Automatic Difficulty Prediction of Programming Problems Using Textual Features

Name: Kruti Surati -23119046
Branch: Production and Industrial Engineering, 3rd year
Project: AutoJudge – Predicting Programming Problem Difficulty

1. Introduction

Online competitive programming platforms such as LeetCode, Kattis, and Codeforces host thousands of programming problems of varying difficulty levels. Accurately determining the difficulty of a problem is essential for learners to plan their practice and for platforms to organize content effectively. Traditionally, problem difficulty is manually assigned by experts or derived from user statistics, which can be subjective and time-consuming.

This project aims to **automatically predict the difficulty of programming problems using only textual information**. Two tasks are addressed:

1. **Classification:** Predict the difficulty class of a problem (Easy / Medium / Hard).
2. **Regression:** Predict a continuous numerical difficulty score.

The system uses **natural language processing (NLP)** techniques to extract features from problem statements and machine learning models to make predictions. A simple **web-based user interface** is also developed to allow users to input a problem and obtain predictions.

2. Problem Statement

The objectives of this project are:

- To build a machine learning system that works **only on textual data.**
- To predict:
    - **Problem difficulty class** (Easy / Medium / Hard)
    - **Problem difficulty score** (continuous value)

- To evaluate the system using standard metrics such as:
    - Accuracy and confusion matrix for classification
    - MAE and RMSE for regression
- To provide predictions through a **simple web interface.**

3. Dataset Description

3.1 Dataset Source

The dataset consists of programming problems collected from online competitive programming platforms. Each problem is stored as a JSON object in a .jsonl (JSON Lines) file.

3.2 Dataset Size

- Total number of problems: **~4,100**
- Each row represents one programming problem

3.3 Dataset Fields Used

The following fields are used:

- title – Problem title
- description – Full problem description
- input_description – Input format
- output_description – Output format
- problem_class – Difficulty class (Easy / Medium / Hard)
- problem_score – Numerical difficulty score

Only textual information is used as input features, satisfying the project constraint.

4. Data Preprocessing

4.1 Loading the Dataset

The dataset is loaded from a .jsonl file using pandas, where each line is parsed as a separate JSON object.

4.2 Handling Missing Values

- All textual fields were checked for missing values.
- Missing values were replaced with empty strings ("").
- This ensures consistency during text vectorization

4.3 Text Cleaning and Combination

- To provide richer context to the model, multiple text fields were combined into a single feature:
- combined_text = title + description + input_description + output_description
- This approach helps the model understand the complete structure and intent of each problem.

5. <u>Feature Engineering</u>

5.1 Text Vectorization

Textual data cannot be directly used by machine learning models. Therefore, **TF-IDF (Term Frequency–Inverse Document Frequency)** vectorization was applied.

TF-IDF characteristics:

- Captures important words
- Reduces the impact of common stop words
- Converts text into numerical feature vectors

  5.2 Vectorizer Configuration
- Maximum features: **5000**
- Stop words: English
- Output: Sparse numerical matrix

---

6. <u>Machine Learning Models</u>

6.1 Classification Model

Objective: Predict difficulty class (Easy / Medium / Hard)

Model Used: Logistic Regression
Reason for selection:

- Works well with high-dimensional sparse data
- Interpretable and fast
- Suitable baseline for text classification

6.2 Regression Model

Objective: Predict difficulty score

Model Used: Linear Regression
Reason for selection:

- Simple and interpretable
- Works effectively with TF-IDF features
- Suitable for continuous target prediction

7. Experimental Setup

7.1 Train-Test Split

- Training set: **80%**
- Test set: **20%**
- Random state fixed for reproducibility

7.2 Tools and Libraries

- Python
- Pandas, NumPy
- Scikit-learn
- Streamlit (for web interface)
- Joblib (for model persistence)

8. Evaluation and Results

8.1 Classification Results

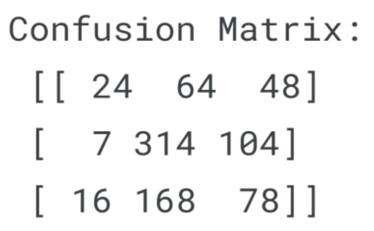
Accuracy: 50.5%

Several classical machine learning models were experimented with in order to improve the difficulty classification performance. These included Logistic Regression, Support Vector Machines (SVM), and different TF-IDF configurations such as varying maximum feature sizes and n-gram ranges. Additionally, multiple train–test splits and hyperparameter settings were evaluated to assess their impact on classification accuracy. Despite these efforts, the overall accuracy showed only marginal variation, indicating that the problem difficulty prediction task is inherently challenging when relying solely on textual problem descriptions.

The final classification model achieved an accuracy of approximately **50–53%** on the test dataset. Although this accuracy may appear moderate, it is important to note that problem difficulty classification is a highly subjective task. The perceived difficulty of programming problems can vary significantly across users depending on their background, experience, and familiarity with specific problem types. Furthermore, the model operates exclusively on textual information without access to solution code, runtime constraints, or problem-solving statistics, which naturally limits predictive performance.

Confusion Matrix-

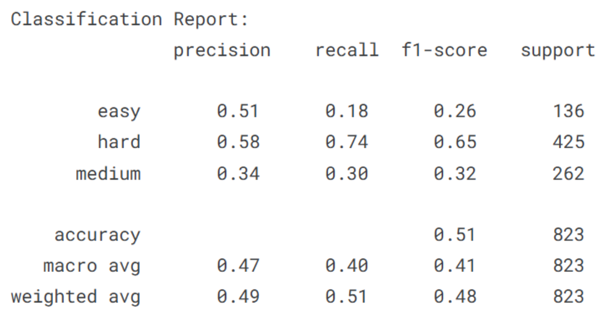
The confusion matrix shows:



```
Confusion Matrix:
 [[ 24  64  48]
 [  7 314 104]
 [ 16 168  78]]
```

- Stronger performance for **Hard** problems
- Moderate confusion between **Easy** and **Medium**
- Expected overlap due to subjective boundaries

Precision, Recall, F1-score



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

        easy       0.51      0.18      0.26       136
        hard       0.58      0.74      0.65       425
      medium       0.34      0.30      0.32       262

    accuracy                           0.51       823
   macro avg       0.47      0.40      0.41       823
weighted avg       0.49      0.51      0.48       823
```

- Hard problems achieved the highest recall
- Medium problems were the most challenging to classify
- Overall performance is reasonable for a text-only model

8.2 Regression Results

- **Mean Absolute Error (MAE):** 1.73
- **Root Mean Squared Error (RMSE):** 2.07

These results indicate that the predicted difficulty scores are reasonably close to actual values given the inherent subjectivity of difficulty scoring.

9. Web Interface

9.1 Description

A simple **Streamlit-based web application** was developed to allow users to interact with the model.

9.2 User Inputs

The interface provides text boxes for:

- Problem Title
- Problem Description
- Input Description
- Output Description

In addition to the problem, input, and output descriptions, the problem title was also included as part of the textual input to provide additional contextual information. Only textual fields were used for prediction, in accordance with the project requirements.

9.3 Output Displayed

Upon clicking **Predict**, the system displays:

- Predicted difficulty class (Easy / Medium / Hard)
- Predicted difficulty score

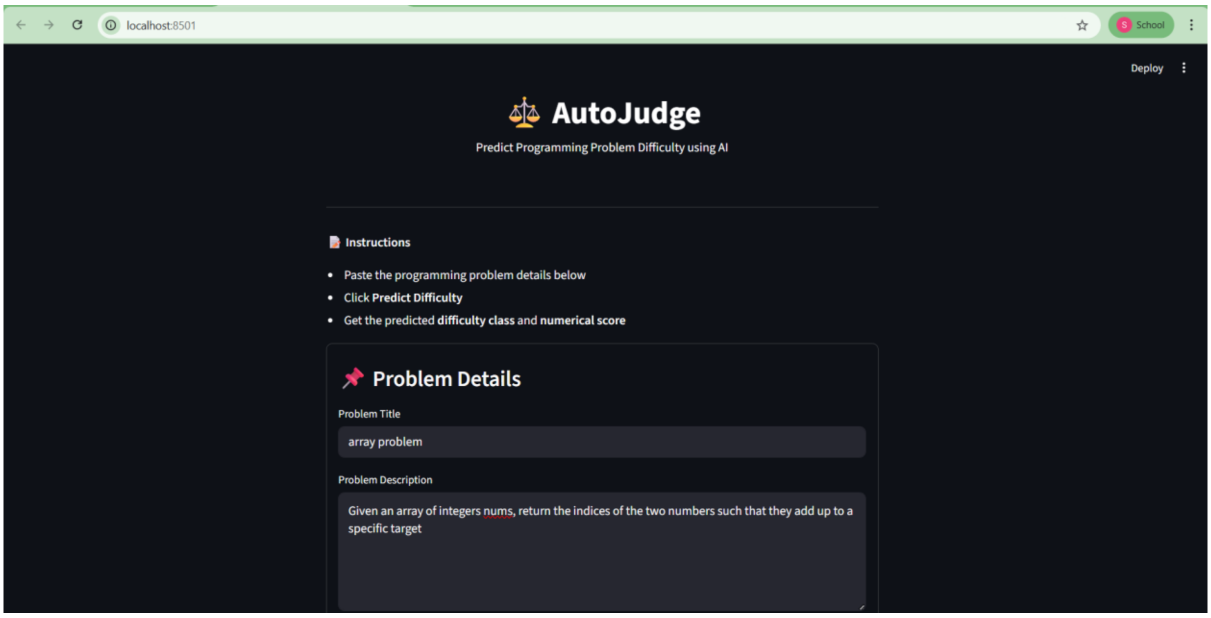
9.4 Sample Prediction

Example:

- **Problem title:** array problem
- **Problem Description:** Given an array of integers nums, return the indices of the two numbers such that they add up to a specific target.
- **Input Description:** nums = [2,7,11,15], target = 9
- **Output Description:** Return [0,1]
- **Predicted Difficulty Class:** MEDIUM
- **Predicted Difficulty Score :**4.76

The interface runs locally and does not require authentication or a database.

AutoJudge Web Interface with Sample Prediction-



⚖️ AutoJudge

Predict Programming Problem Difficulty using AI

Instructions

- Paste the programming problem details below
- Click **Predict Difficulty**
- Get the predicted **difficulty class** and **numerical score**

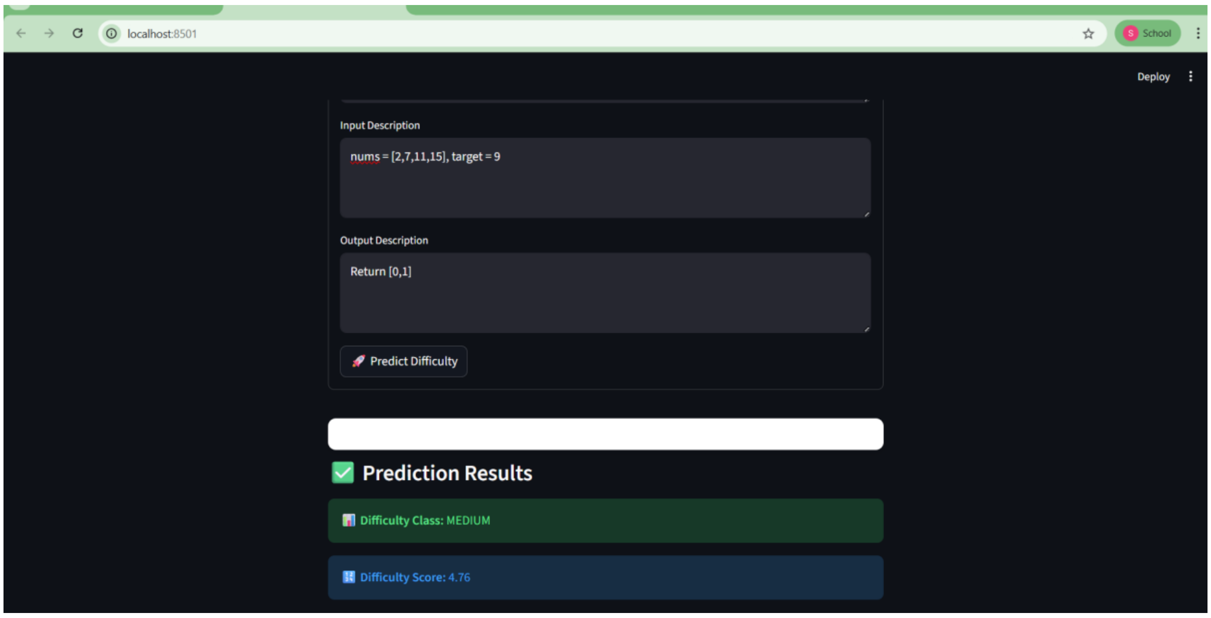
📌 Problem Details

Problem Title

array problem

Problem Description

Given an array of integers nums, return the indices of the two numbers such that they add up to a specific target



Input Description

```
nums = [2,7,11,15], target = 9
```

Output Description

```
Return [0,1]
```

Prediction Results

Difficulty Class: MEDIUM

Difficulty Score: 4.76

10. Model Persistence

The trained components are saved using Joblib:

- tfidf_vectorizer.pkl
- difficulty_classifier.pkl
- difficulty_regressor.pkl

  This allows reuse of trained models without retraining.

11. Future Work

- Can use advanced models (SVM, Random Forest, or Transformers)
- Address class imbalance with resampling
- Improve UI with confidence scores
- Extend to multi-platform datasets

12. Conclusion

- This project demonstrates that **programming problem difficulty can be reasonably predicted using only textual information**. By combining NLP techniques with classical machine learning models, both categorical and numerical difficulty predictions were achieved. The system satisfies all project requirements, including evaluation, saved models, and a working web interface.