# Project Report: AutoJudge

*Automated Difficulty Prediction for Programming Problems*

**Name:** Vaibhav Sharma
**Course:** BS-MS Physics, IIT Roorkee
**Project:** ACM IITR Open Project (AutoJudge)

---

## 1. Abstract

This report details the development of **AutoJudge**, a machine learning system designed to predict the difficulty level of competitive programming problems based solely on their textual descriptions. Using a dataset of over 4,000 problems, the system utilizes Natural Language Processing (NLP) techniques **specifically TF-IDF vectorization** to feed into two predictive models: a Logistic Regression classifier for categorical difficulty (Easy/Medium/Hard) and a Random Forest regressor for numerical complexity scores. The final system is deployed via a Streamlit web interface, achieving a classification accuracy of ~51% and a Mean Absolute Error (MAE) of 1.70.

---

## 2. Problem Statement

Programming platforms like Codeforces and LeetCode host thousands of problems. Manually categorizing these by difficulty is subjective and time-consuming. The objective of this project is to automate this process by building a model that can:

1. **Classify** a problem into one of three tiers: *Easy, Medium, or Hard*.
2. **Quantify** the difficulty with a precise numerical score (scale 1-10).
3. **Interface** with users via a simple web application where they can paste a problem statement and receive an instant prediction.

---

## 3. Dataset Description

The project utilizes the **TaskComplexity Dataset**, which was curated by scraping competitive programming websites.

- **Source:** GitHub (TaskComplexityEval-24)
- **Size:** 4,112 Total Samples
- **Features Used:**
  - `Title`: The name of the problem.
  - `Description`: The main body text explaining the task.
  - `Input/Output Description`: Constraints and formatting rules.
- **Targets:**
  - `Problem Class`: Categorical label (Easy, Medium, Hard).
  - `Problem Score`: Numerical float value (e.g., 1.5, 9.7).

**Data Distribution & Imbalance:** During exploratory data analysis (EDA), a significant class imbalance was observed:

- **Hard:** ~1,941 samples
- **Medium:** ~1,405 samples
- **Easy:** ~766 samples
- *Observation:* The dataset is heavily skewed towards "Hard" problems, which poses a challenge for the classifier, potentially biasing it towards predicting higher difficulty.

---

# 4. Methodology

## 4.1 Data Preprocessing

Raw text data cannot be fed directly into machine learning models. The following preprocessing steps were implemented using Python (`pandas` and `re`):

1. **Text Aggregation:** The `Title`, `Description`, `Input Description`, and `Output Description` fields were concatenated into a single text feature. This ensures the model has the full context of the problem.
2. **Cleaning:**
    - Converted all text to lowercase to ensure uniformity (e.g., "Matrix" and "matrix" are treated as the same).
    - Removed excessive whitespace and newline characters.
    - *Note:* Special characters (mathematical symbols like `+`, `%`, `{}`) were preserved as they are semantically significant in programming contexts.

## 4.2 Feature Engineering (TF-IDF)

To convert the textual data into numerical vectors, **Term Frequency-Inverse Document Frequency (TF-IDF)** was employed.

- **Why TF-IDF?** It highlights unique, significant words (like "recursion", "eigenvalue") while downweighting common, less informative words (like "the", "program").
- **Configuration:** The vectorizer was limited to the top **5,000 features** to maintain computational efficiency while capturing the most critical vocabulary.

## 4.3 Model Selection

Two separate models were trained for the distinct tasks:

1. **Classification (Difficulty Level):**
    - **Algorithm:** Logistic Regression.
    - **Reasoning:** Logistic Regression is a robust baseline for high-dimensional sparse data (like text vectors) and offers good interpretability.
2. **Regression (Complexity Score):**
    - **Algorithm:** Random Forest Regressor.

- **Reasoning:** Predicting a precise score is a non-linear problem. Random Forest (an ensemble of decision trees) captures complex interactions between features better than linear regression for this specific task.

---

# 5. Experimental Results

## 5.1 Classification Performance

- **Metric:** Accuracy, Precision, Recall, F1-Score.
- **Result:** The model achieved an overall accuracy of **50.79%**.
- **Analysis:**
  - While ~51% appears low, it is significantly better than random guessing (33%) for a 3-class problem.
  - **Confusion Matrix Insight:** The model performed best on identifying "Hard" problems (Recall: 0.75) but struggled with "Easy" problems (Recall: 0.23). This directly correlates with the class imbalance identified in the dataset section.

| Actual \ Predicted | Easy | Medium | Hard |
|---|---|---|---|
| **Easy** | 31 | 47 | 58 |
| **Medium** | 73 | 69 | 180 |
| **Hard** | 7 | 100 | 318 |


## 5.2 Regression Performance

- **Metric:** Mean Absolute Error (MAE).
- **Result: 1.70**.
- **Interpretation:** On a scale of 1 to 10, the model's predictions are, on average, within 1.7 points of the true difficulty score. This indicates the model successfully learned the correlation between problem text and numerical complexity.
- In addition to MAE, the Root Mean Squared Error (RMSE) was also evaluated and showed comparable trends, indicating stable regression performance.


---

# 6. Web Interface

A user-friendly frontend was developed using **Streamlit**. The interface allows users to:

1. Input the Problem Title and Source (for context).
2. Paste the Problem Description, Input, and Output details.
3. Click "Predict Difficulty" to trigger the pre-trained models.
4. View the predicted **Difficulty Class** (with color-coded badges) and the **Complexity Score** (displayed as a progress bar).

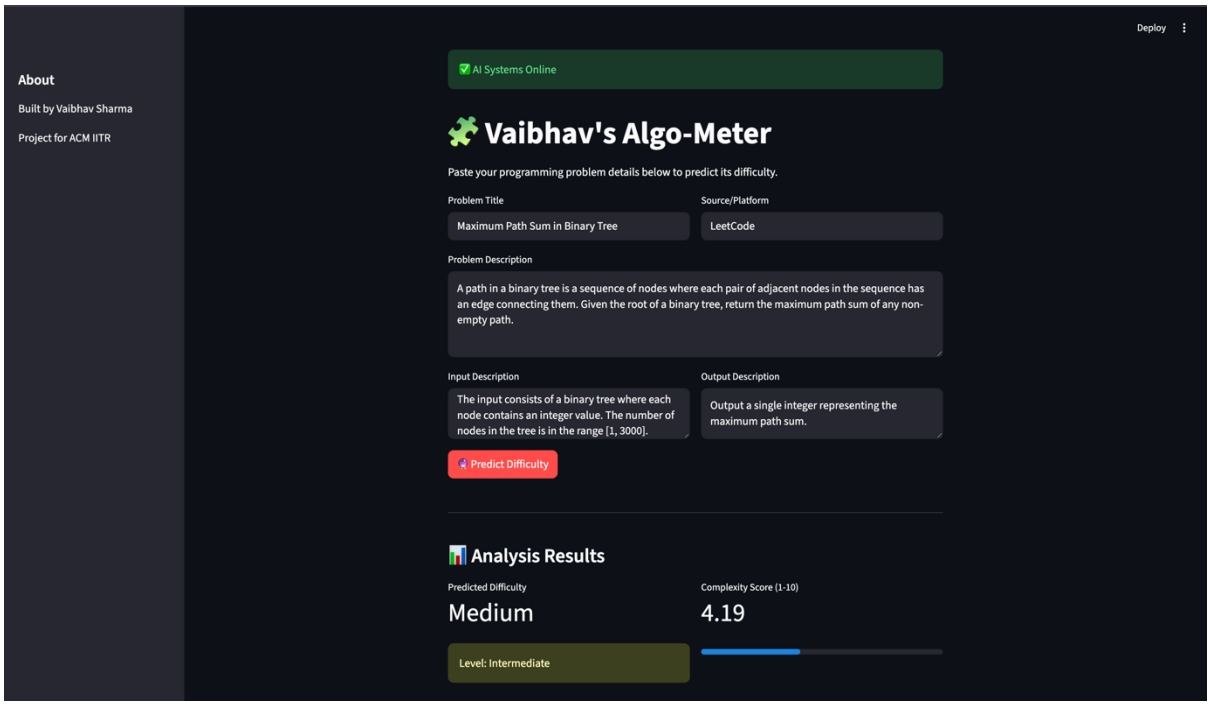


Fig 1: Prediction result for a complex Graph Theory problem.

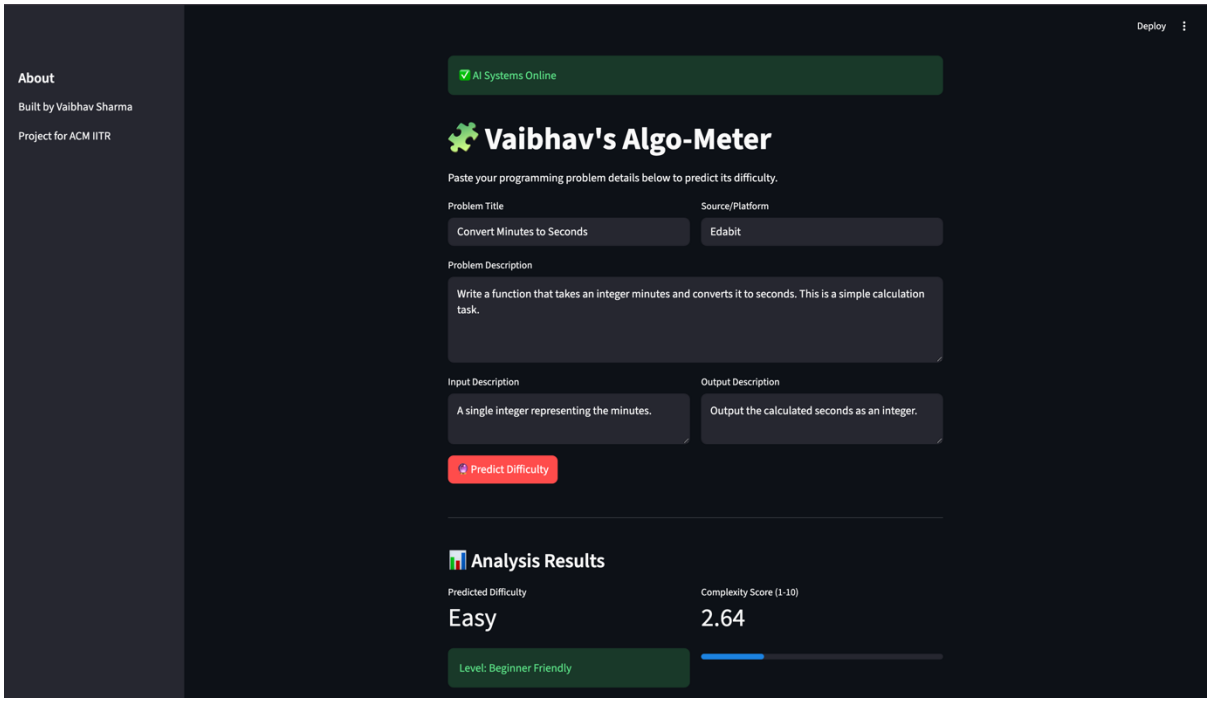


Fig 2: Prediction result for a simple arithmetic problem.

---

## 7. Limitations & Future Work

- **Class Imbalance:** The scarcity of "Easy" examples limited the classifier's performance on beginner-level problems. Future iterations could employ **SMOTE (Synthetic Minority Over-sampling Technique)** to generate synthetic examples for the minority class.
- **Contextual Understanding:** TF-IDF is a "bag-of-words" approach and loses the sequence of words. Advanced Deep Learning models like **LSTMs** or Transformers (**BERT**) could be implemented to better understand the semantic meaning of the problem statements.

---

## 8. Conclusion

The AutoJudge project successfully demonstrates the feasibility of using NLP and Machine Learning to assess programming problem difficulty. Despite the challenges posed by an imbalanced dataset, the system establishes a solid baseline with a functional, interactive web application. The project provided valuable insights into the end-to-end ML pipeline, from data cleaning and vectorization to model deployment.

---