REPORT ON PROJECT: AUTOJUDGE

NAME: CHIRANSHU SARRAF

ENROLLMENT NUMBER: 24117041

Section 1. Introduction

Competitive programming platforms host a large collection of algorithmic problems that vary widely in difficulty. For users , these platforms assign a numerical difficulty rating to each problem, allowing users to select problems appropriate to their skill level.

On platforms such as Codeforces, difficulty ratings are derived from historical user performance data, including submission statistics and solution success rates. While this approach is effective, it relies on the availability of sufficient user interaction data. For newly created problems or problems that have not yet been attempted by many users, reliable difficulty estimation becomes challenging.

An alternative perspective is to examine whether difficulty can be inferred directly from the problem statement itself. Problem descriptions often contain implicit signals related to complexity, such as required algorithmic techniques, constraints, and input–output structure. Extracting and interpreting these signals using natural language processing techniques presents an interesting machine learning challenge.

This project explores the feasibility of predicting competitive programming problem difficulty using only textual information, without relying on any user statistics or platform-specific metadata. The focus is on classical machine learning methods applied to problem statements, with the goal of producing both categorical difficulty labels and numerical difficulty scores.

To this end, we present AutoJudge, a text-based difficulty prediction system that combines natural language feature extraction with supervised learning models. The following sections describe the formal problem statement, dataset selection, preprocessing pipeline, modeling approach, experimental evaluation, and the deployed web interface.

Section 2. Problem Statement

The objective of this project is to design and implement an automated system that predicts the difficulty of competitive programming problems using only their textual problem statements.

Given a programming problem described by its title, problem description, input format, and output format, the system is required to perform the following two tasks:

1. Difficulty Classification:
   Predict a categorical difficulty label for the problem, belonging to one of the classes *Easy*, *Medium*, or *Hard*.

2. Difficulty Score Prediction:
   Predict a numerical difficulty score that represents the estimated complexity level of the problem.

The system must operate under the constraint that no user performance data, submission statistics, or platform-specific metadata are used. All predictions must be derived exclusively from the textual content of the problem statement.

The final solution should be capable of processing unseen problem statements and producing consistent difficulty estimates that align with commonly accepted difficulty ranges in competitive programming.

Section 3: Dataset Description

The dataset used in this project consists of competitive programming problems taken from the Codeforces platform and made available through Hugging Face. It contains real-world programming problems along with their corresponding textual descriptions and numerical difficulty ratings.

During the initial phase of the project, the dataset suggested in the problem statement was explored through preliminary experimentation. However, several limitations were observed when using this dataset under a strict text-only setting:
• The classification accuracy obtained was approximately 50%, indicating limited predictive capability.
• Model predictions were strongly biased toward a single difficulty class, with most problems being classified as Medium.
• The dataset exhibited weak class separability when relying solely on textual features, resulting in poor generalization on unseen samples.

Due to these limitations, the initially suggested dataset was not used for the final implementation, as the objective of this project is reliable difficulty estimation based only on problem statements.

The dataset used for the final system was obtained from the following source:
https://huggingface.co/datasets/open-r1/codeforces

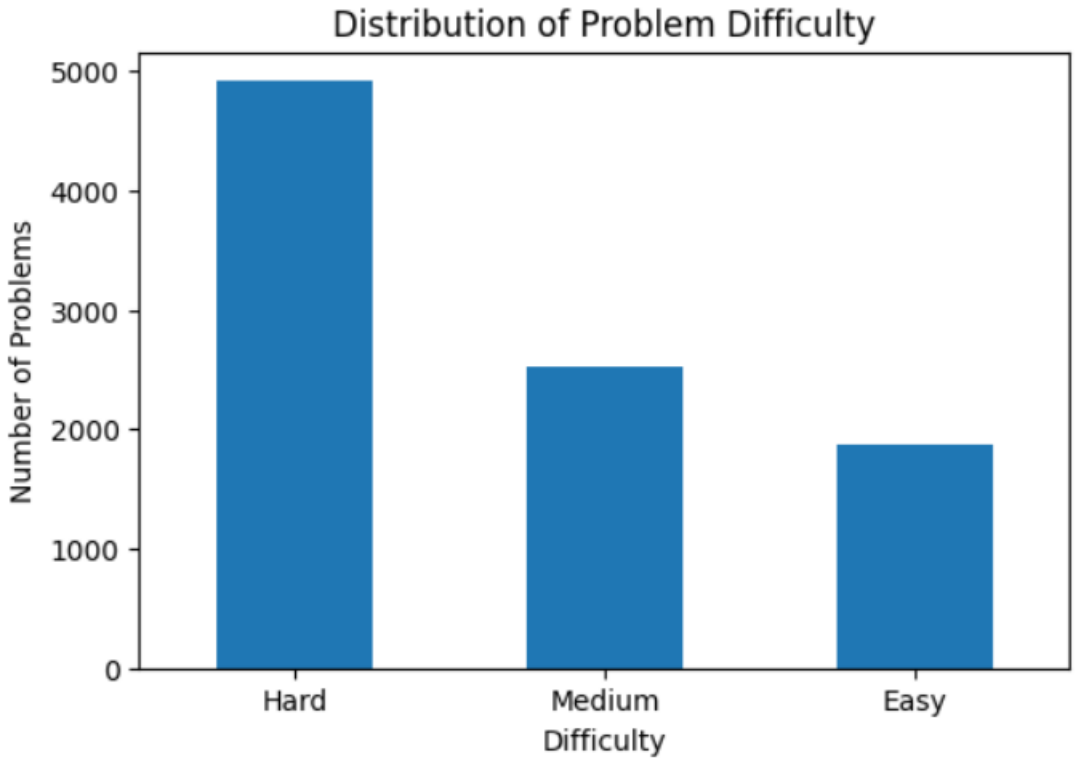
The original dataset contains approximately 27 feature columns, including problem identifiers, contest-related information, execution constraints, editorial content, tags, and other auxiliary metadata. The dataset is provided with predefined training and testing splits, consisting of 9,556 problems in the training set and 468 problems in the test set, both sharing the same feature structure.

Each data sample represents a single programming problem and includes detailed textual information describing the problem requirements. A numerical difficulty rating is also provided for each problem, which serves as the target variable for difficulty estimation.

For this project, only the following fields were retained: problem title, problem description, input format, output format, and numerical difficulty rating. All other metadata and user-related information were intentionally excluded to maintain a strict text-only prediction setting.

In addition to the numerical difficulty rating, a categorical difficulty label was derived for each problem to support the classification task. The details of this label construction are described in the data preprocessing section.

Figure 1: Distribution of problems across Easy, Medium, and Hard difficulty classes in the dataset.



Section 4: Data Preprocessing

The raw dataset obtained from Hugging Face contains a large number of metadata and columns that are not directly relevant to text-based difficulty prediction. As a first step, the dataset was filtered to retain only the fields required for this project: problem title, problem description, input format, output format, and numerical difficulty rating. All other columns were removed to enforce a strict text-only setting and to reduce noise in the learning process.

Rows with missing numerical difficulty ratings were removed from the dataset, as such samples cannot be used for supervised learning. For the remaining data, missing values in textual fields were handled by replacing them with empty strings. This ensured that all problem statements could be processed uniformly without introducing invalid or undefined inputs during feature extraction.

In order to support the classification task, an additional categorical column named difficulty was created from the numerical difficulty rating. The following thresholds were used to assign difficulty classes:

- Easy: rating less than 1200
- Medium: rating between 1200 and 1799
- Hard: rating 1800 and above

These thresholds align with commonly accepted difficulty ranges used in competitive programming platforms.

After preprocessing, the final dataset consisted of clean textual fields and two target variables: a numerical difficulty rating for regression and a categorical difficulty label for classification. This processed dataset served as the input for subsequent feature extraction and model training stages.

Section 5: Feature Engineering

After data preprocessing, the next step involved transforming the textual problem statements into numerical representations suitable for machine learning models. Since the objective of the project is text-based difficulty prediction, feature engineering focused exclusively on natural language processing techniques.

For each problem, the title, description, input format, and output format were concatenated into a single textual representation. This combined text provides a unified view of the problem statement and ensures that all relevant textual information contributes to feature generation. The text combination process is deterministic and was applied consistently across training, testing, and inference stages to maintain feature alignment.

Prior to vectorization, basic text normalization was performed. All text was converted to lowercase, newline characters were removed, non-alphanumeric characters were replaced with whitespace, and extra spaces were normalized. These steps reduce noise in the text while preserving meaningful tokens related to algorithmic concepts and constraints.

To convert the cleaned text into numerical features, Term Frequency–Inverse Document Frequency (TF-IDF) vectorization was used. The vectorizer was configured to extract both unigrams and bigrams, allowing the model to capture individual keywords as well as short phrases commonly associated with problem difficulty. English stopwords were removed, and the vocabulary size was limited to the most informative features to control dimensionality.

Importantly, the TF-IDF vectorizer was fitted only on the training dataset. The trained vectorizer was then reused without modification to transform the test data and to generate features during inference in the web application. This approach prevents information leakage from the test set and ensures that evaluation results accurately reflect model generalization performance.

The trained TF-IDF vectorizer was saved and reused across both the classification and regression pipelines, ensuring consistent feature representations for all downstream tasks.

Section 6: Methodology and Experimental Setup

The AutoJudge system addresses two related prediction tasks: categorical difficulty classification and numerical difficulty score prediction. Both tasks operate on the same TF-IDF feature representation derived from problem statements, ensuring consistency across the modeling pipeline.

For the classification task, multiple classical machine learning models were evaluated to predict the difficulty class of a problem as Easy, Medium, or Hard. The models considered include Logistic Regression, Linear Support Vector Machine (SVM), and Random Forest Classifier. Logistic Regression and Linear SVM serve as strong linear baselines commonly used in text classification tasks, while Random Forest was included to capture non-linear relationships between textual features.

For the regression task, the goal is to predict a numerical difficulty score corresponding to the problem's estimated complexity. The models evaluated for this task include Linear Regression, Gradient Boosting Regressor, and Random Forest Regressor. Linear Regression provides a baseline for numerical prediction, while Gradient Boosting and Random Forest models are capable of modeling complex non-linear patterns present in high-dimensional TF-IDF feature spaces.

All models were trained using the preprocessed training dataset and evaluated on a held-out test set provided by the dataset. The same TF-IDF feature representation was used for both classification and

regression tasks. Model evaluation was performed using appropriate metrics for each task, with classification models assessed using accuracy and confusion matrices, and regression models assessed using Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

Based on comparative evaluation across these metrics, Random Forest models were selected as the final models for both classification and regression due to their superior generalization performance and robustness when handling textual feature interactions.

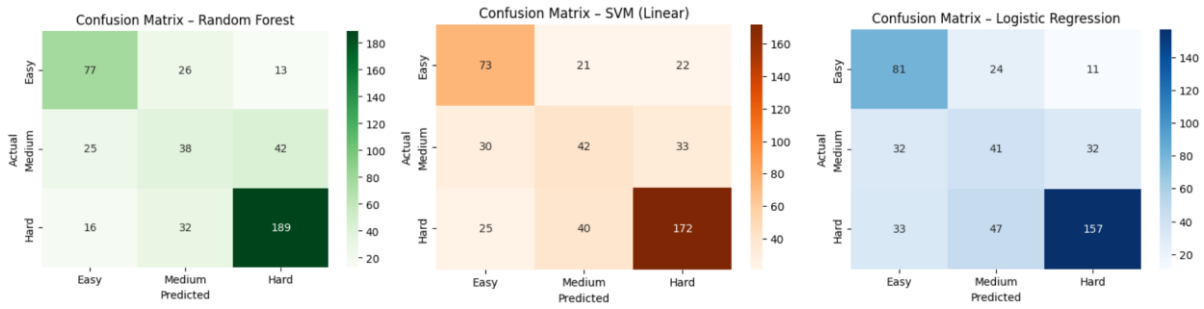
Section 7: Experimental Results and Evaluation

The performance of the proposed system was evaluated separately for the classification and regression tasks using the predefined test split of the dataset. All evaluation results reported in this section correspond to models trained on the training set and evaluated on unseen test data.

For the difficulty classification task, three models were evaluated: Logistic Regression, Linear Support Vector Machine, and Random Forest Classifier. Classification performance was measured using overall accuracy, along with confusion matrix analysis to better understand class-wise prediction behavior.

The Logistic Regression model achieved an accuracy of approximately 60.9 percent on the test set. The Linear SVM model showed a slight improvement, achieving an accuracy of around 62.7 percent. Among the evaluated models, the Random Forest Classifier performed best, achieving an accuracy of approximately 66.4 percent. Confusion matrix analysis indicated that the Random Forest model provided improved separation between Medium and Hard difficulty classes compared to the linear baselines, while maintaining reasonable performance on Easy problems.

Figure 2**:** Confusion matrices for difficulty classification on the test dataset using Random Forest, Linear SVM, and Logistic Regression models.



The confusion matrices illustrate class-wise prediction behavior for the evaluated classification models. Among the three models, the Random Forest classifier demonstrates improved separation between Medium and Hard difficulty classes, with fewer extreme misclassifications compared to the linear models. Logistic Regression and Linear SVM exhibit higher confusion between adjacent difficulty levels, particularly for Medium problems. These observations, along with overall accuracy results, motivated the selection of Random Forest as the final classification model

For the numerical difficulty score prediction task, regression models were evaluated using Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). The Linear Regression model produced relatively high error values, with an MAE of approximately 694 and an RMSE of around 869, indicating limited ability to capture non-linear relationships in the data. Gradient Boosting Regressor improved upon this baseline, achieving an MAE of approximately 525 and an RMSE of around 665.

The Random Forest Regressor achieved the lowest error among the evaluated models, with an MAE of approximately 508 and an RMSE of around 664. These results suggest that tree-based models are better suited for modeling the complex and non-linear associations between textual features and problem difficulty.

The comparative results for both classification and regression tasks are summarized using tables, and representative confusion matrices are included as figures to illustrate class-wise prediction behavior.

Section 8: Web Interface and Sample Predictions

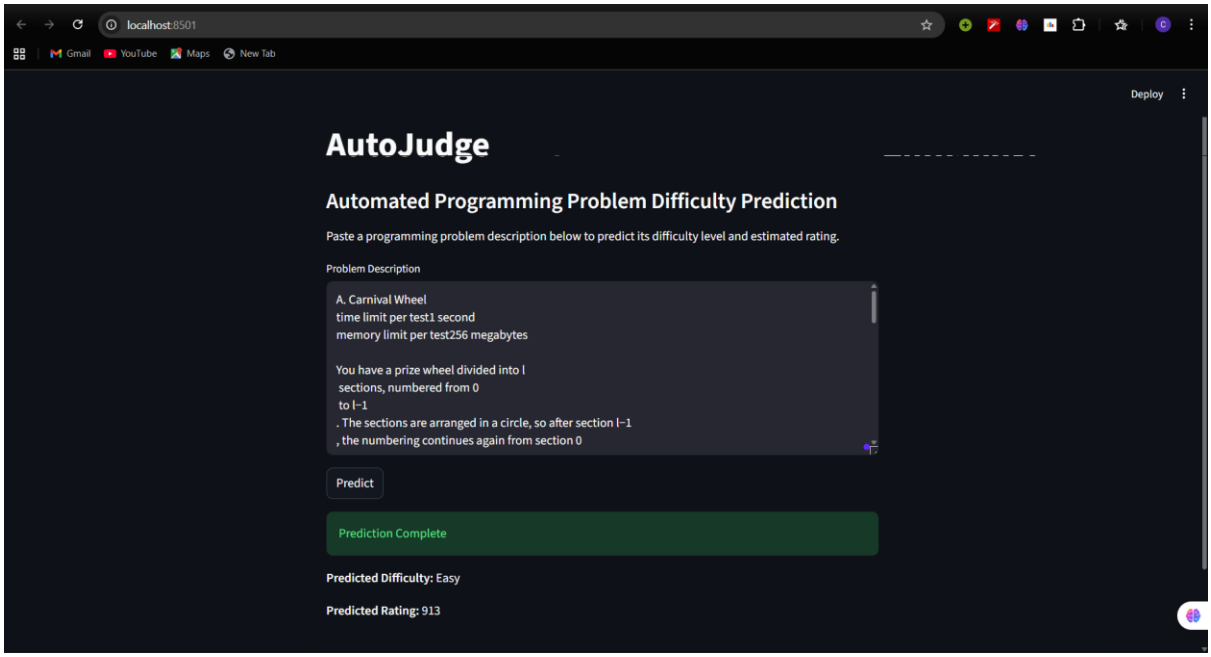
To demonstrate the practical usability of the proposed system, a web-based interface was developed using the Streamlit framework. The web application serves as an inference layer that allows users to interact with the trained models without requiring any knowledge of the underlying machine learning pipeline.

The application loads the pre-trained TF-IDF vectorizer, difficulty classification model, and difficulty score regression model from disk at runtime. No model training is performed within the web interface. Given a new problem statement, the same preprocessing and feature extraction steps used during training are applied before generating predictions.

The user interface accepts four textual inputs: problem title, problem description, input format, and output format. Upon submission, these inputs are combined and processed to generate a feature representation, which is then passed to the classification and regression models. The application outputs a predicted categorical difficulty label (Easy, Medium, or Hard) along with a predicted numerical difficulty score.

Sample predictions were generated using real competitive programming problem statements. Screenshots of the web interface and example prediction outputs are included to illustrate the end-to-end functionality of the system. These examples confirm that the application correctly integrates the trained models and produces consistent difficulty estimates for unseen problems.

Figure 3: Streamlit-based web interface demonstrating difficulty class and numerical difficulty score prediction for an unseen programming problem.



Section 9: Conclusion and Future Work

This project presented AutoJudge, a machine learning–based system for predicting the difficulty of competitive programming problems using only their textual problem statements. By operating under a strict text-only constraint and relying on classical machine learning techniques, the system demonstrates that meaningful difficulty estimation is feasible without the use of user performance data or platform-specific metadata.

The complete pipeline, including data preprocessing, feature engineering, model training, evaluation, and deployment, was implemented in a modular and reproducible manner. Experimental results showed that Random Forest models provided the best performance for both difficulty classification and numerical difficulty score prediction, outperforming linear baselines on the evaluated dataset. The integration of these models into a Streamlit-based web application further demonstrated the practical applicability of the approach.

Despite these encouraging results, the system has certain limitations. Difficulty estimation based solely on textual information cannot fully capture all aspects of problem complexity, particularly those related to implementation effort or edge-case handling. Additionally, the performance of the models is influenced by the quality and distribution of difficulty ratings available in the dataset.

Future work could explore incorporating richer textual representations, such as problem examples or constraint analysis, while still adhering to classical learning paradigms. Further improvements may also include dataset expansion, refinement of difficulty thresholds, or exploration of additional interpretable models to enhance prediction reliability.