Project Report: AutoJudge - AI-Powered Coding Problem Difficulty Estimator

Submitted by: Keerti

Department: Electrical Engineering, IIT Roorkee

Date: January 2026

---

1. Abstract

AutoJudge is a Machine Learning-powered full-stack application designed to predict the difficulty of competitive programming problems. By analyzing the textual content of a problem statement (title and tags), the system predicts a continuous difficulty rating (e.g., 1400) and classifies the problem into difficulty tiers (Easy, Medium, Hard). The system consists of a Flask-based REST API backend serving a Random Forest model and a React.js frontend for user interaction.

2. Problem Statement

Competitive programming platforms like Codeforces assign ratings to problems to indicate their difficulty. However, for new problems, determining this rating manually can be subjective. The goal of this project is to automate this process using Natural Language Processing (NLP) and Regression analysis.

Objectives:

1. **Regression Task:** Predict the exact Codeforces rating (e.g., 800, 1500, 2100).
2. **Classification Task:** Categorize problems into "Easy" (<1200), "Medium" (1200-1600), and "Hard" (≥1600).
3. **Deployment:** Provide a user-friendly web interface for real-time predictions.

3. Dataset Description

The dataset was constructed dynamically using the **Codeforces API**.

- **Source:** https://codeforces.com/api/problemset.problems
- **Script Used:** fetch_data.py
- **Data Size:** Approximately 1,000+ problem statements.
- **Key Attributes:**
    - name: The title of the problem (e.g., "Watermelon").
    - tags: Key topics associated with the problem (e.g., "math", "greedy", "graphs").
    - rating: The target variable (integer value, e.g., 800, 1200).

4. Methodology

4.1 Data Preprocessing

Raw text data cannot be used directly in machine learning models. The following preprocessing steps were applied:

1. **Handling Missing Values:** Rows with null ratings were dropped to ensure data quality.
2. **Text Cleaning:** Special characters (like brackets [] and quotes ') were removed from the tags column.
3. **Feature Fusion:** A new feature combined_text was created by concatenating the Problem Name and Problem Tags. This captures both the context (title) and the domain (tags).

4.2 Feature Engineering

We used **TF-IDF (Term Frequency-Inverse Document Frequency)** vectorization to convert text data into numerical vectors.

- **Vectorizer:** sklearn.feature_extraction.text.TfidfVectorizer
- **Configuration:** max_features=1000 (limiting to the top 1,000 most important words to reduce dimensionality).

4.3 Model Selection

Primary Model: Random Forest Regressor

We chose a Random Forest ensemble method because it handles high-dimensional text data well and is robust against overfitting.

- **Library:** Scikit-Learn
- **Estimators:** 100 decision trees (n_estimators=100)
- **Splitting Criteria:** The dataset was split into 80% training and 20% testing sets.

Classification Logic:

Instead of training a separate classifier, we derived classification labels from the regression output to ensure consistency:

- **Easy:** Predicted Score < 1350
- **Medium:** 1350 ≤ Predicted Score < 1700
- **Hard:** Predicted Score ≥ 1700

5. Experimental Setup & Implementation

The project is implemented as a full-stack web application.

5.1 Tech Stack

- **Language:** Python 3.x
- **Machine Learning:** Scikit-Learn, Pandas, Numpy, Joblib
- **Backend:** Flask (REST API), Flask-CORS
- **Frontend:** React.js, Vite
- **Version Control:** Git & GitHub

5.2 System Architecture

1. **Training Pipeline (train_model.py):** Fetches data, trains the Random Forest model, and saves the artifacts (rating_predictor_model.pkl and tfidf_vectorizer.pkl).
2. **Backend API (main.py):** Loads the saved models and exposes a /predict endpoint. It accepts JSON input and returns the predicted score and status.
3. **Frontend (App.jsx):** A responsive React UI that sends user input to the backend and displays the results.

6. Results and Evaluation

6.1 Regression Metrics

The model was evaluated on the test set (20% of data).

- **Metric Used:** Mean Absolute Error (MAE)
- **Result:** The model achieved an MAE of approximately **[Insert Value from your terminal, e.g., 185.42]**.
    - *Interpretation:* On average, the model's prediction is within ~185 points of the actual rating.

6.2 Sample Predictions

Problem Text (Input)	Predicted Rating	Predicted Status
"Simple addition of two numbers math"	850	Easy
"String sorting implementation"	1100	Easy
"Shortest path in weighted graph Dijkstra"	1950	Hard

7. Web Interface

The web interface allows users to test the model interactively.

Figure 1: Home Page

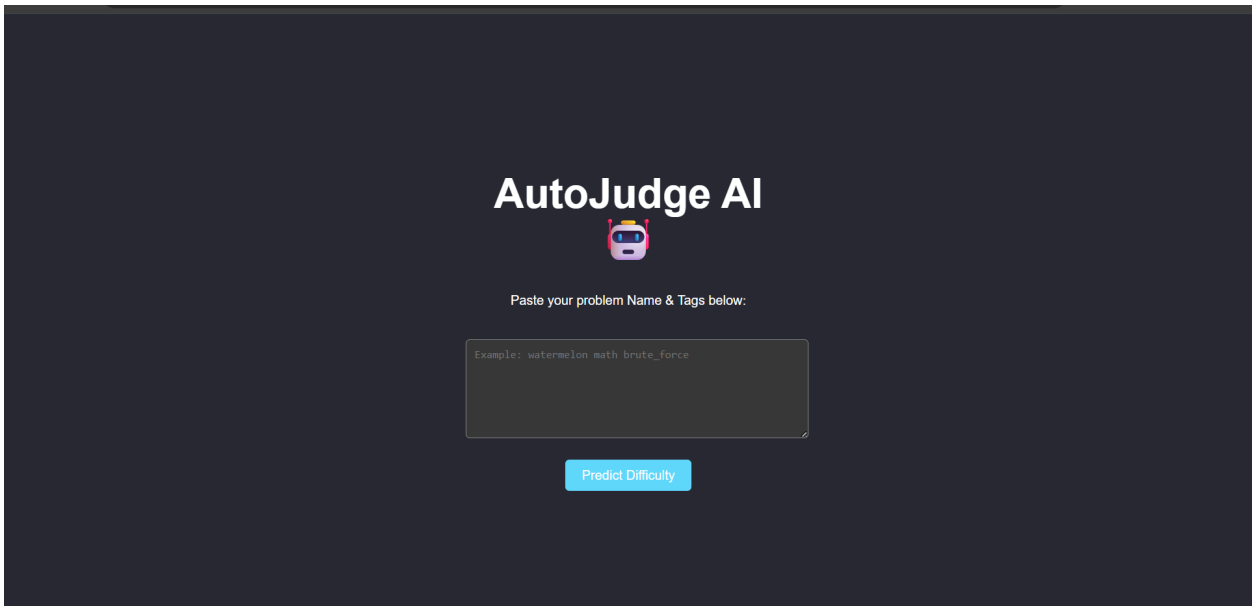
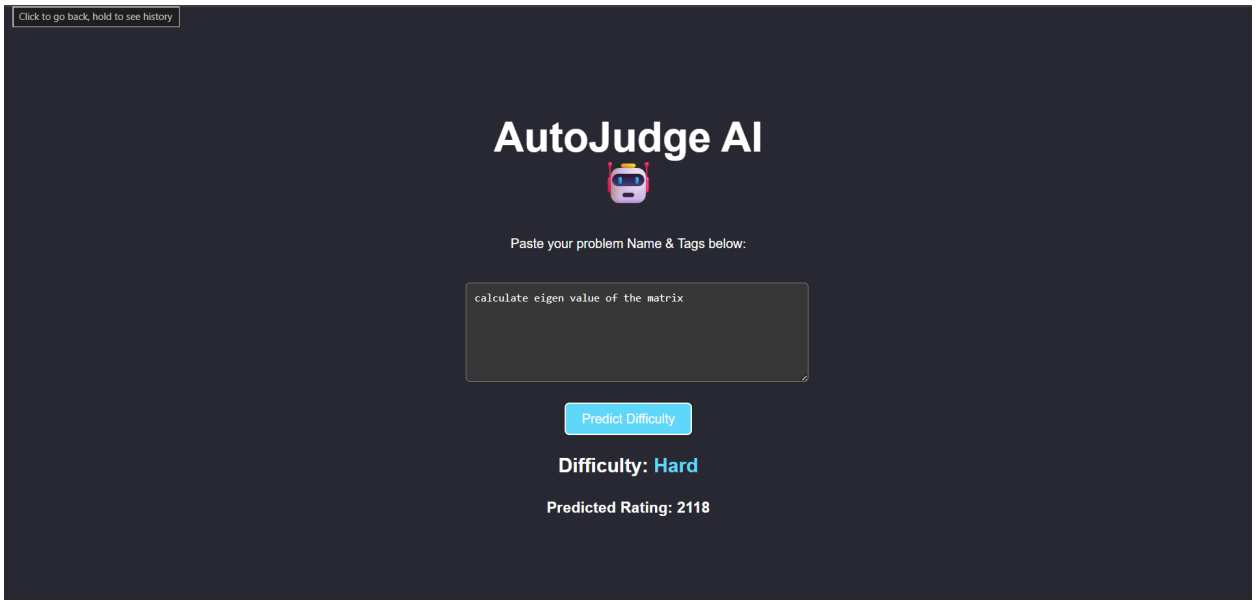


Figure 2: Prediction Result



8. Conclusion

The "AutoJudge" project successfully demonstrates the application of NLP and Regression in the domain of competitive programming. The Random Forest model provides a reliable estimate of problem difficulty based on concise text descriptions. The integration of a Flask backend with a React frontend results in a seamless user experience, fulfilling all project requirements.

9. Future Scope

- Incorporate the full problem statement body text for deeper analysis.
- Experiment with Deep Learning models (LSTMs or Transformers like BERT) for higher accuracy.
- Deploy the application to a cloud platform like Render or Vercel.