

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.csv), consists of programming problems collected from the platform “Kattis”.

3.1 Dataset Fields

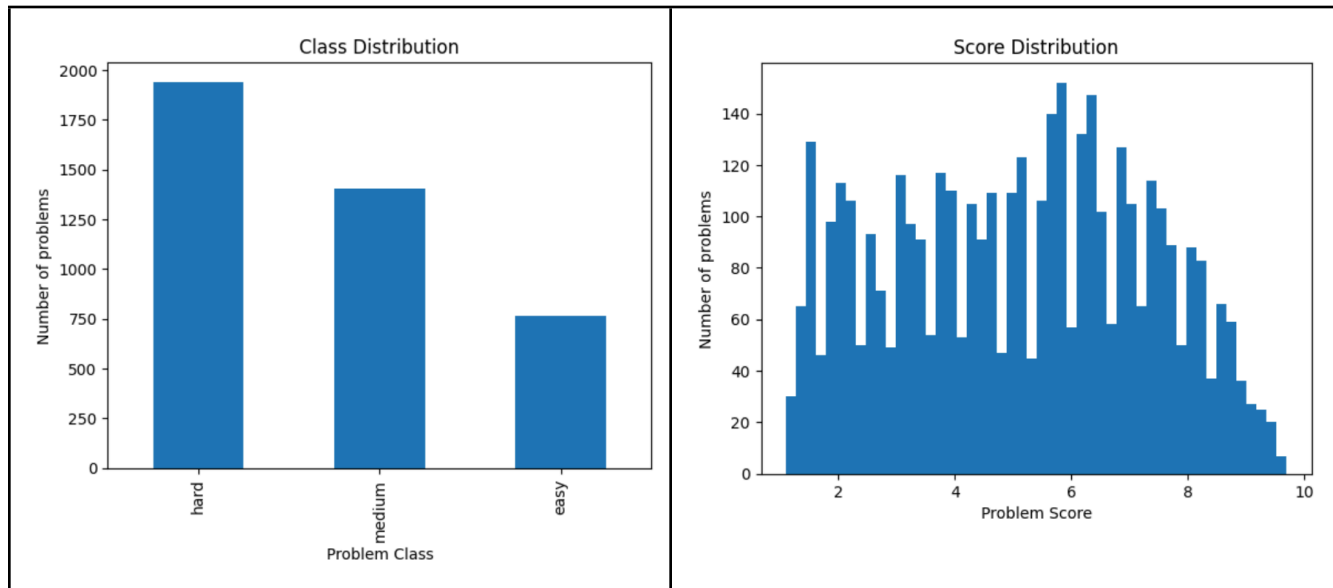
Each data sample includes:

- 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)

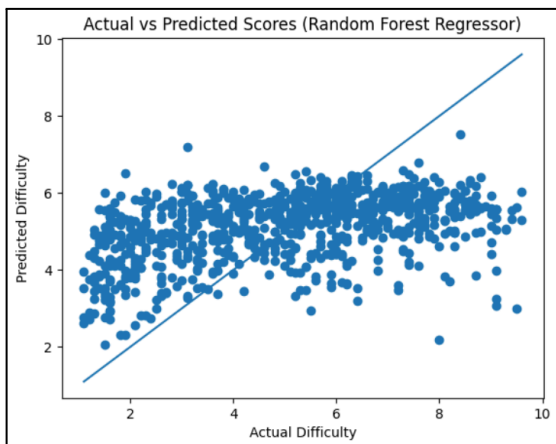
- Achieved an accuracy of approximately **54.1%**, highest among all three models.
- Showed moderate confusion between medium and hard classes.
- Easy problems were sometimes misclassified in the medium or hard category.
- Selected as the final classification model due to overall better performance.

7. Regression Models

7.1 Experimental Setup

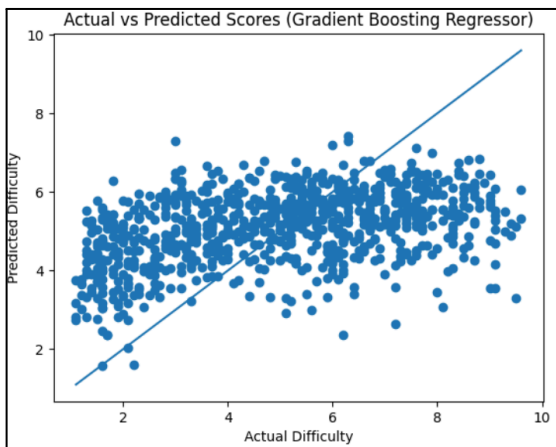
- The same experimental setup as classification was used.
- Each model's performance was evaluated using:
 - Mean Absolute Error (MAE)
 - Root Mean Squared Error (RMSE)

7.2 Models Evaluated



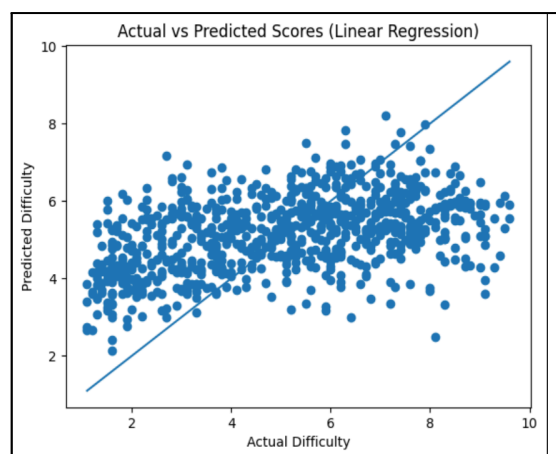
Random Forest Regressor

- MAE: **1.684**
- RMSE: **2.029**
- Struggled to predict very high scores ($\sim >6$)
- 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



The screenshot shows the 'AutoJudge' web interface, which is a 'Programming Problem Difficulty Predictor'. It features a dark-themed layout with several input fields and a prediction button. The inputs are organized into sections: 'Problem Title', 'Problem Description', 'Input Description', 'Output Description', 'Sample Input', and 'Sample Output'. Each section has a label and a corresponding text input area. At the bottom, there is a 'Predict Difficulty' button. The interface is designed to collect user input for a programming problem and then predict its difficulty class and score.

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

AutoJudge

Programming Problem Difficulty Predictor

Problem Title:

Uuu

Problem Description:

Unununium (Uuu) was the name of the chemical element with atom number 111, until it changed to Rutherfordium (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: `Stack`! Write a function that takes an undirected graph as input and outputs the number of connected components. You can use the `Stack` data structure to help you.

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 $N \leq 1005$ and $M \leq 5005$.

Output Description:

The output consists of M lines where the S :th contains two integers u_S and v_S ($1 \leq u_S, v_S \leq N$). This indicates that the vertices u_S and v_S are connected with an edge in your graph.

Sample Input:

7 10

Sample Output:

1 2
2 3
1 3
3 4
5 6
6 7
5 7
1 7
2 5
1 5

Predict Difficulty

Difficulty Class:hard

Difficulty Score:8.19

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

AutoJudge

Programming Problem Difficulty Predictor

Problem Title:

Working at the Restaurant

Problem Description:

Last night, Tom went on a date with a really nice girl. However, he forgot to take his credit card with him and he had no cash in his wallet, so he ended up working at the restaurant to pay for the bill. His task is to take plates from the waiter when he comes from the tables, and pass them along when the dishwasher requests them. It is very important for the plates to be washed in the same order as they are brought from the tables, as otherwise it could take too long before a plate is washed, and leftover food might get stuck. Trying to hold all the plates in his hands is probably not a great idea, so Tom puts them on a table as soon as the waiter hands them over to him, and picks them up from the table again when the time comes to pass them to the dishwasher. There is some food on the table which will be ordered to eat, and with these plates it is not possible to serve the food. The waiter has some plates in his hands, and he is waiting for the dishwasher to take them. You have to write a program that takes the input and outputs the sequence of operations to be performed with the plates. The content of each line will be one of the following:

Input Description:

The input has several test cases, at most 500. Each case begins with a line containing a number N ($1 \leq N \leq 1000$), followed by N lines which contain either `DROP m` or `TAKE m`, where $m > 0$ is the number of plates to take or drop. `DROP m` represents that the next event is the waiter bringing m plates to Tom, so he has to drop them on the table, while `TAKE m` represents that the next event is Tom taking m plates from the table and passing them along in the right order. You can assume that he never receives a `TAKE m` instruction when there are fewer than m plates on the table, and that the sum of all values of m corresponding to `DROP` operations does not exceed 1000. Note that these plates have some food on them and they will be ordered to eat. You have to output the sequence of operations to be performed with the plates.

Output Description:

For every test case, the output will be a series of lines describing the operations to be performed with the plates. The content of each line will be one of the following:

Sample Input:

3
DROP 100
TAKE 50
TAKE 20
DROP 3
DROP 5
TAKE 8
0

Sample Output:

DROP 2 100
MOVE 2->1 100
TAKE 1 50
TAKE 1 20
DROP 2 3
DROP 2 5
MOVE 2->1 8
TAKE 1 8

Predict Difficulty

Difficulty Class:medium

Difficulty Score:5.66

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