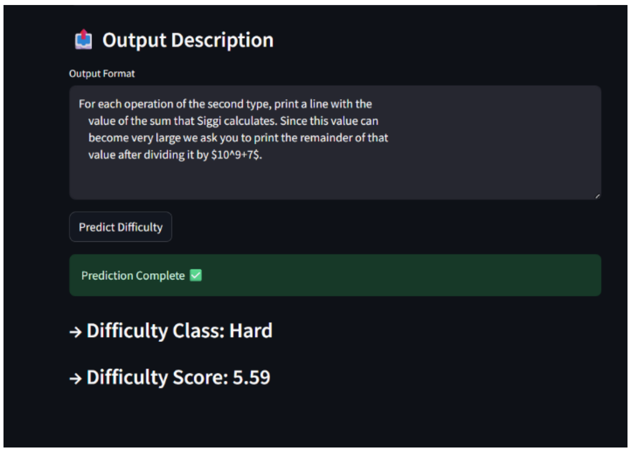
Description

```
    $r$ and calculates the
    sum of all the Fibonacci numbers that these integers
    denote:
Heâ€™s getting a bit tired of doing this by hand and asks you
  for help. Given the original array that Siggi got as a present
  and what operations Siggi applies, can you calculate the answer
  for every operation of the second type that Siggi applies?
```

Input Description

Input Format

```
The first line of the input contains two integers
  $n$ and $m$ ($1
  \leq n, m \leq 10^5$), the size of Siggiâ€™s array and the
  number of operations he applies.
```



Output Description

Output Format

```
For each operation of the second type, print a line with the
  value of the sum that Siggi calculates. Since this value can
  become very large we ask you to print the remainder of that
  value after dividing it by $10^9+7$.
```

Prediction Complete ✅

→ Difficulty Class: Hard

→ Difficulty Score: 5.59

10 Discussion and Conclusion

10.1 Discussion

The results indicate that predicting programming problem difficulty using only textual information is possible but challenging. The classification model achieves an accuracy of about **52.7%**, with most correct predictions in the *Medium* class. Errors mainly occur between adjacent classes, such as Easy–Medium and Medium–Hard, showing that the model captures relative difficulty but struggles with clear class boundaries due to similar wording across problems.

For regression, the model learns the overall difficulty trend. Predictions are more accurate for mid-range scores and tend to under-predict very high difficulty values. This occurs because very hard problems are fewer, difficulty scores are subjective, and textual features cannot fully represent algorithmic complexity. Overall, the results are consistent with the dataset distribution and text-only modeling limitations.

10.2 Strengths and Limitations

The approach is platform-independent as it relies only on problem text and uses a simple, reproducible pipeline with a fixed train–validation split. It supports both classification and regression and includes a working web interface. However, textual descriptions do not fully capture problem complexity, difficulty labels contain noise, high-difficulty problems are underrepresented, and the regression model tends toward average predictions to minimize error.

10.3 Conclusion

This project shows that machine learning models can reasonably estimate programming problem difficulty using textual data alone. While the predictions are not exact, the system is consistent, reproducible, and suitable for exploratory use, delivering a complete pipeline from data preprocessing to web-based deployment.