AutoJudge: Predicting Programming Problem Difficulty

1. Problem Statement

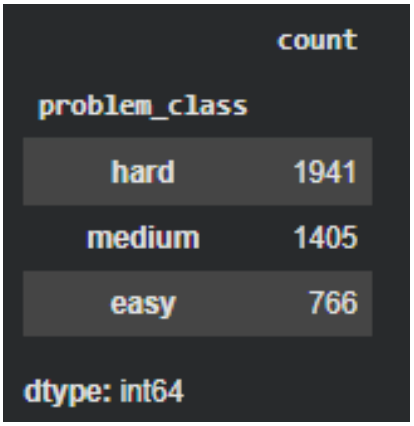
Online coding platforms (like Codeforces, Kattis, CodeChef) classify programming problems as **Easy, Medium, or Hard**, and also assign a **difficulty score**.
However, this process usually depends on human judgment and user feedback. In this Project we aim to built an intelligent system that can predict Problem Class(Easy/Medium/Hard) along with Problem score ranging between 1.1 to 9.7 as per the task complexity dataset. These can be categorised into classification and a regression problem

2. Dataset Used

The Dataset used is the Task complexity Dataset the one mentioned in the problem statement word document. The dataset consists of 4112 rows and 8 columns.
The columns are as follows:

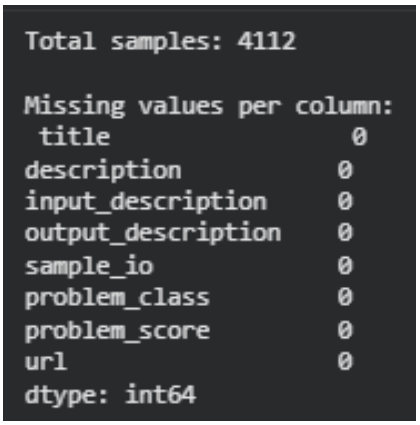
```
Index(['title', 'description', 'input_description', 'output_description',
       'sample_io', 'problem_class', 'problem_score', 'url'],
      dtype='object')
```

There are no null rows. All columns except the problem_score which is float type are object type consisting of texts in majority cases where as the sample input output consists of numbers,brackets in form of lists.
The problem_score ranges from 1.1 to 9.7.
The problem_class consist of 3 values “Easy”,”Medium” and “Hard” with the following values:



problem_class	count
hard	1941
medium	1405
easy	766

dtype: int64



```
Total samples: 4112

Missing values per column:
 title                 0
description            0
input_description      0
output_description     0
sample_io              0
problem_class          0
problem_score          0
url                    0
dtype: int64
```

3. Data Preprocessing

The problem title, description, input description, and output description were concatenated to form a unified semantic text representation. These fields contain natural language explanations that describe the task requirements, constraints, and expected outputs, making them suitable for semantic analysis. Minimal text cleaning was applied to avoid removing algorithmic keywords such as graph, dynamic programming, and recursion, which are strongly indicative of problem complexity.

The dataset also includes a `sample_io` field containing example inputs and outputs. Exploratory analysis revealed that this field is stored as a list and primarily consists of numeric values, brackets, and formatting symbols rather than natural language. Direct inclusion of this field in text vectorization resulted in sparse and noisy representations. Therefore, instead of treating `sample_io` as raw text, it was processed separately to extract structural complexity features. This separation ensures that semantic and structural information are handled appropriately and reduces noise in the learned representations.

All preprocessing steps were performed consistently across the training and test sets to avoid information leakage and ensure fair evaluation.

4. Feature Extraction

To capture multiple dimensions of programming problem difficulty, a hybrid feature extraction strategy was adopted, combining semantic, engineered, and structural features.

Semantic features were extracted using TF-IDF vectorization applied to the consolidated problem text. TF-IDF effectively captures the importance of algorithmic terms and conceptual keywords that frequently correlate with higher difficulty levels. Separate TF-IDF vectorizers were trained for classification and regression tasks to maintain feature consistency during inference.

In addition to TF-IDF, engineered semantic features were derived from the problem text to provide interpretable signals. These include text length as a proxy for problem detail, the frequency of mathematical symbols indicating formula-heavy or constraint-driven tasks, and the occurrence of algorithmic keywords such as graph, dp, tree, and greedy. These features complement TF-IDF by capturing structural patterns that may not be emphasized by purely statistical text representations.

Structural features were extracted from the `sample_io` field to represent input–output complexity. Since example inputs and outputs often reflect parsing difficulty and

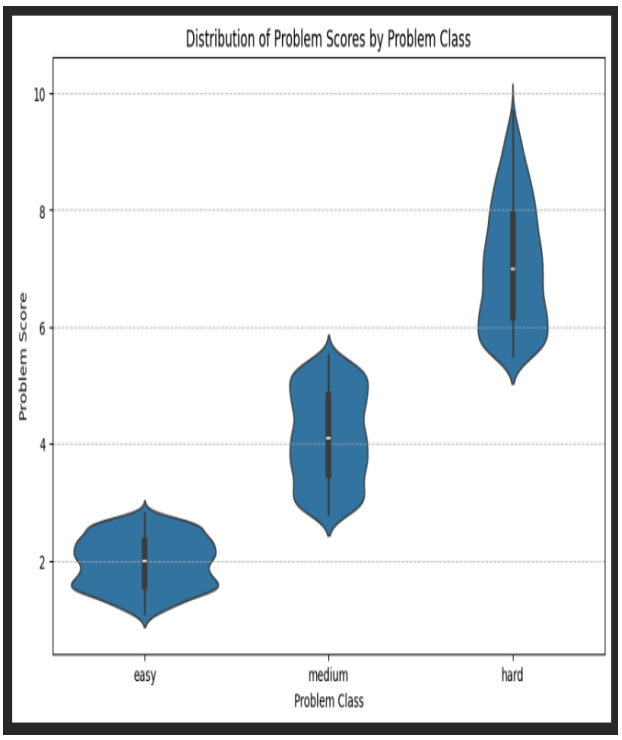
implementation complexity rather than semantic content, explicit structural signals were computed instead of applying TF-IDF. These include the number of numeric values, number of lines, number of brackets, length of the largest numeric token, and the presence of matrix-like patterns. Together, these features capture the formatting and structural demands imposed by the problem.

The final feature representation combines TF-IDF features with scaled engineered semantic and structural features. This hybrid approach allows the model to jointly reason about linguistic complexity, algorithmic indicators, and input–output structure, resulting in a more robust and expressive representation of programming problem difficulty.

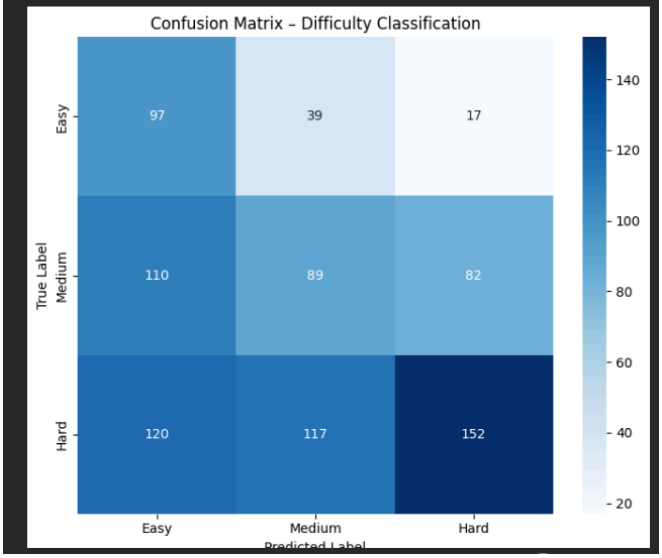
5. Model Training

For the classification Problem 3 models were used namely Logistic Regression,SVM and Random forest classifier. The methodology involved combining the tfidf vectors and the engineered features consisting of structural features of sample_io like has_matrix,num_lines,max_num_length etc. and the semantic features like mathematical symbols, keywords like dp(dynamic programming),graph etc. and length of the X_train which is dataframe of the full_text on each row combined on terms of features like title,description,input and output description. Initially only semantic features were incorporated later realising that structural features of sample_io also plays a crucial role in determining the difficulty class and score as it increased the accuracy of all the models. Also different tfidf vectors was used in for Random classifier as X_train_tfidf_rf the max_features parameter of tfidf was lowered which performed better from the initial case.The model performance were evaluated based on accuracy and confusion matrix which are as follows for all the 3 models. From the images it is clear that random classifier performs best with accuracy of 0.53462 and based on confusion matrix logistic regression predicts lot more fake easy,mediums and hards,svm predicts a lot of fake hards and mostly hards only which leads the model to be biased to one specific class so its not chosen the random classifier has somewhat unbiased confusion matrix in comparison to other models it although predicts many mediums as hards this can catered by setting threshold based on the regression model prediction of problem score as both these have close relation so the ranges of existed dataset were analysed so as to decide a threshold for each class. As clear from below violin graph that based on problem score the classes can be assigned but it resutled in poor accuracy in both the cases score based class prediction which was based on threshold and also the one involving hybrid setup wherein initial classification was done and based on predicted score and threshold classes were modified but this resulted in poor accuracy as well so it was decided to

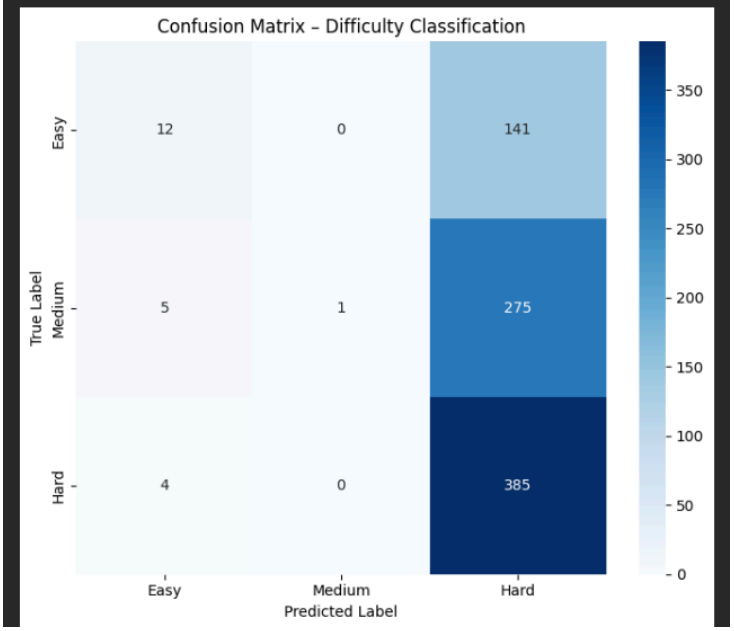
go ahead with random classifiers as the prediction model.



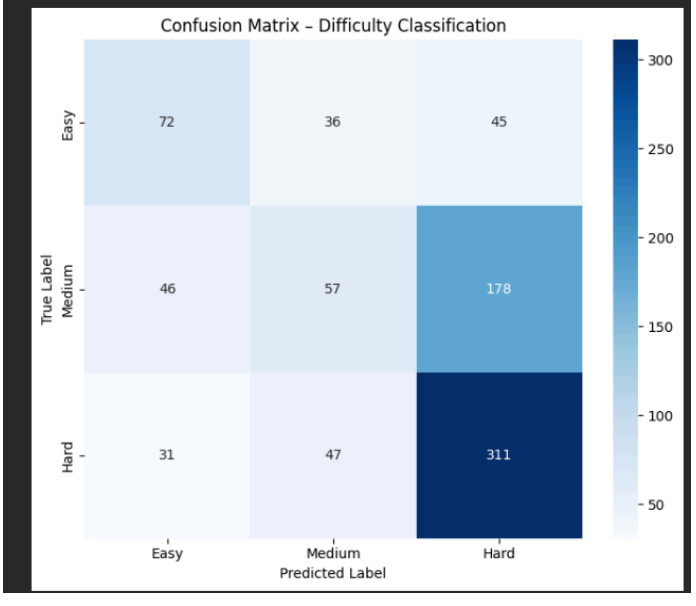

```
Logistic Regression Accuracy: 0.4106925880923451
              precision    recall  f1-score   support

        easy       0.30      0.63      0.40       153
        hard       0.61      0.39      0.47       389
      medium       0.36      0.32      0.34       281

    accuracy                           0.41       823
   macro avg       0.42      0.45      0.41       823
weighted avg       0.47      0.41      0.42       823
```




```
SVM Accuracy: 0.4835965978128797
              precision    recall  f1-score   support

        easy       0.57      0.08      0.14       153
        hard       0.48      0.99      0.65       389
      medium       1.00      0.00      0.01       281

    accuracy                           0.48       823
   macro avg       0.68      0.36      0.26       823
weighted avg       0.67      0.48      0.33       823
```




```
Random Forest Accuracy: 0.534629404617254
              precision    recall  f1-score   support

        easy       0.48      0.47      0.48       153
        hard       0.58      0.80      0.67       389
      medium       0.41      0.20      0.27       281

    accuracy                           0.53       823
   macro avg       0.49      0.49      0.47       823
weighted avg       0.50      0.53      0.50       823
```



The problem score prediction also used combined tfidf vectors and the engineered features as their input and 3 models were evaluated based on RMSE and MAE. The models were LinearRegression,RandomForestRegressor and GradientBoostRegressor.

```
Linear Regression MAE: 5.254112271575798
Linear Regression RMSE: 6.853036349070965
Random Forest MAE: 1.7040587200132407
```

```
Random Forest RMSE: 2.0462015585703917
Gradient Boosting MAE: 1.6982901407384212
Gradient Boosting RMSE: 2.0379786401063114
```

The gradient boosting regressor have least rmse and mae so its parameters were chosen for problem range prediction.

6. Web Interface UI (Streamlit)

A lightweight web-based interface was developed using **Streamlit** to allow interactive difficulty prediction.

Interface Functionality

The UI allows users to:

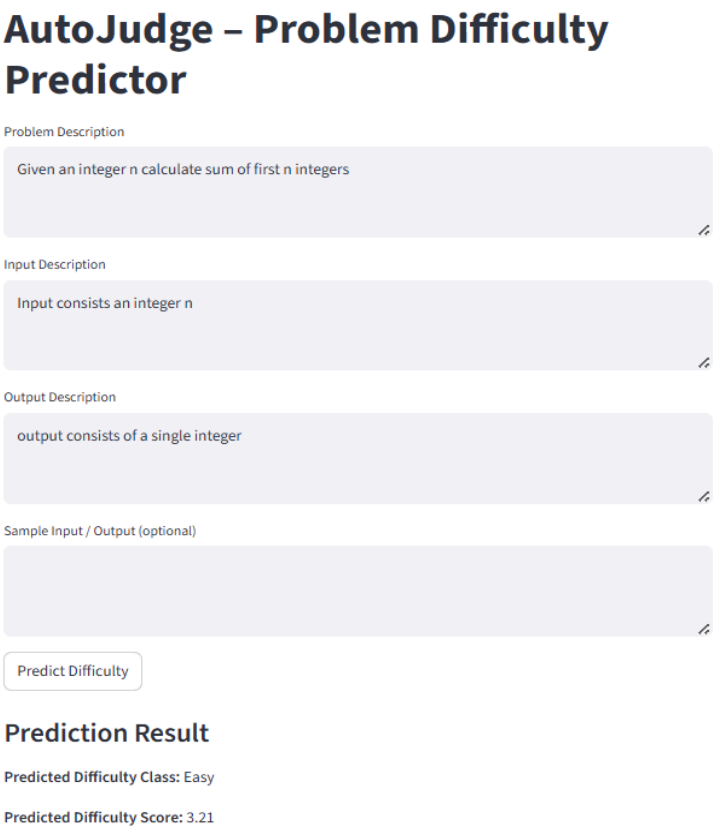
- Paste:
  - Problem description
  - Input description
  - Output description
  - Sample input/output (optional)
- Click a **Predict Difficulty** button
- View:
  - Predicted difficulty class (Easy / Medium / Hard)
  - Predicted difficulty score (continuous)

The interface does not require authentication or a database and is designed to be easily runnable on a local machine using the provided environment setup.

7. Sample Predictions

To qualitatively evaluate the behavior of the proposed system, representative programming problems of varying complexity were tested through the deployed Streamlit interface. Simple arithmetic problems with minimal input structure and no algorithmic keywords were classified as Easy and assigned low difficulty scores, reflecting their straightforward nature. Problems involving classical dynamic programming techniques, such as maximum subarray sum or tree traversal, were classified as Medium with moderate difficulty scores, capturing both algorithmic intent and moderate input complexity. Structurally complex problems involving weighted graphs, multiple constraints, and large input sizes were classified as Hard and assigned high difficulty scores. These examples demonstrate that the model effectively captures both semantic indicators (algorithmic keywords) and structural characteristics (input size and formatting), resulting in stable and interpretable predictions across

difficulty levels.However since the class predictions and difficulty score are separate so in certain instances it might appear they both might not agree specially problems in the boundary range but it is due to different patterns being read by both the models and as explained above while trying to correlate both the accuracy was compromised so that was omitted and individual predictions are done.Example Predictions are as follows.



AutoJudge – Problem Difficulty Predictor

Problem Description

If the maximum score and the minimum score differ by at least 10 points, print check again (the judging seems inconsistent, so the panel must re-evaluate).
Otherwise, print final X, where X is the median of the three scores (the score that would be in the middle if all three were sorted in non-decreasing order).

Input Description

A single line contains three integers g,c,ℓ
, representing the scores of Gemini, ChatGPT, and Claude respectively.

80≤g,c,ℓ≤100

Output Description

Print the required answer in a line.

Sample Input / Output (optional)

Predict Difficulty

Prediction Result

Predicted Difficulty Class: Medium

Predicted Difficulty Score: 4.37

AutoJudge – Problem Difficulty Predictor

Problem Description

Greta and Alice are the two permanent hosts of the hit comedy show "QuestExpert". For this season they invited n
programmers to complete quests, set by Alice. After that they all meet in a studio to review how well they did and complete the final studio quest

Input Description

It is guaranteed that the sum of n
over all test cases does not exceed 5·105
.

Output Description

For each test case, output on a separate line the minimum number of rounds that could lead to the given scores.

Sample Input / Output (optional)

2
4
10

Predict Difficulty

Prediction Result

Predicted Difficulty Class: Hard

Predicted Difficulty Score: 5.89

8. Conclusion

In this project, an automated system for predicting programming problem difficulty was developed using a combination of machine learning techniques and engineered features. By integrating TF-IDF-based semantic representations with carefully designed semantic and structural features, the system successfully models both the conceptual and input-level complexity of problems. Extensive experimentation showed that a Random Forest classifier provides the most reliable difficulty classification, while a Gradient Boosting regressor offers meaningful continuous difficulty estimates. The final design separates classification and regression responsibilities to ensure stability and interpretability. The deployed Streamlit interface demonstrates the practicality of the approach, enabling real-time difficulty assessment. Overall, the system provides a robust and extensible framework for automated difficulty estimation, with potential applications in online judges, educational platforms, and problem recommendation systems.