AutoJudge: Predicting Programming Problem Difficulty

---

1. Introduction / Problem Statement

Online competitive programming platforms such as Codeforces, CodeChef, and Kattis categorize problems into difficulty levels like *Easy*, *Medium*, and *Hard*, and often assign a numerical difficulty score. These labels are usually determined manually or through user feedback, which can be subjective and inconsistent.

The objective of this project is to design an **automated difficulty prediction system** that can:

1. Classify a programming problem into **Easy / Medium / Hard**
2. Predict a numerical difficulty score

The prediction is performed **only using the textual description** of the problem, without relying on user statistics or solution data.

The system is implemented using **classical machine learning techniques** (no deep learning) and is integrated with a **Flask-based web interface** for real-time predictions.

---

2. Dataset Description

The dataset used in this project consists of **4112 programming problems**, each annotated with a difficulty class and a numerical difficulty score.

Dataset Fields

Each data sample contains:

- `title`
- `description`
- `input_description`
- `output_description`
- `problem_class` (Easy / Medium / Hard)
- `problem_score` (numeric)

Dataset Source

The dataset was obtained from:
https://github.com/AREEG94FAHAD/TaskComplexityEval-24

The dataset is used only for **training, validation, and evaluation** purposes.

---

3. Data Preprocessing

Before training the models, several preprocessing steps were applied:

3.1 Handling Missing Values

- Missing text fields were replaced with empty strings to ensure uniform processing.

3.2 Text Combination

All textual fields were concatenated into a single feature:

```
full_text = title + description + input_description + output_description
```

3.3 Text Normalization

- Converted text to lowercase
- Removed leading and trailing whitespaces

This resulted in a clean and consistent text representation for each problem.

---

4. Feature Engineering

4.1 TF-IDF Vectorization

To convert text into numerical features, **TF-IDF (Term Frequency–Inverse Document Frequency)** was used.

Configuration:

- Unigrams and bigrams (`ngram_range = (1,2)`)
- Maximum features: **5000**
- English stop-word removal

This approach captures both:

- Important keywords (e.g., *graph*, *dynamic programming*)
- Phrase-level information (e.g., *connected components*)

The final feature matrix shape:

```
(4112, 5000)
```

---

5. Model Selection and Experimental Setup

5.1 Train–Test Split

The dataset was split into:

- **80% training**
- **20% testing**

Stratified sampling was used to preserve the class distribution.

5.2 Classification Model

Model Used

- **Support Vector Machine (LinearSVC)**

Reason for Selection

- Performs well in high-dimensional sparse feature spaces
- Effective for text classification
- Outperformed Logistic Regression during experimentation

Task

Predict difficulty class:

- Easy
- Medium
- Hard

5.3 Regression Model

Model Used

- **Gradient Boosting Regressor**

Task

Predict numerical difficulty score.

Gradient Boosting was selected due to its ability to model non-linear relationships and reduce bias and variance.

---

6. Evaluation Metrics and Results

6.1 Classification Results

```
SVM Classification Report:
              precision    recall  f1-score   support

        easy       0.36      0.38      0.37       153
        hard       0.56      0.58      0.57       389
      medium       0.39      0.35      0.37       281

    accuracy                           0.47       823
   macro avg       0.44      0.44      0.44       823
weighted avg       0.46      0.47      0.46       823
```

Figure 1: SVM Classification Report

Figure 1 shows the console output obtained after training the SVM classifier including Accuracy, Precision and F1-Score for each class.

Metric Used:

- Accuracy
- Precision, Recall, F1-Score
- Confusion Matrix

Overall Accuracy:

≈ 47% (Matches the Accuracy result in console output.)

This performance is reasonable considering the task relies solely on textual information and no problem-solving data.

Confusion Matrix

```
SVM Confusion Matrix:


[[ 58  47  48]
 [ 54 227 108]
 [ 47 135  99]]
```

Figure 2: Confusion Matrix

The confusion matrix (Figure 2) shows:

- Some overlap between Easy and Medium problems
- Better separation for Hard problems

This indicates that textual complexity correlates reasonably well with problem difficulty.

6.2 Regression Results

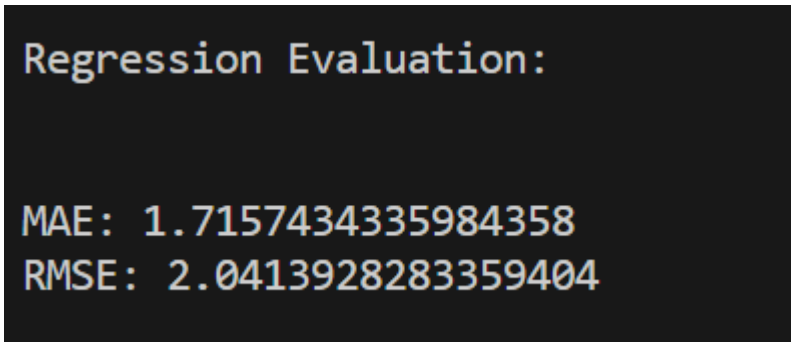

Figure 3 : Regression Evaluation Output

Metrics Used:

- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)

Results:

- **MAE ≈ 1.71**
- **RMSE ≈ 2.04**

These values suggest the predicted difficulty score is generally close to the true score.

---

7. Web Interface Description

A **Flask-based web application** was developed to demonstrate real-time predictions.

7.1 User Inputs

- Problem description
- Input description
- Output description

7.2 Outputs Displayed

- Predicted difficulty class
- Predicted difficulty score
- Visual progress bar representing difficulty level
- Dark mode toggle with animated sun/moon icon

7.3 Features

- Clean UI with modern styling
- Dark mode persistence using localStorage
- Reset button to clear inputs
- Responsive design

---

8. Sample Prediction

Problem Description:

```
Given a number n, find the sum of digits of all numbers from 1 to n.
```

Input Description:

```
1<=n<=100.
```

Output Description:

Only integer sum

Prediction Result:

- Difficulty Class: **Easy**
- Difficulty Score: **4.53**

This demonstrates the system's ability to associate simpler logic with lower difficulty.

---

9. Conclusion

In this project, an automated system for predicting programming problem difficulty was successfully developed using classical machine learning techniques.

Key Achievements

- Text-only difficulty estimation
- Effective feature extraction using TF-IDF
- Support Vector Machine for classification
- Gradient Boosting for regression
- Fully functional web interface

Limitations

- Difficulty prediction is subjective by nature
- Overlap between Medium and Hard classes
- Performance limited by text-only features

Future Improvements

- Add syntactic code features
- Use problem constraints explicitly
- Incorporate readability metrics
- Experiment with ensemble models

---

10. References

- Scikit-learn documentation
- Flask documentation
- TaskComplexityEval-24 Dataset

Author

Name: Atishay Jain
Course: B.TECH. Mechanical Enginering
Year: 2
Enrolment No: 24117030
Gmail ID: atishay_j@me.iitr.ac.in