AutoJudge: Programming Problem Difficulty Prediction Model

1. Introduction

Online programming platforms like Codeforces, Kattis, and CodeChef use problem difficulty ratings to help users to choose problems. These ratings are mainly given by experts or are based on user feedback, which can vary from user to user and consume a lot of time. A model like this can help by predicting difficulty in a faster way and more consistently.

The goal of this project is to build a model that:

1. Uses only textual description of a problem.
2. Predicts the difficulty class of a programming problem.
3. Predicts a numerical difficulty score.
4. Shows predictions through a web interface.

2. Problem Statement

Given the textual description of a programming problem, which includes important parameters such as:

- Problem description
- Input description
- Output description
- Sample input/output

the task is to:

- Classify the problem as Easy, Medium, Hard.
- Predict a numerical score that represents the difficulty of a given problem.

3. Dataset Description

The dataset used for this project (problems_data.jsonl), consists of programming problems collected from the platform “Kattis”.

3.1 Dataset Fields

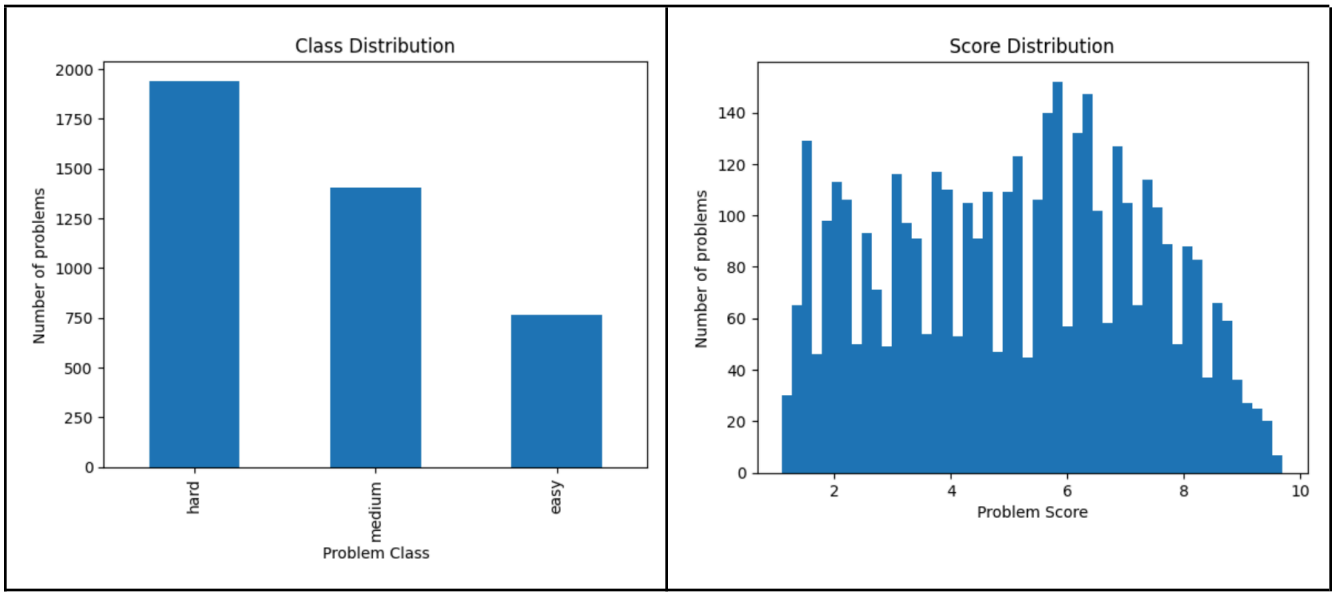
Each data sample includes:

- Title
- Problem description
- Input description
- Output description
- Sample input/output

- Problem Class -(Easy/Medium/Hard)
- Problem Score -(Numerical difficulty score between 1-10)
- url

3.2 Dataset Statistics

- **Total problems:** 4112
- **Average difficulty score:** 5.1
- **Class distribution:**
    - Hard problems: 1941
    - Medium problems: 1405
    - Easy problems: 766



The dataset is imbalanced, with a much higher number of hard problems than medium and easy problems. This imbalance affected the classification results and was dealt with using class weighting during model training.

4. Data Preprocessing

To prepare the given dataset for feature extraction, the following steps were applied:

4.1 Preprocessing Steps

- Combined the following text columns into a single column:
    - Title
    - Problem description
    - Input description
    - Output description
    - Sample input/output

- Replaced missing values with empty strings.
- Converted all text to lowercase.

- Removed newline and tab characters.
- Normalized extra whitespace.
- Saved the processed dataset as cleaned_data.csv

5. Feature Engineering

Feature engineering was used to convert raw problem text into numerical features that can be used by the two models.

5.1 Textual Features

- TF-IDF Vectorization was used to convert the text data to numerical features.
- Unigrams (Single words) and bigrams (Two word phrases) were considered to be evaluated.
- Common English stopwords were removed.
- Sublinear term frequency scaling (Logarithmic scaling) was used to reduce the effect of frequent words.

5.2 Numeric Features

The following numeric features were also added:

- **Text length:** Total number of characters in the text.
- **Word count:** Total number of words in the text.
- **Keyword frequencies:** Normalized count of common terms such as-graph,dp,greedy,bit,lcm, modulo,prime,tree,probability,etc.

Keyword frequencies were normalized by word count to prevent longer problem descriptions from overpowering feature values.

Mathematical symbol features (e.g., number of operators and numeric expressions) were also implemented but removed because they did not improve accuracy.

6. Classification Models

6.1 Experimental Setup

- Train-test split: 80% for training and 20% for testing.
- Stratified sampling was used to maintain the original distribution of the class.
- Text data was converted into numeric features using TF-IDF vectorization, along with the numeric features.
- Class weighting was applied during training to handle different classes of problems.
- Different combinations of parameter values were tested, the combination which performed the best was selected for the final model.
- Each model's performance was evaluated using:
    - Accuracy
    - Confusion matrix

6.2 Models Evaluated

```
Logistic Regression
Accuracy: 52.733900364520046 %

Confusion Matrix:
        easy  hard  medium
easy      60    48      45
hard      19   273      97
medium    21   159     101
```

Logistic Regression

- Achieved an accuracy of approximately **52.7%**.
- Showed noticeable confusion mainly between Medium and Hard classes.
- Performed well in identifying hard problems.
- Confusion matrix shows that class boundaries cannot be clearly separated through a linear surface.

```
Random Forest
Accuracy: 51.88335358444714 %

Confusion Matrix:
        easy  hard  medium
easy      46    76      31
hard      13   329      47
medium    19   210      52
```

Random Forest Classifier

- Achieved an accuracy of approximately **51.52%**.
- Mostly predicted problems as hard.
- Showed significant confusion between easy and hard problems.
- Can be used as a comparative model against the other two linear models.

```
Linear SVM
Accuracy: 54.07047387606318 %

Confusion Matrix:
        easy  hard  medium
easy      57    53      43
hard      16   293      80
medium    18   168      95
```

Linear Support Vector Machine (SVM)

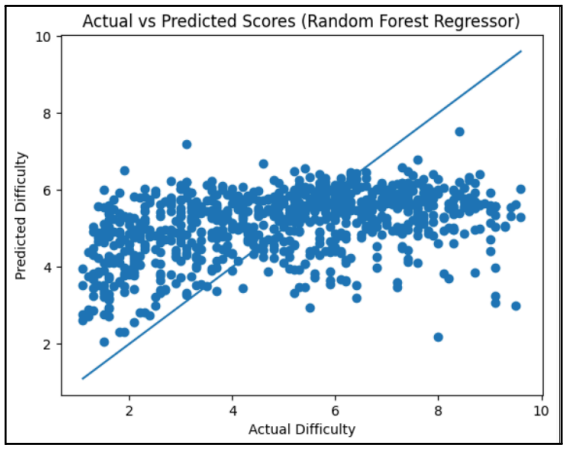
- Achieved an accuracy of approximately **54.1%**, highest among all three models.
- Showed moderate confusion between medium and hard classes.
- Easy problems were sometimes misclassified in the medium or hard category.
- Selected as the final classification model due to overall better performance.

7. Regression Models

7.1 Experimental Setup

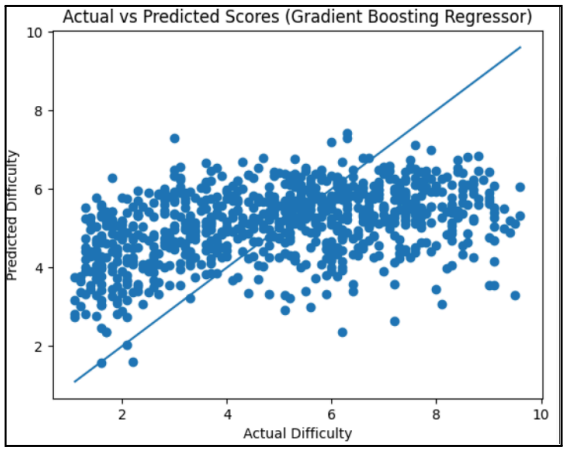
- The same experimental setup as classification was used.
- Each model's performance was evaluated using:
    - Mean Absolute Error (MAE)
    - Root Mean Squared Error (RMSE)

7.2 Models Evaluated



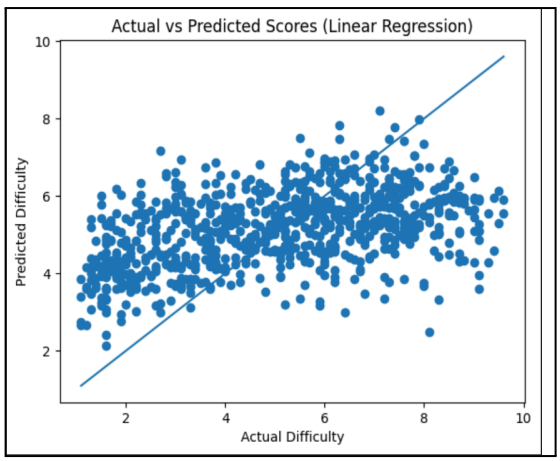
Random Forest Regressor

- MAE: **1.684**
- RMSE: **2.029**
- Struggled to predict very high scores (~ >6)
- Predictions deviate noticeably from the ideal line (y=x).
- Required approximately 2-3 minutes to train the model.



Gradient Boosting Regressor

- MAE: **1.694**
- RMSE: **2.024**
- Predictions follow the general trend slightly better than Random Forest Regressor.
- Errors are more distributed compared to Random Forest Regressor.
- Extreme difficulty scores (>6) are still not predicted accurately.
- Gives slightly better performance than random forest regressor and with lower training time.



Ridge Regression

- MAE: **1.648**
- RMSE: **1.989**
- Predictions are more scattered compared to other two models.
- Concentrates predictions in the middle range, leading to lower accuracy at extreme values.
- Selected as the final regression model due to overall lower MAE and RMSE.

8. Web Interface

A web interface was developed using Streamlit to display the difficulty class and score.

8.1 Interface Features

The interface contains:

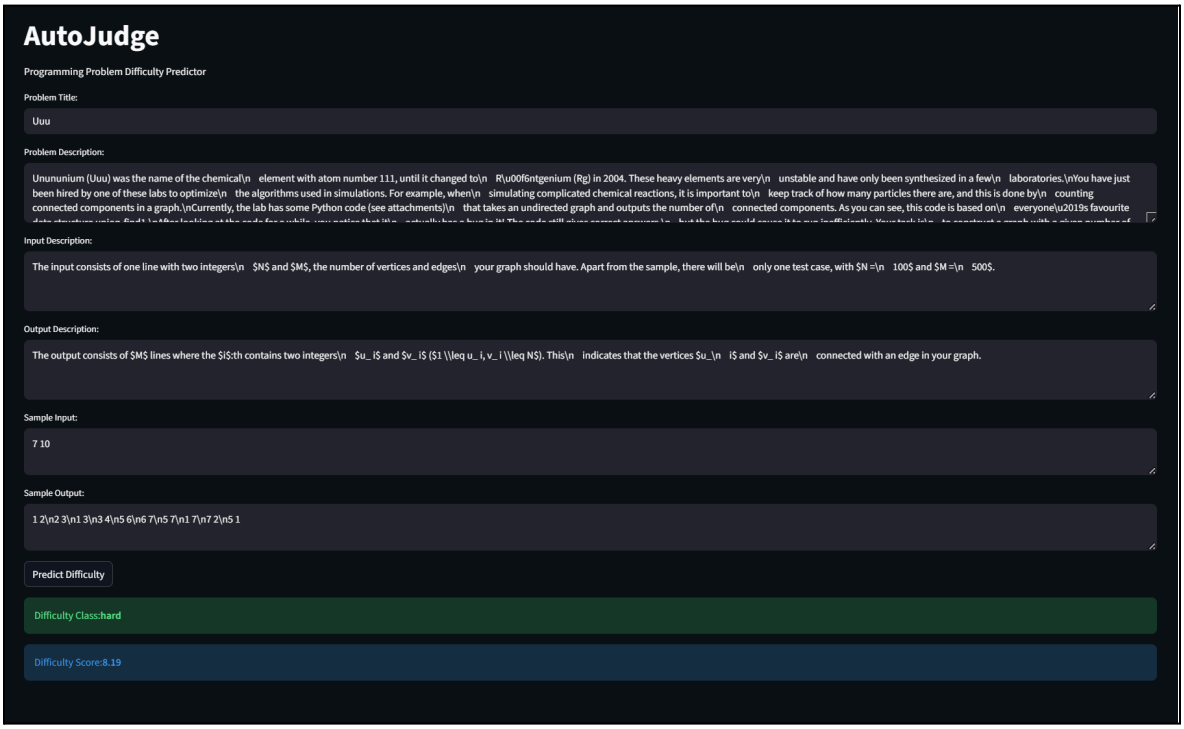
- Input fields for:
    - Title
    - Problem description
    - Input description
    - Output description
    - Sample input/output
- A prediction button to run the model.
- It displays:
    - Predicted difficulty class
    - Predicted difficulty score
- Prevents predictions when no problem description is provided.

8.2 Backend Integration

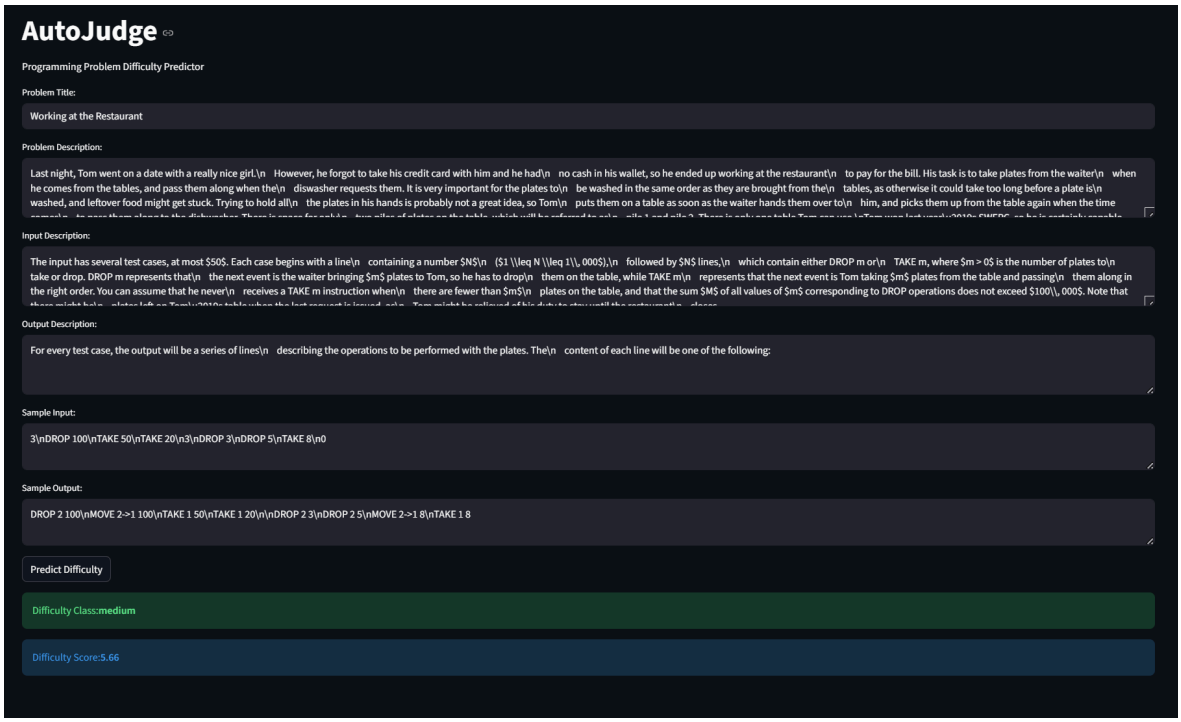
- Loads the trained classification and regression models.
- Applies the same preprocessing and feature extraction steps which were used during training.
- A class-based adjustment is added to the predicted score to make it realistic, and the final score is adjusted to make sure it is in the range 1-10.

- Streamlit web interface

- Dataset problem 1 prediction; Actual(Class:hard,Score:9.7)



- Dataset problem 2562 prediction; Actual(Class:medium,Score:4.3)



Some more sample predictions:

- Dataset problem 590 :Actual(Class:hard,Score:7.6) :Predicted(Class:hard,Score:6.9)
- Dataset problem 1348 :Actual(Class:hard,Score:6.3) :Predicted(Class:hard,Score:6.80)
- Dataset problem 1791 :Actual(Class:hard,Score:5.7) :Predicted(Class:hard,Score:7.30)

- Dataset problem 2180 :Actual(Class:medium,Score:5.0) :Predicted(Class:hard,Score:8.08)
- Dataset problem 2914 :Actual(Class:medium,Score:3.7) :Predicted(Class:medium,Score:5.12)
- Dataset problem 3374 :Actual(Class:easy,Score:2.7) :Predicted(Class:easy,Score:3.87)
- Dataset problem 4110 :Actual(Class:easy,Score:1.1) :Predicted(Class:easy,Score:1.88)

9. Results and Discussion

The results show that:

- The model predicts difficulty classes with reasonable consistency.
- Predicted difficulty scores mostly follow the actual difficulty trend.
- Extreme difficulty scores are harder to predict with accuracy.
- Classification and regression predictions generally go well with each other.

The experimental results show that through textual features alone we can obtain meaningful information to estimate the problem difficulty. The classification models achieved moderate accuracy, with Linear SVM performing slightly better than other two models. Regression models were also able to predict numerical difficulty scores with reasonable error, with the Linear Regression model having lower errors compared to other models, and mostly keeping predictions within a small range (+-2) of the actual values.

The model performs well in identifying hard problems, but some confusion is there between medium and hard difficulty classes, this shows the subjective nature of difficulty labeling. The web interface is easy to use and provides clear and interpretable predictions. Overall, the results show that automated difficulty prediction is possible and can be improved a lot using better features and more advanced models.

Author

Name:Shreyas Dangi

Enrollment Number:24113121