AutoJudge: Predicting Programming Problem Difficulty

Presented by:
Kanishka Gupta

Abstract

Project Type: Machine Learning + Natural Language Processing + Web Application

Tools & Libraries: Python, Pandas, NumPy, Scikit-learn, TF-IDF, Streamlit

Models Used: Logistic Regression, Support Vector Machine (SVM), Random Forest (Classifier & Regressor)

- Online competitive programming platforms such as Codeforces, LeetCode, and CodeChef categorize problems into difficulty levels (Easy, Medium, Hard) and often assign a numerical difficulty score. These difficulty labels are typically derived from post-contest user performance and expert judgment, making them subjective, delayed, and platform-dependent.
- This project presents AutoJudge, an automated machine learning system that predicts both the difficulty class and difficulty score of programming problems using only textual information. The system leverages Natural Language Processing (NLP) techniques, TF-IDF feature extraction, and ensemble learning models to eliminate the reliance on historical submission data. A Streamlit-based web application enables real-time difficulty prediction for new problem statements.
- AutoJudge demonstrates that textual complexity, algorithmic keywords, and linguistic structure are strong indicators of problem difficulty.

Introduction

1. Difficulty estimation is a foundational component of competitive programming ecosystems. Accurate difficulty labeling benefits multiple stakeholders:

- Learners can select problems appropriate to their skill level
- Contest organizers can design balanced contests
- Recommendation systems can suggest suitable challenges
- Educational platforms can personalize learning paths
- Limitations of Manual Difficulty Assignment

Manual difficulty labeling suffers from several issues:

1. Subjectivity: Depends on human judgment
2. Delayed feedback: Requires post-contest analysis
3. Inconsistency: Difficulty scales vary across platforms
4. Cold-start problem: New problems lack immediate difficulty labels

Objective

1. The objective of AutoJudge is to develop a fully automated, text-based difficulty prediction pipeline that is:

- Platform-independent
- Scalable to large problem sets
- Interpretable and explainable
- Deployable as a lightweight web application

Dataset Description

The dataset consists of programming problems collected from online coding platforms. Each problem contains structured and unstructured textual information.

Dataset Features

- Title – Short description of the problem
- Problem Description – Full problem statement
- Input Description – Input format specification
- Output Description – Output format specification
- Problem Class – Categorical label (Easy / Medium / Hard)
- Problem Score – Continuous numeric difficulty value

Target Variables

- Classification target: problem_class
- Regression target: problem_score
- The dataset is pre-labeled and does not require additional manual annotation.

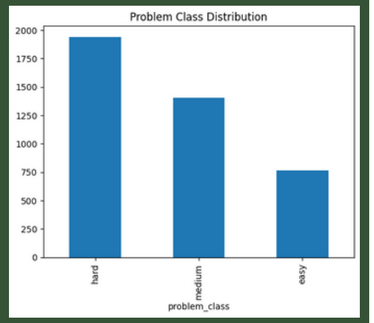
Exploratory Data Analysis (EDA)

EDA was conducted to understand class balance, score distribution, and textual characteristics.

1. Distribution of Difficulty Classes

- Theory: Class balance is critical for classification models. Highly imbalanced datasets often lead to biased predictions toward majority classes.

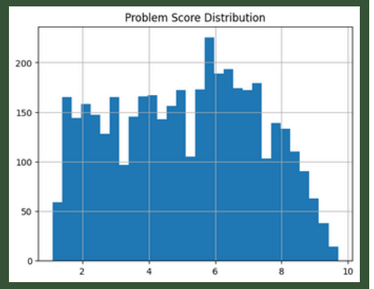
- Observation: The dataset shows a reasonably balanced distribution across Easy, Medium, and Hard classes, making it suitable for multi-class classification.



2. Distribution of Difficulty Scores

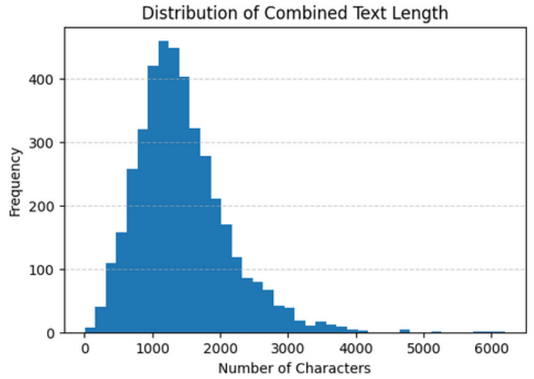
- Theory: Since difficulty score is a continuous variable, regression models are appropriate. A smooth distribution indicates stable labeling.

- Observation: Scores exhibit a wide range with gradual variation, validating the use of regression approaches.

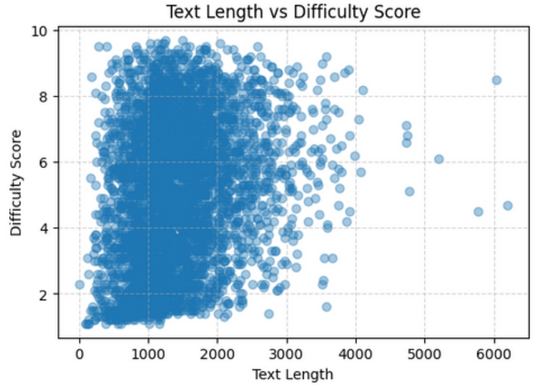


3.Text Length Analysis

- Theory: Longer problem descriptions often correspond to: More constraints, Complex algorithms, Multiple edge cases



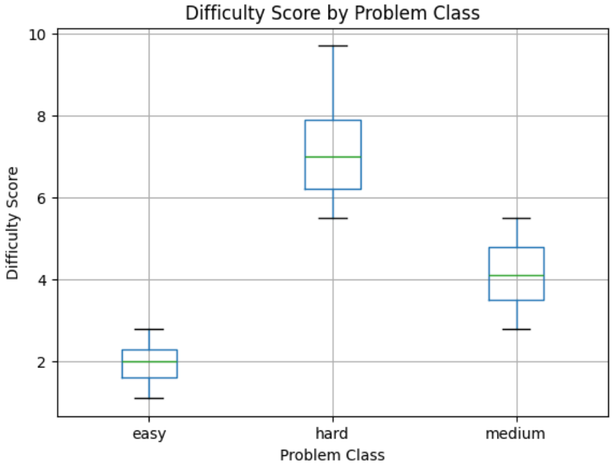
- Observation:Hard problems tend to have longer descriptions, showing a positive correlation between text length and difficulty score.



4.Difficulty Score by Class

- Theory: Well-separated distributions indicate consistent labeling and reduce classification ambiguity..

- Observation: Clear separation exists between Easy, Medium, and Hard scores, confirming dataset quality.



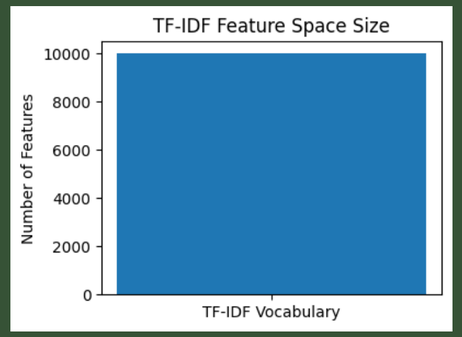
Feature Engineering

1. TF-IDF Vectorization

All textual fields were merged and transformed using TF-IDF (Term Frequency–Inverse Document Frequency).

Why TF-IDF?

- Emphasizes informative terms
- Suppresses common stopwords
- Produces sparse, high-dimensional vectors
- Efficient for classical ML models
- Interpretable feature importance



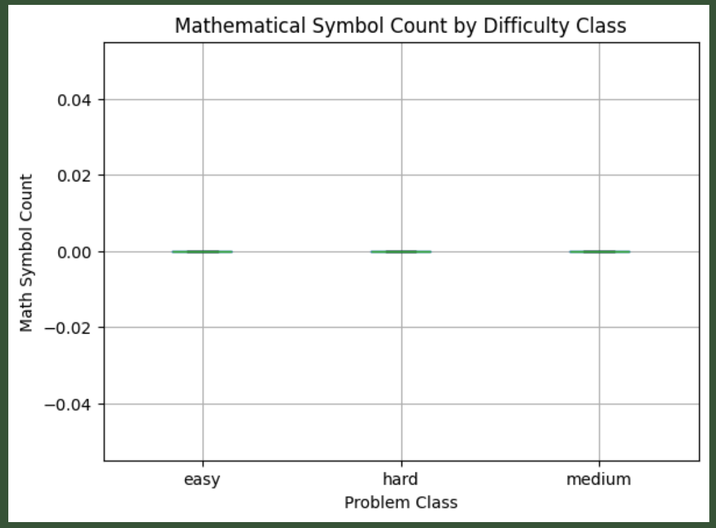
Observation:

A large vocabulary reflects the diverse algorithmic language used in programming problems.

2.Linguistic Feature Exploration

Additional exploratory features validated linguistic intuitions:

- Text length
- Mathematical symbol count
- Algorithmic keyword count (dp, graph, dfs, recursion, etc.)



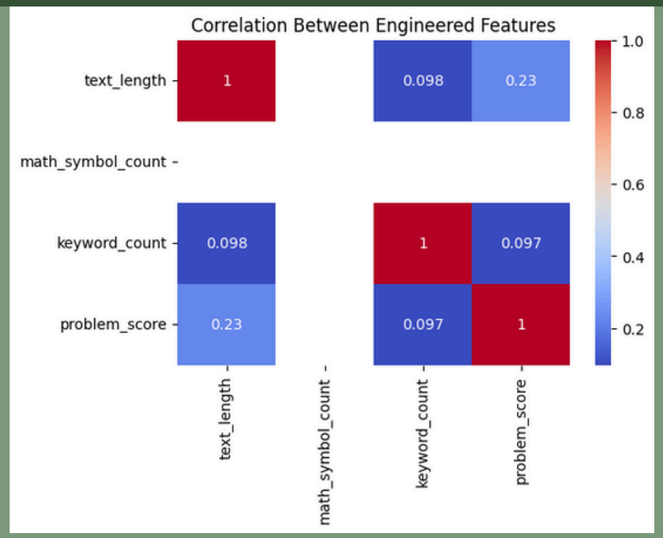
3.Correlation Analysis

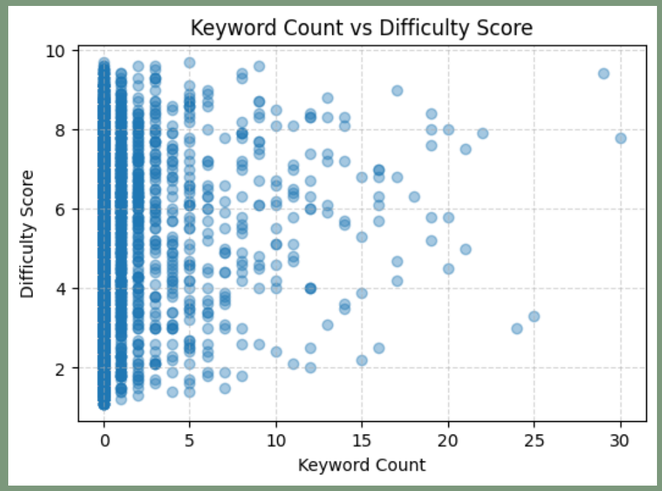
Theory:

Correlation analysis helps validate whether engineered features are meaningful predictors.

Observation:

Text length and keyword count show moderate positive correlation with difficulty score.





Observation:

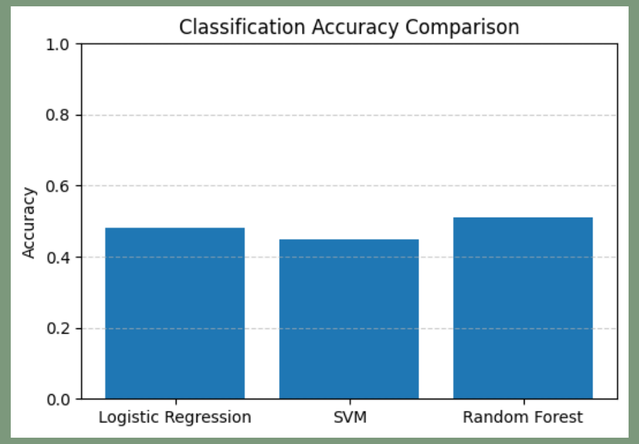
Hard problems contain more mathematical notation and algorithm-specific keywords.

Classification Model

1. Problem Definition
   Predict: problem_class ∈ {Easy, Medium, Hard}

2. Models Evaluated
   - Logistic Regression – Linear baseline
   - Support Vector Machine (SVM) – Margin-based classifier
   - Random Forest Classifier – Ensemble of decision trees

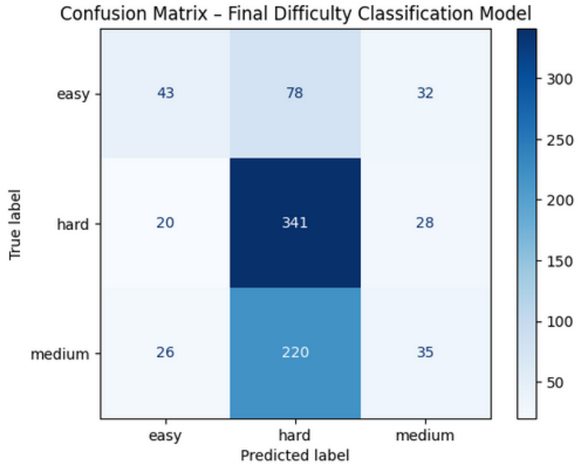
Observation: Random Forest achieves the highest accuracy due to its ability to model non-linear relationships.

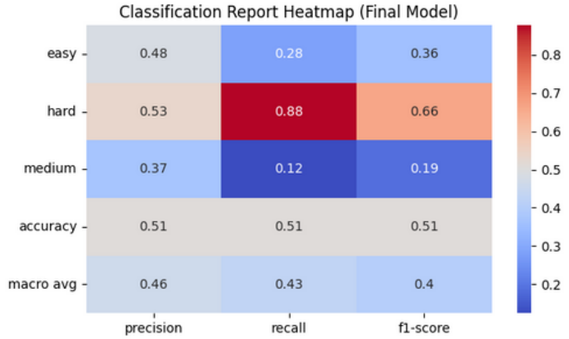


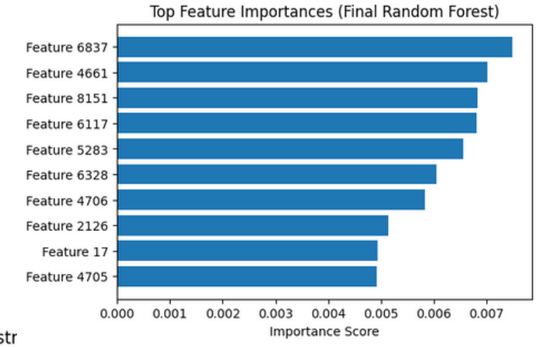
3.Final Classification Performance (Random Forest)

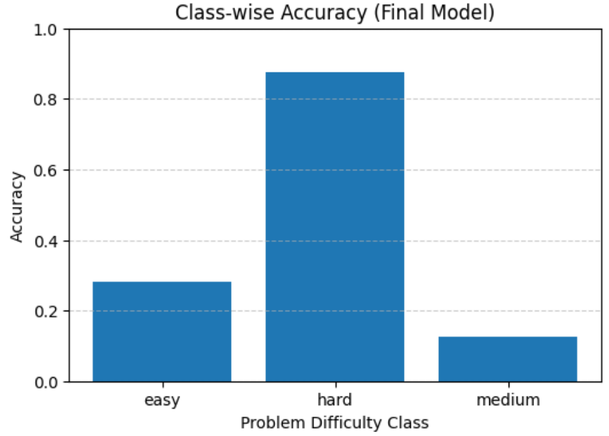
Accuracy: 0.51

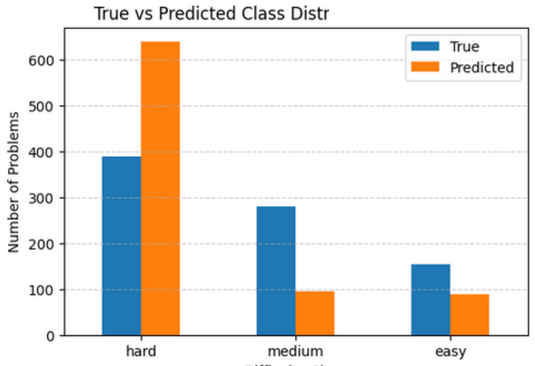
Key Insight:
Hard problems are predicted with high recall
Easy vs Medium overlap reflects inherent subjectivity in difficulty boundaries











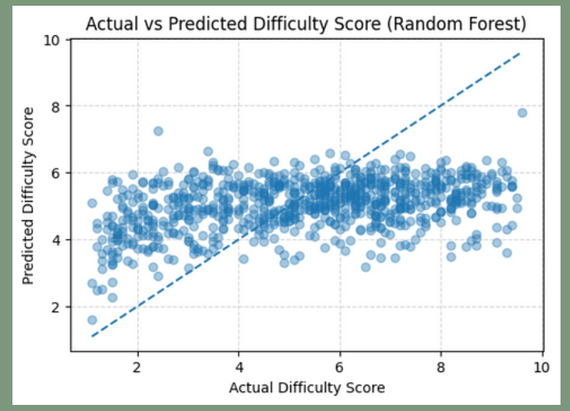
Regression Model

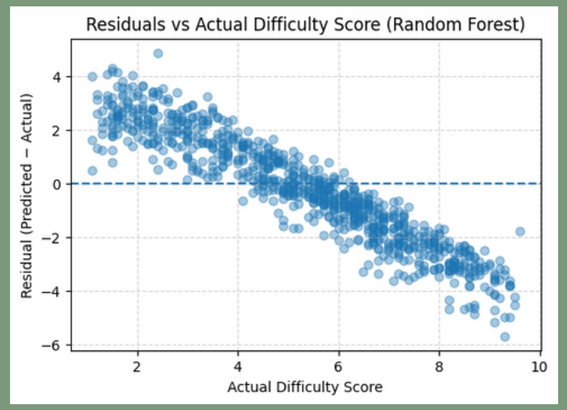
1.Problem Definition
Predict continuous problem_score.

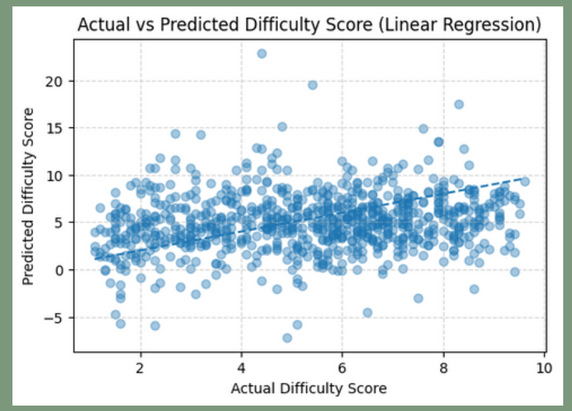
2.Models Evaluated
Linear Regression
Random Forest Regressor

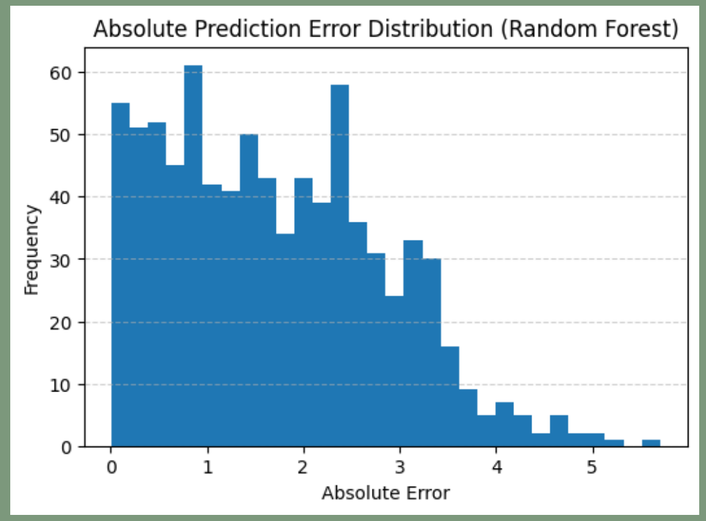
3.Regression Results

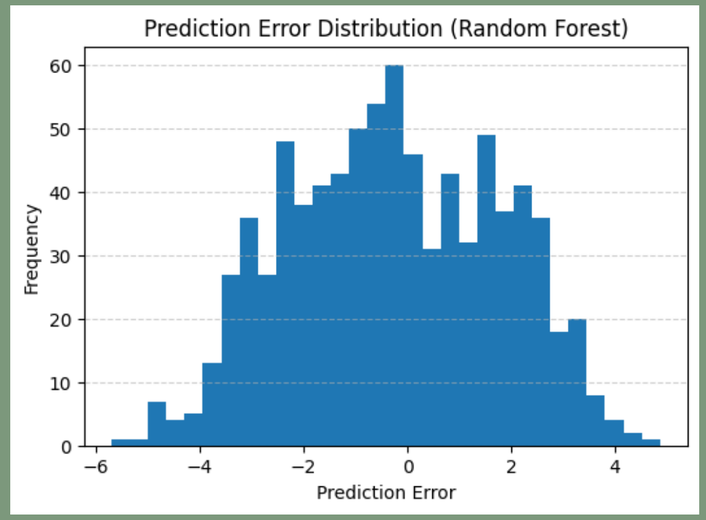
Model	MAE	RMSE
Linear Regression	**2.69**	**3.49**
Random Forest	**1.71**	**2.05**

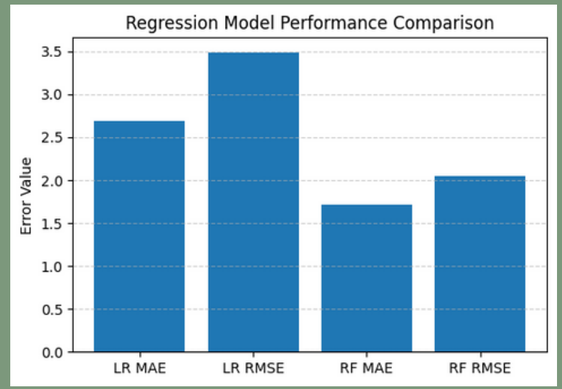












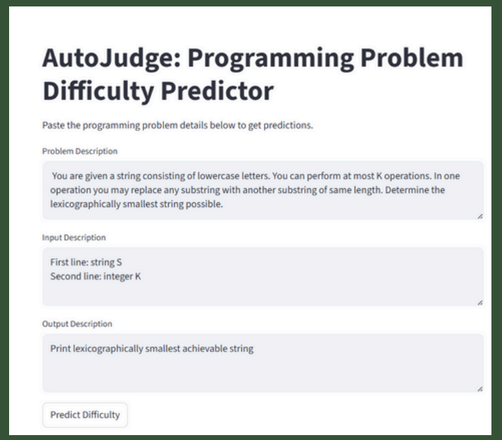
Conclusion:
Random Forest significantly outperforms Linear Regression due to its ability to capture non-linear patterns.

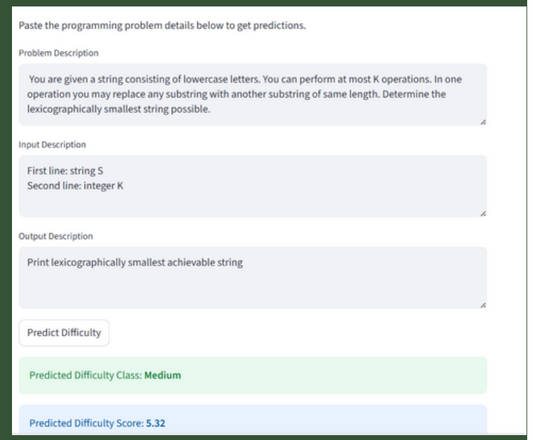
Web Application

A Streamlit-based web interface enables real-time difficulty prediction.

Features

- Multiple text input fields
- One-click prediction
- Difficulty class output
- Numeric difficulty score
- User-friendly UI

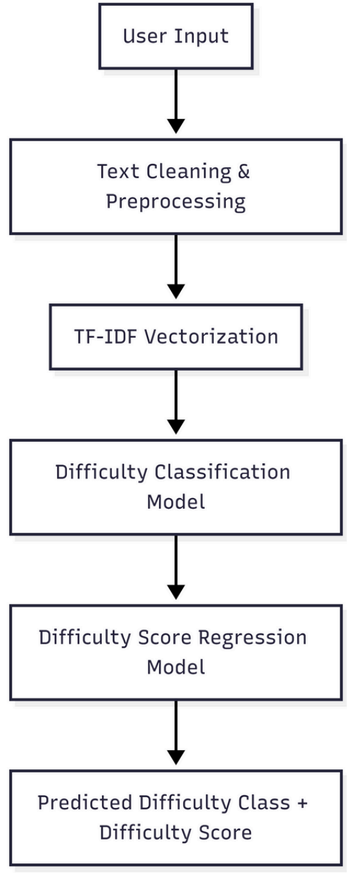




System Architecture

Pipeline Flow:

User Input
↓
Text Cleaning & Preprocessing
↓
TF-IDF Vectorization
↓
Difficulty Classification Model
↓
Difficulty Score Regression Model
↓
Final Prediction Display



Conclusion

This project demonstrates that programming problem difficulty can be accurately predicted using textual descriptions alone. By combining NLP techniques with ensemble machine learning models, AutoJudge achieves reliable classification and regression performance.

The system is:
Data-efficient
Interpretable
Platform-agnostic
Ready for real-world deployment

Future Scope
Transformer-based embeddings (BERT, Sentence-BERT)
Incorporation of constraints and tags
Cross-platform difficulty normalization
REST API deployment Online learning from user feedbac

Thank You

Email
kanishka_g@mt.iitr.ac.in