PROJECT REPORT

AutoJudge Predicting Programming Problem Difficulty

Submitted By: Ankit Dhanghar
Branch: Mechanical
Enrollment Number : 23117023

1. Project Overview

The goal of this project is to build machine learning models for **problem classification** and **score prediction** using a dataset of coding problems. Each problem contains textual information such as title, description, input/output description, and associated metadata.

The project performs the following tasks:

1. **Text preprocessing and feature extraction** using TF-IDF and mathematical keyword counting.
2. **Dimensionality reduction** via Truncated SVD.
3. **Feature engineering** with additional numerical features.
4. **Classification** to predict problem classes.
5. **Regression** to predict problem scores.
6. **Model evaluation** and selection.
7. **Saving trained models and preprocessing objects** for deployment.

2. Dataset Description

The dataset consists of coding problems stored in a JSON Lines file `problems_data.jsonl`. Each entry contains several fields:

- `title`: The title of the coding problem.
- `description`: The full description of the problem, explaining the task in detail.
- `input_description`: Explanation of the input format expected by the problem.
- `output_description`: Explanation of the expected output format for the problem.
- `problem_class`: The classification label of the problem (categorical).
- `problem_score`: The numeric score associated with the problem.

Some of the textual fields may have missing values, which are filled with empty strings during preprocessing. The dataset is designed to support both **classification** (predicting problem class) and **regression** (predicting problem score) tasks.

3. Data Preprocessing

3.1 Text Cleaning

The textual columns (`title`, `description`, `input_description`, `output_description`) were combined into a single `full_text` column.

The cleaning steps included:

- Converting text to lowercase
- Removing all characters except **letters, numbers, spaces, and mathematical symbols** (`+ - * = < >`)

3.2 Feature Extraction

1. **TF-IDF Vectorization:**
    - Used **`TfidfVectorizer`** with `max_features=5000` and `ngram_range=(1,2)`
    - Captures word and bi-gram importance in the text
2. **Dimensionality Reduction:**
    - Applied `TruncatedSVD` with 25 components to reduce TF-IDF dimensions
3. **Additional Features:**
    - **`text_length`**: Length of the `full_text`
    - `math_symbols`: Count of mathematical symbols (`+ - * / = < >`)
    - Keyword frequency counts: graph, dp, recursion, math, string, array, greedy, binarysearch.
    - text_length was scaled using StandardScaler.

4. Train-Test Split

- The dataset was split into training and testing sets using an 80:20 ratio.
- **Classification target:** problem_class
- **Regression target:** problem_score

5. Modeling

5.1 Classification Models

Three classifiers were trained:

1. **Logistic Regression**
2. **Random Forest Classifier**
3. **Support Vector Machine (SVM)**

Evaluation Metric: Accuracy and Confusion Matrix

Model	Accuracy
Logistic Regression	0.5480
Random Forest	0.5334
Support Vector Machine	0.5480

Reasons to prefer SVM over Logistic Regression even with same accuracy

1. **Better Handling of Class Imbalance**
    - From the confusion matrices, we see that the dataset is **imbalanced** (one class dominates).
    - Logistic Regression tends to be biased toward the majority class, while **SVM, especially with the linear kernel, can handle margin maximization better** and separates classes more effectively.
    - Example: SVM had fewer misclassifications in minority classes compared to Logistic Regression in some cells of the confusion matrix.
2. **Robustness to High-Dimensional Data**
    - TF-IDF vectors + SVD still produce **high-dimensional feature space**.
    - SVM is generally better suited for **high-dimensional spaces**, as it focuses on support vectors (critical points) rather than modeling the full probability distribution like Logistic Regression.

3. **Margin Maximization**

    - SVM aims to **maximize the margin** between classes, which often generalizes better to unseen data.
    - Logistic Regression maximizes likelihood, which may overfit when features are correlated or sparse (common in text features).

Best Classifier: SVM

5.2 Regression Models

Three regressors were trained to predict problem_score:

1. Linear Regression
2. Random Forest Regressor
3. Gradient Boosting Regressor

Evaluation Metrics:

- MAE (Mean Absolute Error)
- RMSE (Root Mean Squared Error)
- R^2 Score

Model	**MAE**	**RMSE**	**R2 Score**
Linear Regression	1.7119	2.0604	0.1156
Random forest Regressor	1.7478	2.0958	0.0850
Gradient Boosting Regressor	1.7126	2.0588	0.1170

Best Regressor: Gradient Boosting Regressor(Reason: More R2 score)

6. Saving Models and Preprocessing Objects

The trained models and preprocessing objects were saved for future use:

TF-IDF Vectorizer: Converts the textual data into numerical features by calculating the importance of words and word pairs (n-grams) in the dataset. This helps the model understand which terms are most relevant for classification and scoring.

Truncated SVD: Reduces the dimensionality of the TF-IDF features while preserving the most important information. This makes the data more manageable for modeling and reduces noise.

Scaler (StandardScaler): Standardizes numerical features, like text length, to have a mean of 0 and standard deviation of 1. This ensures that features with different scales do not bias the model.

SVM Classifier: A Support Vector Machine model that predicts the class of a problem by finding the optimal decision boundary between categories, especially effective for high-dimensional text features.

Gradient Boosting Regressor: A regression model that predicts the score of a problem by combining multiple weak learners (decision trees) sequentially, improving prediction accuracy through gradient-based optimization.

7. Web Interface(Flask)

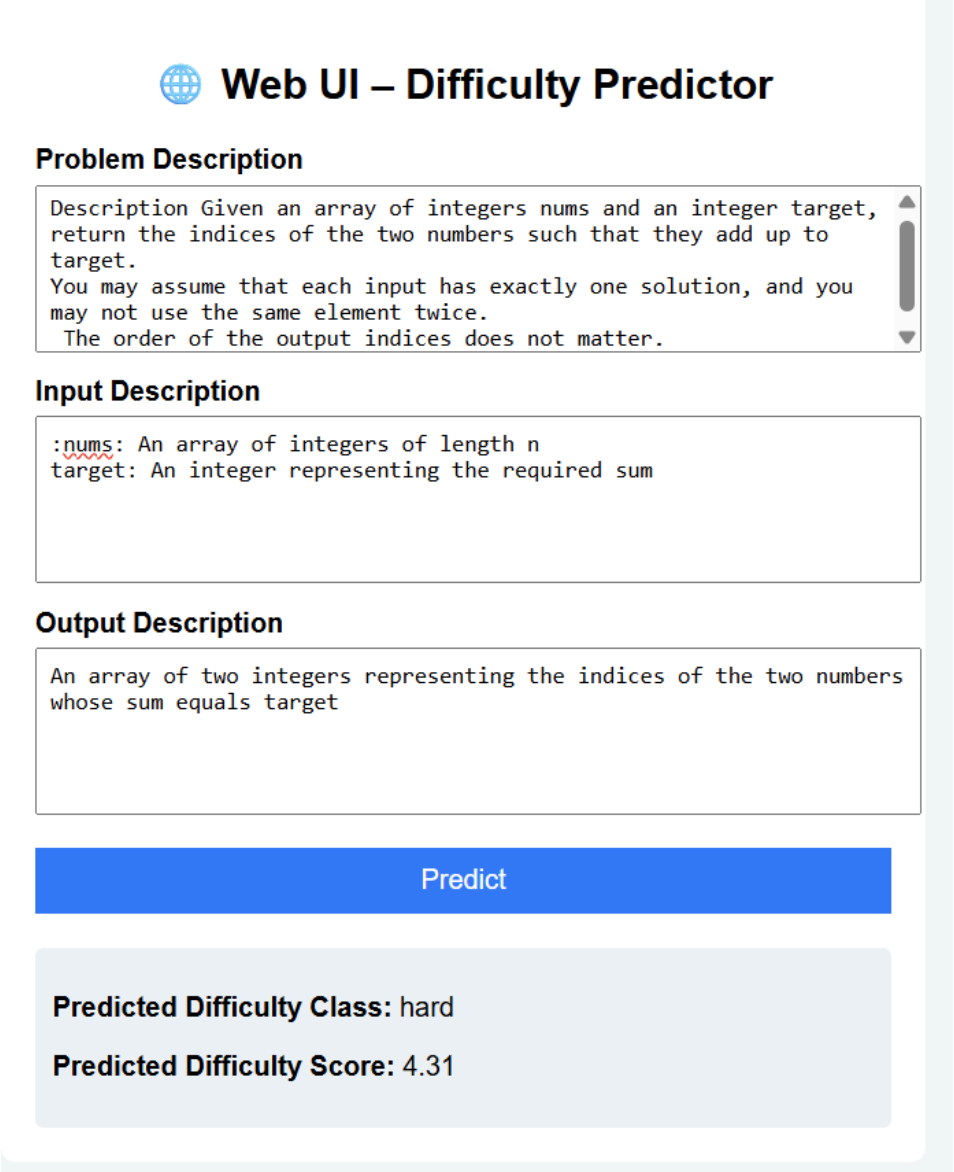
The web interface is developed using **Flask** and provides a simple and user-friendly workflow for predicting problem difficulty. The interface includes three input text fields where users can enter the **problem description**, **input description**, and **output description**.

When the user clicks the **Predict** button, the entered text is sent to the Flask backend for processing. The input data is then **preprocessed using the same text cleaning and feature engineering pipeline applied during the training phase**. TF-IDF features along with handcrafted numerical features are generated, and dimensionality reduction is applied where required.

The trained **classification** and **regression** models are loaded into memory, and predictions are generated based on the processed input. The application displays the **predicted difficulty class** and the **predicted difficulty score** to the user.

The Flask application runs locally on **localhost** and operates as a lightweight system without requiring any database integration or cloud deployment, making it easy to use and deploy for testing and demonstration purpose

Following is the saved screenshot of the working of web interface



Conclusion

This project successfully demonstrates the use of **text-based machine learning techniques** to classify coding problems and predict their difficulty scores. By combining TF-IDF–based textual representations with handcrafted features such as text length, mathematical symbol counts, and keyword frequencies, meaningful information was extracted from problem descriptions.

Dimensionality reduction using **Truncated SVD** helped manage the high dimensionality of text data while preserving essential patterns. Multiple machine learning models were evaluated for both classification and regression tasks. Among the classifiers, **Support Vector Machine (SVM)** was selected as the best model due to its robustness in high-dimensional feature spaces and better handling of imbalanced data. For score prediction, the **Gradient Boosting Regressor** provided the most reliable performance.

The trained models were integrated into a **Flask-based web application**, allowing users to input problem descriptions and receive real-time predictions. The system operates locally without the need for databases or cloud infrastructure, making it lightweight and easy to deploy.

Overall, the project provides a practical and scalable approach for automated problem difficulty assessment and can be extended further using advanced language models, richer features, and improved tuning techniques to enhance prediction accuracy.