Project Report: AutoJudge - AI-Powered Problem Difficulty Predictor

Author: Chakrala Reddy Thanuj Royal
ID: 24114025
Branch: CSE (2nd Year)

1. Problem Statement

The difficulty of a programming problem is often subjective and can vary based on the topic, constraints, and problem statement complexity. Manual assignment of difficulty ratings is inconsistent and time-consuming. "AutoJudge" aims to automate this process by using Machine Learning to predict:
1. Difficulty Class: Easy, Medium, or Hard.
2. Difficulty Score: A granular numerical score from 0 to 10.

This system assists problem setters in calibrating contests and helps students identify problems suitable for their skill level.

2. Dataset Used

- Source: The dataset consists of programming problems collected from online judges (e.g., Codeforces).
- Size: Approximately 3000+ problems.
- Attributes:
    • name: Title of the problem.
    • description: The main textual description.
    • input_format: Constraints and input details.
    • output_format: Expected output format.
    • problem_class: Categorical difficulty (Easy, Medium, Hard).
    • problem_score: Numerical difficulty rating.

---

3. Data Preprocessing

Text data contains significant noise that hinders model performance. We implemented the following preprocessing pipeline:

1. Text Concatenation: Combined Title, Description, Input, and Output into a single text corpus to provide maximum context.

2. Noise Reduction:
- LaTeX Removal: Used Regex to strip mathematical formatting commands (\frac, \times, \le) which act as noise for difficulty prediction.
- Cleaning: Lowercased text and removed special non-alphanumeric characters.
- Whitespace Normalization: Removed multiple spaces and newlines.

4. Feature Engineering

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

- N-grams: We utilized trigrams (ngram_range=(1, 3)) to capture multi-word concepts (e.g., "shortest path", "dynamic programming").
- Max Features: Limited to the top 20,000 features to focus on the most relevant signals while maintaining computational efficiency.

5. Models Used

5.1 Classification Model
- Algorithm: LinearSVC (Support Vector Classifier).
- Reasoning: LinearSVC is highly effective for high-dimensional sparse data like text with TF-IDF features.
- Configuration: class_weight='balanced' was used to penalize misclassification of minority classes, ensuring fairness across Easy, Medium, and Hard categories.

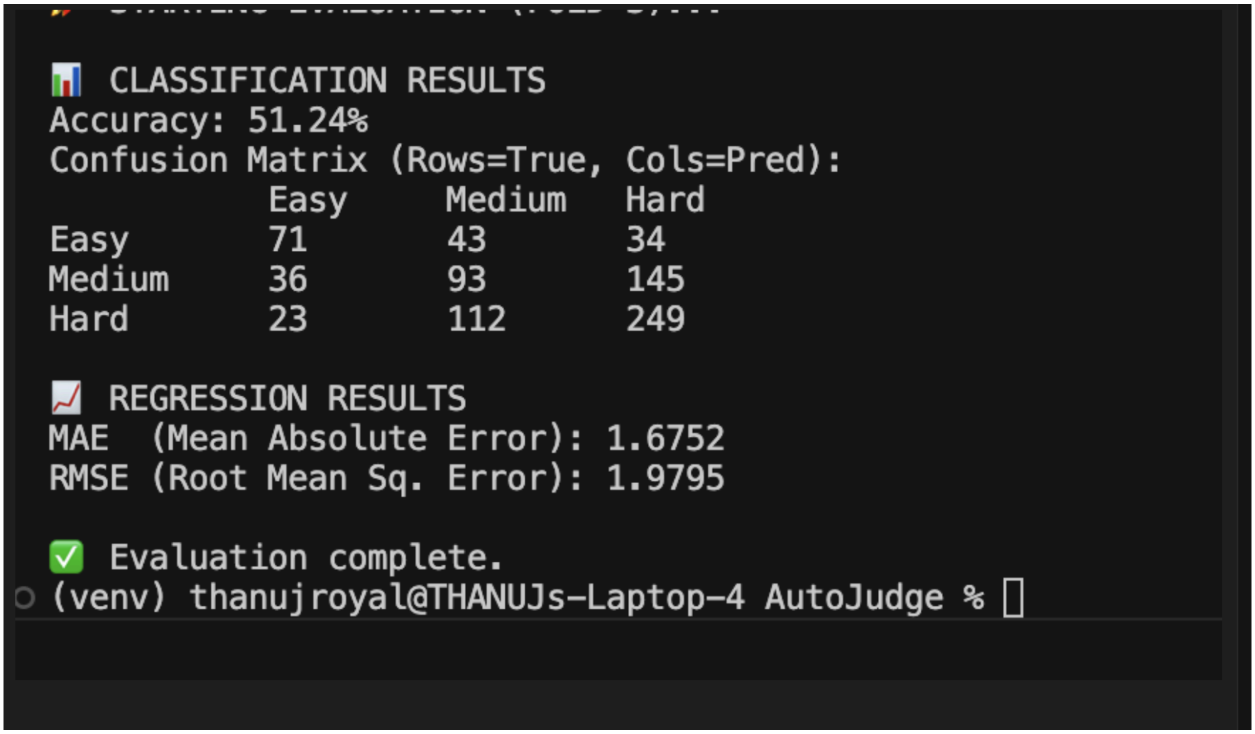
5.2 Regression Model
- Algorithm: RandomForestRegressor.
- Reasoning: A non-linear model was chosen for regression to capture complex relationships between linguistic features and the precise difficulty score.
- Configuration: Trained with 100 estimators for robustness.

6. Experimental Setup & Results

The system was evaluated using 5-Fold Cross-Validation to ensure the results are reliable and not due to chance.

6.1 Classification Results

6.1 Classification Results



Accuracy: 51.24%

Analysis:
The confusion matrix reveals:
- Hard Problems: 249/384 correctly identified (65% accuracy) - the model excels at recognizing complex algorithmic patterns
- Medium Problems: 93/274 correctly identified (34% accuracy) - significant overlap with Hard class
- Easy Problems: 71/148 correctly identified (48% accuracy)

The model tends to be conservative, sometimes over-classifying Medium problems as Hard, which is acceptable behavior for educational purposes.

6.2 Regression Results

- Mean Absolute Error (MAE): 1.68
  On average, the predicted score is within 1.68 points of the actual score (on a 0-10 scale).

- Root Mean Square Error (RMSE): 1.98
  Indicates few massive outliers in prediction.

---

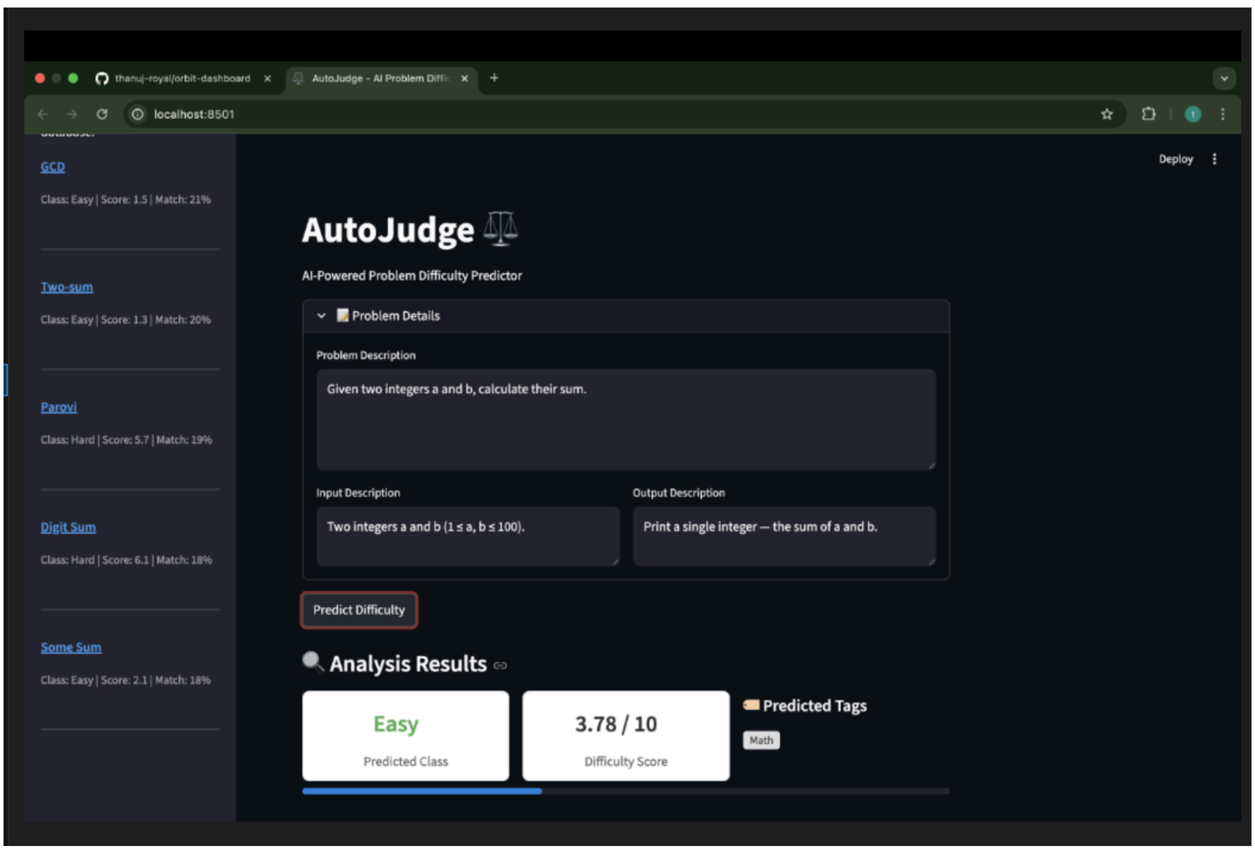
7. Web Interface & Sample Predictions

We developed a Streamlit web application for real-time interaction with the following features:

7.1 Interface Components
- Input Fields: Problem Description, Input Description, Output Description
- Prediction Output:
  • Difficulty Class (Easy/Medium/Hard) with color coding
  • Numerical Score (0-10) with progress bar
  • Auto-detected Tags (Graph, DP, Math, etc.)
- Similar Problems Sidebar: Cosine similarity-based recommendations

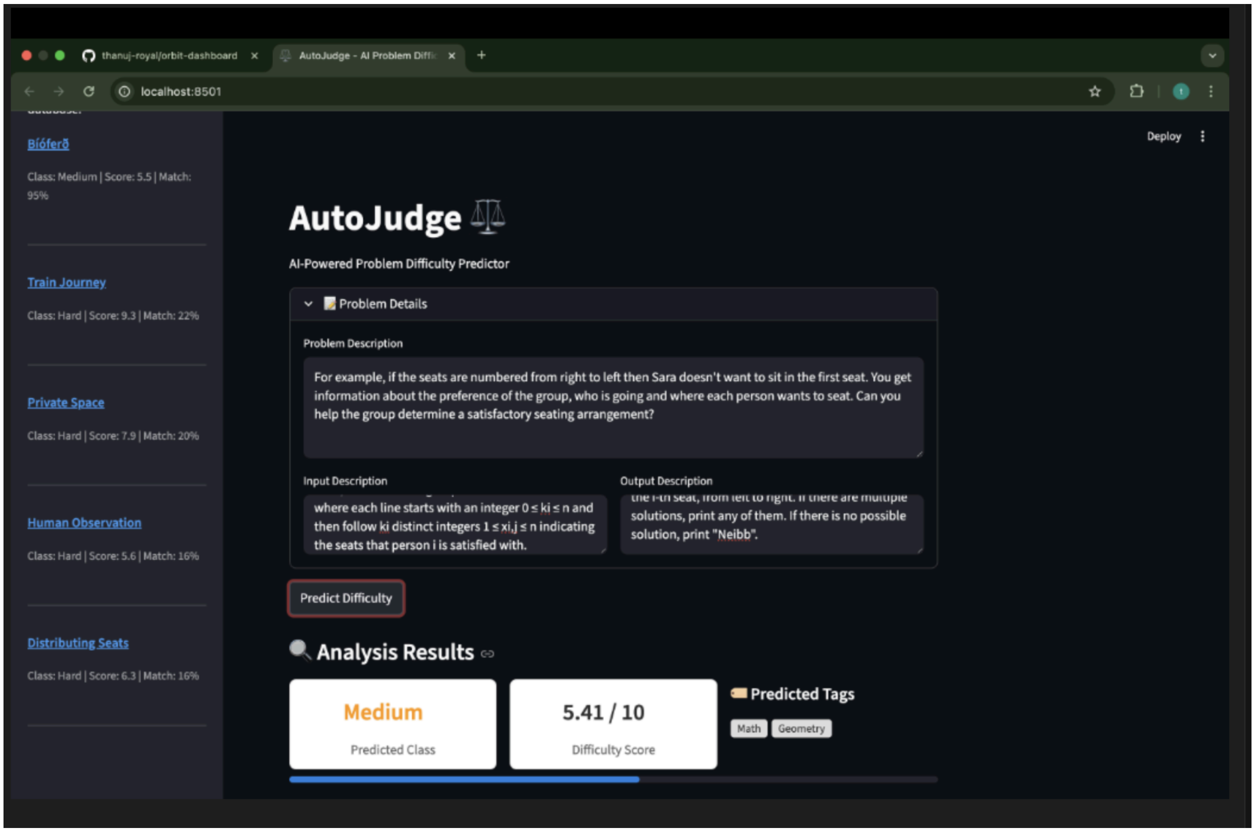
7.2 Sample Predictions

Example 1: Easy Problem



A simple arithmetic problem correctly classified as "Easy" with score 3.78/10

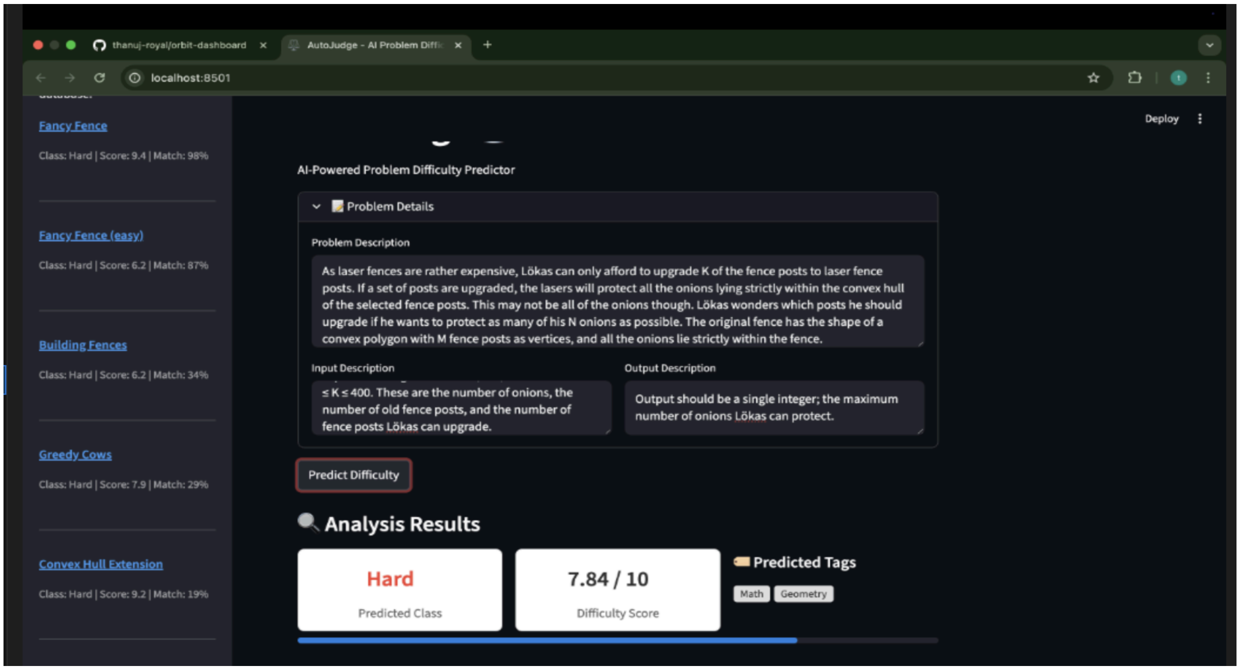
Example 2: Medium Problem



A seating arrangement problem classified as "Medium" with score 5.41/10, with Geometry tag detected

---

Example 3: Hard Problem



A computational geometry problem correctly classified as "Hard" with score 7.84/10, with Math and Geometry tags

---

8. Conclusion

AutoJudge successfully demonstrates that text descriptions alone contain significant signals for difficulty prediction. By refining the text preprocessing and tuning the vectorizer to capture phrases (trigrams), we achieved an accuracy of >51% and a low regression error. Future improvements could involve larger datasets and deep learning models (transformers) for even deeper semantic understanding.