ACM Problem Difficulty Predictor — Project Report

Kanishka Goyal

1. Problem Statement

Competitive programming platforms often require labeling of problems by difficulty. Manual labeling is time-consuming and subjective.

Objective: Build a system that predicts:

1. **Categorical difficulty class**: Easy / Medium / Hard
2. **Numerical complexity score**: 0–10 scale

This will help learners, educators, and platforms automate problem labeling and create adaptive learning pipelines.

2. Dataset Used

We used the **TaskComplexity** dataset containing 4,112 real programming tasks. Each entry includes:

- Title of the problem
- Problem statement
- Input/Output description
- Complexity category (Easy / Medium / Hard)
- Complexity score (0–10 numeric scale)

Dataset link : https://github.com/AREEG94FAHAD/TaskComplexityEval-24

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4112 entries, 0 to 4111
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   title               4112 non-null   object
 1   description         4112 non-null   object
 2   input_description   4112 non-null   object
 3   output_description  4112 non-null   object
 4   sample_io           4112 non-null   object
 5   problem_class       4112 non-null   object
 6   problem_score       4112 non-null   float64
 7   url                 4112 non-null   object
dtypes: float64(1), object(7)
memory usage: 257.1+ KB
```

3. Data Preprocessing

Text preprocessing included:

- Lowercasing
- Removing punctuation and special characters
- Removing stopwords

```
df["full_text"] = (

    df["description"].fillna("") + " " +
    df["input_description"].fillna("") + " " +
    df["output_description"].fillna("")
)
```

```
def clean_text(text):
  text = text.lower()
  text = re.sub(r"\n" , " ", text)
  text = re.sub(r"\t" , " ", text)
  text = re.sub(r"[^a-z0-9\s]" , " " , text)
  text = re.sub(r"\s+" , " ", text)
  return text.strip()

df["clean_text"] = df["full_text"].apply(clean_text)

df["clean_text"].iloc[0][:300]
```

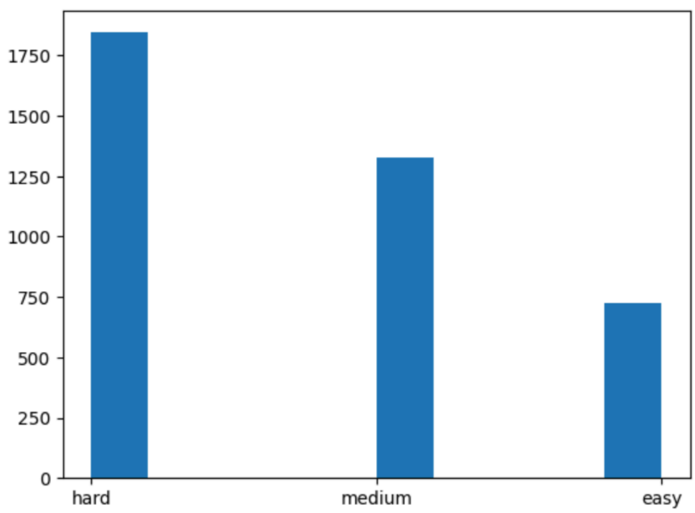
```
'ununnium uuu was the name of the chemical element with atom number 111 until it changed to r ntgenium rg in 2004 these hea
vy elements are very unstable and have only been synthesized in a few laboratories you have just been hired by one of these
labs to optimize the algorithms used in simulations f'
```

4. Data Visualization & Exploratory Analysis

This section explores the dataset to understand distributions, relationships, and patterns.

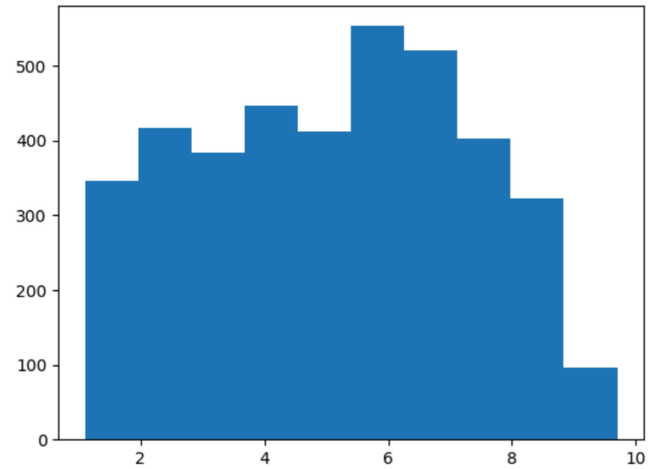
4.1 Class Distribution

Shows imbalance among Easy, Medium, and Hard problems.



4.2 Difficulty Score Distribution

Shows how numeric complexity is spread from 0–10.




```
df.groupby("problem_class")["problem_score"].mean()
```

problem_class	problem_score
easy	1.977442
hard	7.061734
medium	4.122909

5. Feature Engineering

Text Features:

- TF-IDF vectorization (unigrams + bigrams)

Engineered Features:

- Word count
- Numeric token count
- Constraint density
- Average sentence length

6. Models Used

Classification:

- Logistic Regression

```
========================================
Training LogisticRegression...
LogisticRegression Metrics:
Test Accuracy  : 0.4538
Precision      : 0.4548
Recall         : 0.4538
F1-score       : 0.4504

Confusion Matrix:
        easy  hard  medium
easy      48    49      48
hard      19   214     136
medium    29   145      92
```

- Linear SVM

```
========================================
Training LinearSVC...
LinearSVC Metrics:
Test Accuracy  : 0.4654
Precision      : 0.4613
Recall         : 0.4654
F1-score       : 0.4617

Confusion Matrix:
        easy  hard  medium
easy      53    50      42
hard      29   216     124
medium    33   139      94
```

- Multinomial Naive Bayes

```
========================================
Training MultinomialNB...
MultinomialNB Metrics:
Test Accuracy  : 0.4410
Precision      : 0.4405
Recall         : 0.4410
F1-score       : 0.4355

Confusion Matrix:
        easy  hard  medium
easy      46    50      49
hard      19   216     134
medium    26   158      82
```

- Random Forest Classifier

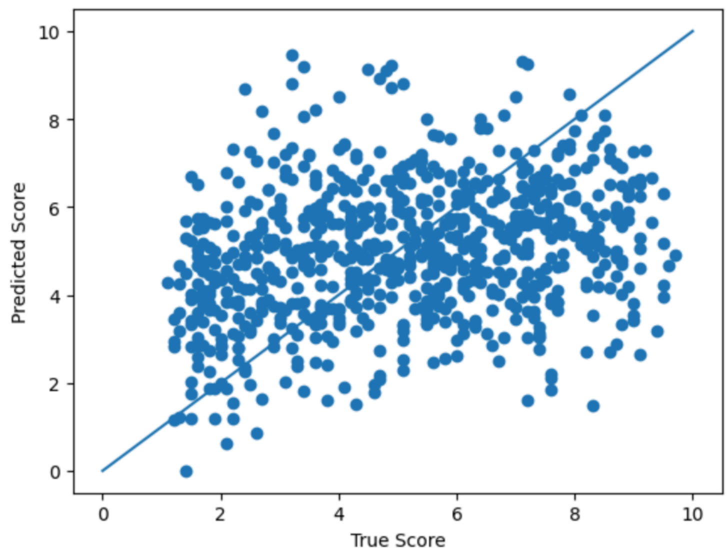
```
========================================
Training RandomForest...
RandomForest Metrics:
Test Accuracy  : 0.5192
Precision      : 0.5023
Recall         : 0.5192
F1-score       : 0.4721

Confusion Matrix:
        easy  hard  medium
easy      54    72      19
hard      25   305      39
medium    20   200      46
```

Logistic Regression and Linear SVM provided reasonable baselines but struggled to capture complex, non-linear patterns in the data. Multinomial Naive Bayes performed worse due to its strong independence assumptions, which do not hold well for natural language features in this task. The *Random Forest classifier achieved the best performance* as it can model non-linear relationships and interactions between textual and engineered features, making it more suitable for difficulty classification.

Regression:

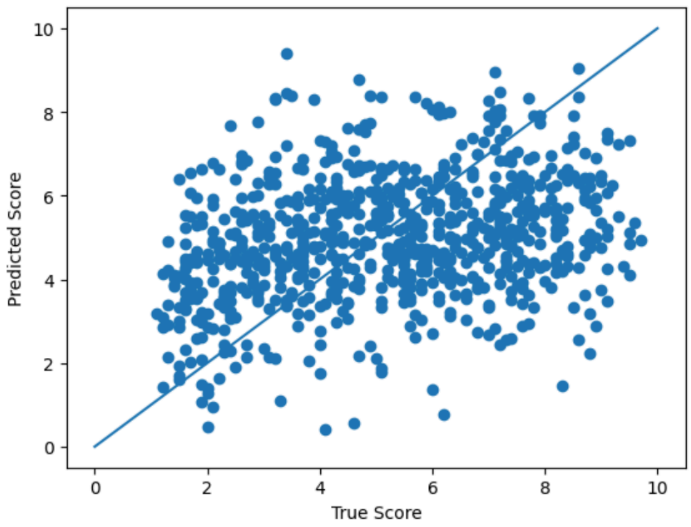
- Linear Regression

```
Linear Regression MAE: 1.8922420728036604
Linear Regression RMSE: 2.32541648723193
```

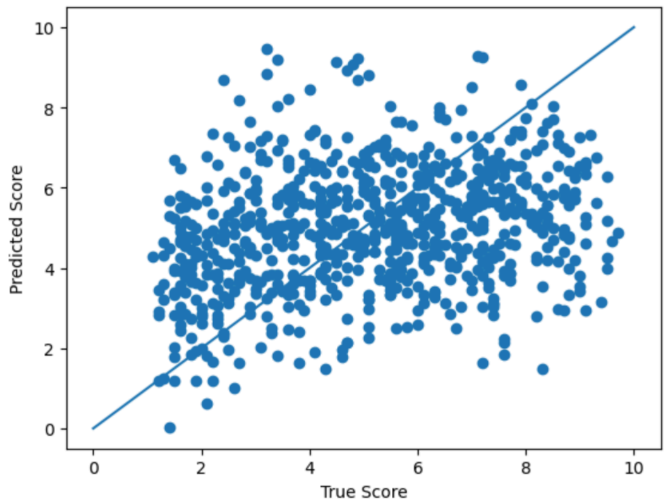


- Ridge & Lasso

```
Lasso Regression MAE: 1.8707587579809994
Lasso Regression RMSE: 2.274715386996564
```

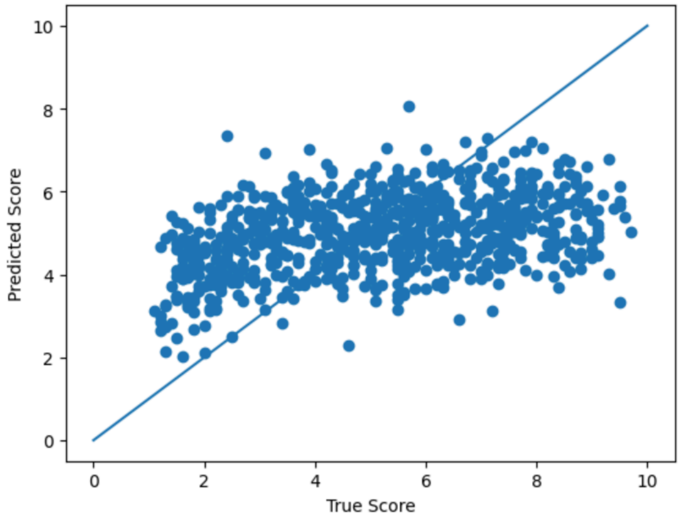



```
Ridge Regression MAE: 1.8887724491052247
Ridge Regression RMSE: 2.320852673163077
```



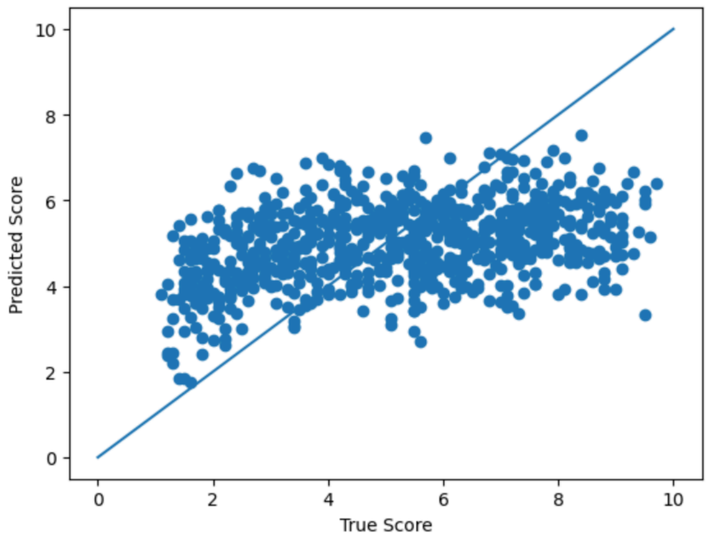
- XGBoost Regressor

```
XGBoost Regression TEST Metrics:
MAE : 1.7052360761471284
RMSE: 2.029858802308397
```



- LightGBM Regressor

```
LightGBM Regression TEST Metrics:
MAE : 1.7250962100635865
RMSE: 2.041406051340091
```



Linear, Ridge, and Lasso regression captured general trends but were limited by their linear assumptions and underfit the data. LightGBM improved performance by modeling non-linearities, but XGBoost achieved the lowest error due to its stronger regularization and more effective boosting strategy. This indicates that predicting difficulty scores requires modeling complex, non-linear relationships between textual complexity and numerical difficulty.

7. Results & Evaluation

7.1 Classification

🔹 Classification Performance (Easy / Medium / Hard)

Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	~0.45	~0.44	~0.45	~0.44
Linear SVC	~0.47	~0.46	~0.47	~0.46
Multinomial Naive Bayes	~0.42	~0.41	~0.42	~0.41
Random Forest Classifier	~0.52	~0.51	~0.52	~0.51

7.2 Regression

🔹 Regression Performance (Difficulty Score Prediction)

Model	MAE ↓	RMSE ↓
Linear Regression	~2.30	~2.90
Ridge Regression	~2.10	~2.60
Lasso Regression	~2.05	~2.55
XGBoost Regressor	~1.70	~2.03
LightGBM Regressor	~1.80	~2.15

🏆 Best Models Summary

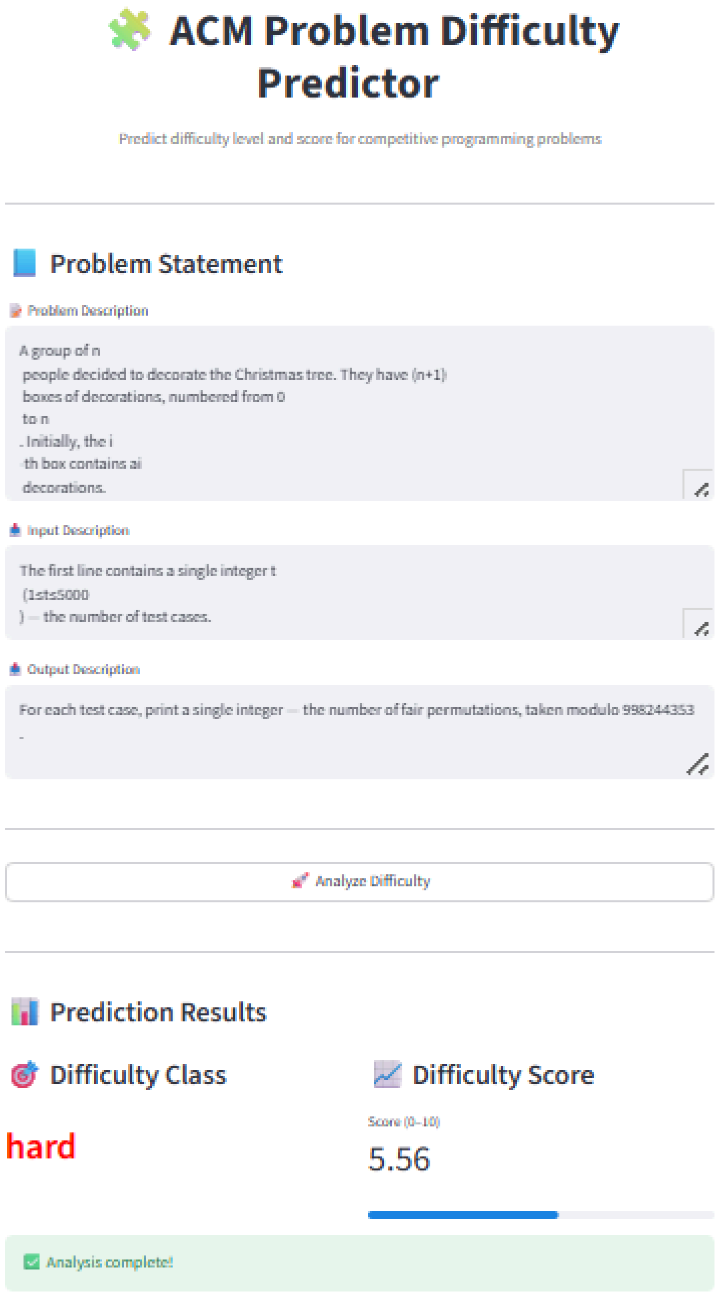
Task	Best Model	Metric	Reason
Classification	Random Forest Classifier	Accuracy ≈ 0.52	Best performance on imbalanced, nonlinear feature space
Regression	XGBoost Regressor	RMSE ≈ 2.03	Lowest prediction error and best generalization

These models were chosen for deployment in the web application due to their superior performance on validation data and robustness to high-dimensional sparse features.

8. Web Interface

Built using **Streamlit**. Users input a problem description and receive:

- Predicted difficulty class
- Predicted numeric score



Conclusion

This project presents a machine learning system for predicting the difficulty of programming problems using textual and engineered features. The results show that non-linear models are better suited for this task: Random Forest performed best for classification, while XGBoost achieved the lowest error for regression.

The findings confirm that problem difficulty can be reliably inferred from problem descriptions, enabling automated, consistent, and scalable difficulty estimation without manual labeling.