AutoJudge: Predicting Programming Problem Difficulty

Nagam Amareswar
BS-MS Physics
23324013

1. Problem Statement

Online coding platforms such as Codeforces, CodeChef, and Kattis classify programming problems into difficulty levels like Easy, Medium, and Hard. They also assign a numerical difficulty score(0 to 10) to each problem. At present, this classification is mainly done using human judgment and user feedback, which can be subjective, slow, and sometimes inconsistent.

The goal of this project, AutoJudge, is to build an automated system that can predict the difficulty of a programming problem using only its textual description. The system uses Natural Language Processing (NLP) and Machine Learning (ML) techniques to remove human bias and provide consistent results.

The system performs two tasks. It predicts:
Problem Class: whether a problem is Easy, Medium, or Hard
Problem Score: numerical score that represents the problem's difficulty

Both predictions are made only from text, without using extra information such as user ratings or submission data.

The text used for prediction includes:

- Problem description
- Input description
- Output description

2. Dataset Description

The dataset used in this project is Task Complexity Dataset, provided as part of the project guidelines. Each data sample represents a single programming problem and contains both textual information about the problem and the target variables.

The dataset contains multiple columns:

- **Title** - short name of the problem
- **Problem Description** - detailed explanation of the problem
- **Input Description** - description of the input format

- **Output Description** - description of the expected output
- **Sample I/0** - examples to clarify the task requirements
- **Problem Class** - difficulty label (Easy / Medium / Hard)
- **Problem Score -** task difficulty score(0 to 10)

The dataset is used as it is, and no external data sources are used.

3. Data Preprocessing

3.1 Data Loading and Initial Checks

The dataset is loaded and examined for missing values. All text columns are checked for **None** and **NaN** values. Any empty strings found in the dataset are replaced with the placeholder **“missing info”** to ensure consistency and prevent errors during text processing.

3.2 Text Cleaning

All textual columns were cleaned in the following way:

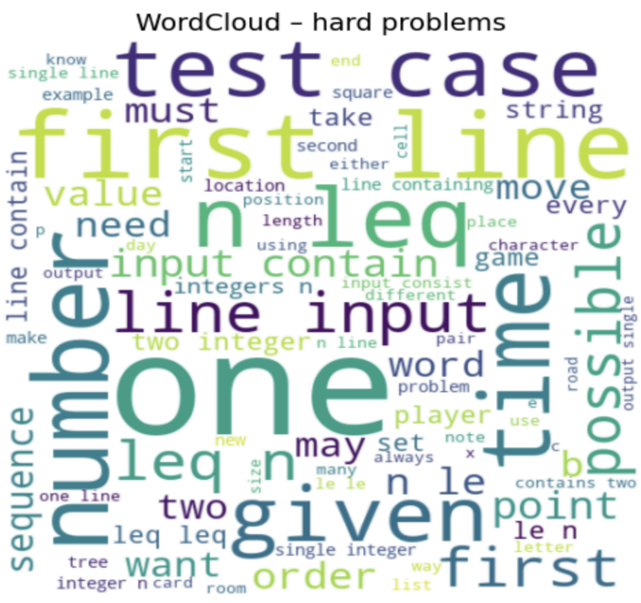
- Converting all text to lowercase
- Removing special characters while preserving mathematical symbols
- Removing newline characters
- Removing english stopwords

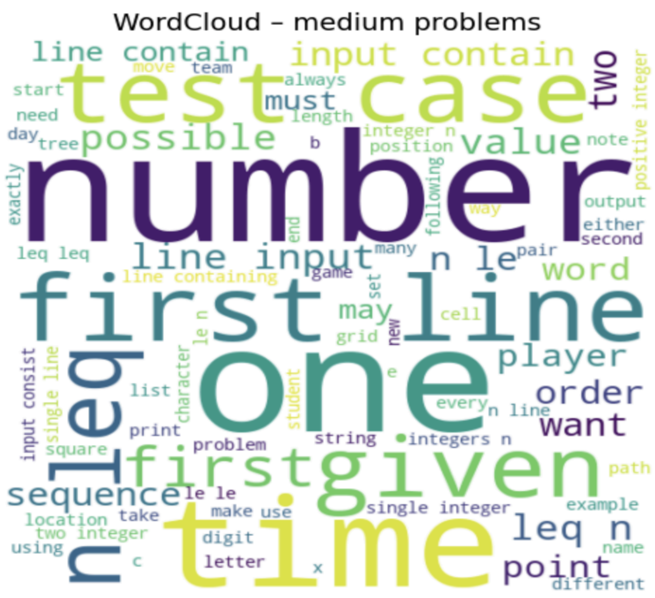
After cleaning, the title, problem description, input description, and output description columns are combined into a single column name “**text”** for further feature extraction.

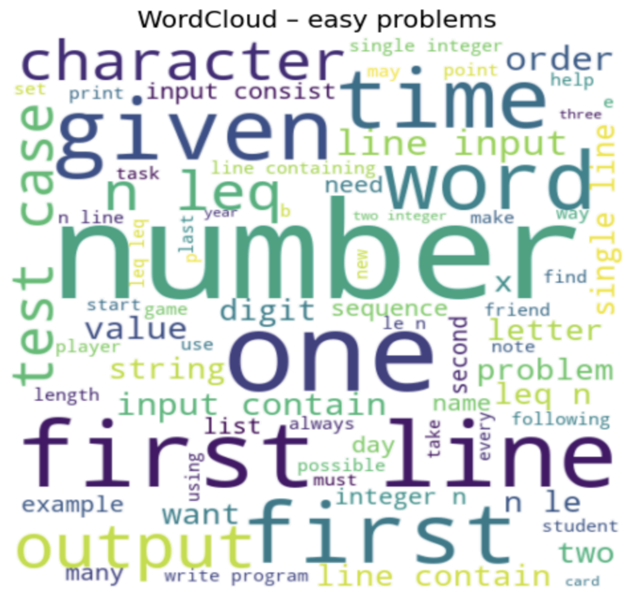
3.3 WordCloud Analysis

To gain an initial understanding of the textual data, WordClouds were generated for the cleaned text of each difficulty class (Easy, Medium, and Hard).

The WordClouds visualize the most frequently occurring words within each class and help identify any class-specific patterns.







The analysis showed that **most high-frequency words are common across all three difficulty levels**, with no clearly distinct vocabulary separating one class from another. This indicates that simple word frequency–based features are insufficient for accurately distinguishing problem difficulty.

As a result, more informative text representations such as **TF-IDF** are required to capture the relative importance of words and improve both classification and regression performance.

4. Feature Engineering

4.1 Keyword-based Feature

A predefined list of algorithm-related keywords (e.g., graph, dp, recursion, greedy) was used. For each problem, the total number of keyword occurrences was computed and used as a single feature. This provides a robust summary of algorithmic complexity.

4.2 Structural Features

The following structural features were extracted from the combined text:

- Text length (number of words)
- Number of mathematical symbols
- Count of conditional statements (if)
- Count of loop indicators (for, while)

These features help represent logical and computational complexity.

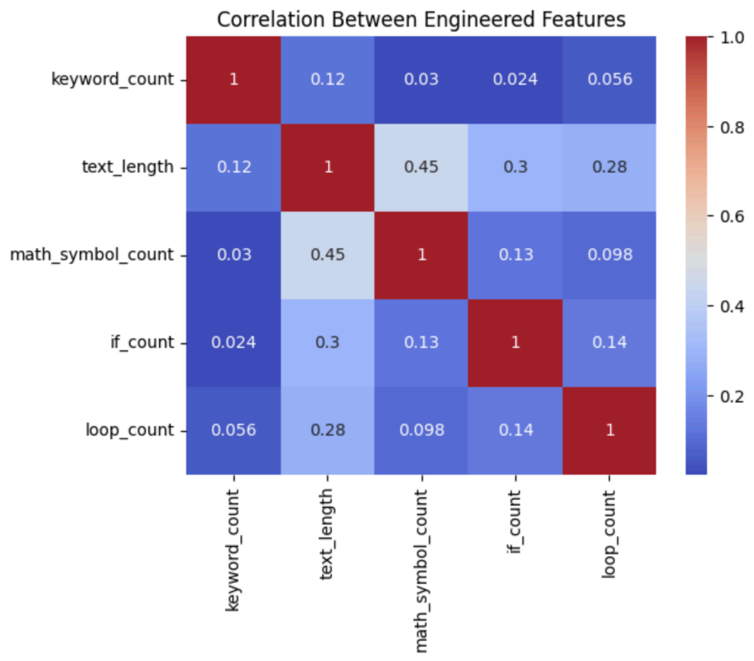
The keyword and structural features were merged into a single dataframe called **engineered_features_df** for use in model training.

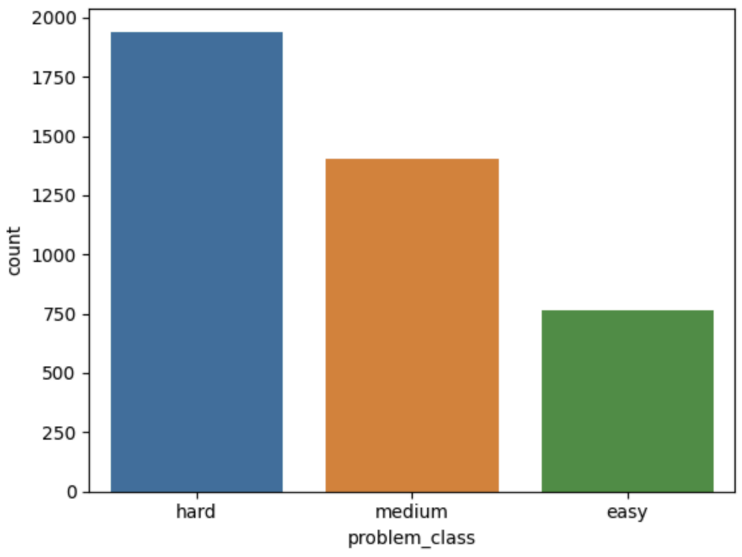
5. Data Visualisation

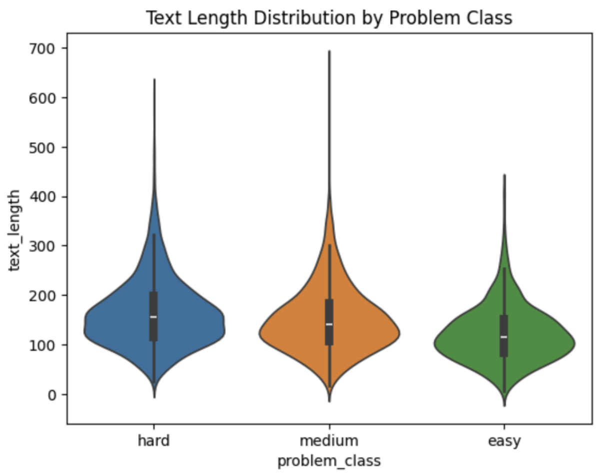
Data visualization was performed to better understand the relationship between the engineered features and problem difficulty.

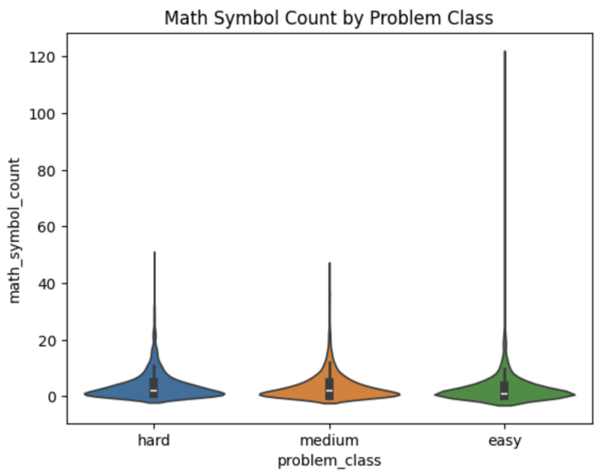
The following visualizations were created:

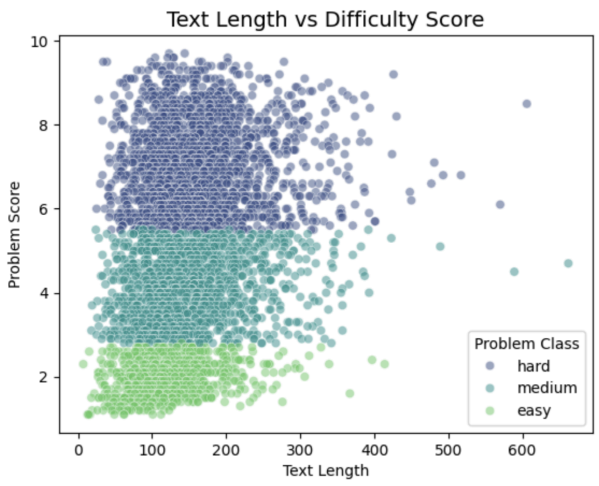
- **Correlation heatmap** of engineered features to examine inter-feature relationships and redundancy.
- **Class distribution plot** to observe imbalance among Easy, Medium, and Hard problems.
- **Violin plots** showing how engineered features such as text length, and mathematical symbol count vary across difficulty classes.
- **Scatter plots** illustrating the relationship between text length and the numerical difficulty score.

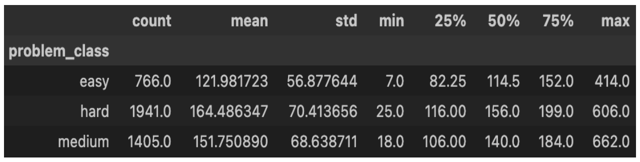

Correlation Between Engineered Features
keyword_count
text_length
math_symbol_count
if_count
loop_count
1 0.12 0.03 0.024 0.056
0.12 1 0.45 0.3 0.28
0.03 0.45 1 0.13 0.098
0.024 0.3 0.13 1 0.14
0.056 0.28 0.098 0.14 1
1.0
0.8
0.6
0.4
0.2


count
problem_class
hard
medium
easy
2000
1750
1500
1250
1000
750
500
250
0


Text Length Distribution by Problem Class
text_length
problem_class
hard
medium
easy
700
600
500
400
300
200
100
0


Math Symbol Count by Problem Class
math_symbol_count
problem_class
hard
medium
easy
120
100
80
60
40
20
0





	count	mean	std	min	25%	50%	75%	max
problem_class								
easy	766.0	121.981723	56.877644	7.0	82.25	114.5	152.0	414.0
hard	1941.0	164.486347	70.413656	25.0	116.00	156.0	199.0	606.0
medium	1405.0	151.750890	68.638711	18.0	106.00	140.0	184.0	662.0

Some Observations:

- The dataset is imbalanced, with Hard problems appearing more frequently than Medium and Easy problems.
- The correlation heatmap shows no strong correlation among engineered features, indicating that each feature contributes unique information.
- Hard and Medium problems generally have longer text descriptions than Easy problems (mean and medium text length is greater for hard problems than medium and easy problems).
- No single feature is sufficient on its own, but combining multiple engineered features provides meaningful signals for difficulty prediction**.**

6. Feature Extraction(Vectorization)

The **text** column was transformed into numerical features using **TF-IDF Vectorizer** (Term Frequency–Inverse Document Frequency)**.** TF-IDF assigns higher importance to terms that are informative for distinguishing problem difficulty.

The final feature vector for each problem was created by **concatenating**:

- TF-IDF features
- Engineered keyword and structural features

7. Model Selection Using Cross-Validation

For predicting problem class and problem score, several classification and regression models were evaluated using **cross_val_score** to determine which model best fit the data.

Cross-validation was chosen because it provides a more reliable estimate of performance than a single train-test split. The model with the **highest average cross-validation score** was selected as the final model.

For classification **Logistic Regression, LinearSVC, Complement Naive Bayes and Random Forest Classifier** are evaluated using **Accuracy score** metric.

For regression **Linear Regression, Lasso Regression, Random Forest Regressor and Gradient Boosting Regressor** are evaluated using **Root Mean Square Error** metric.

In both Classification and Regression, the **Random Forest** model achieved the best cross validation score.

8. Model Training and Evaluation

After model selection using cross-validation, the chosen random forest models were trained on the training dataset using TF-IDF features with bigrams. The trained models were evaluated using :

- Accuracy(52.247% on test data) for classification
- RMSE (2.027 on test data) for regression

The trained TF-IDF vectorizer, classifier, and regressor were saved using the **joblib** library to enable reuse for building web interface.

9. Web Interface

A simple web interface was built using **Streamlit** that allows users to enter the **problem description, input description, and output description**. The input text is cleaned and processed using the same feature engineering and TF-IDF vectorization steps as in training, and the trained models are used to predict the difficulty class and difficulty score(clipped to range between 0 and 10).

10. Conclusion

In this project, an automated system named **AutoJudge** was developed to predict the difficulty of programming problems using only their textual descriptions. The system addresses both **classification** (Easy, Medium, Hard) and **regression** (task difficulty score) tasks by applying **NLP** and **classical ML** techniques.

Overall, the AutoJudge system demonstrates the practical applicability of machine learning for automating problem evaluation in competitive programming platforms. With further improvements such as **larger datasets**, advanced feature extraction, or **transformer-based embeddings**, the system can be extended to achieve even higher accuracy and robustness.