Project Report: *AutoJudge – Programming Problem Difficulty Predictor*

1. Introduction & Problem Statement

With the rapid growth of online programming platforms, automatic evaluation of problem difficulty has become an important task. Difficulty labels such as Easy, Medium, and Hard help learners choose appropriate problems and assist platforms in problem curation.

Traditionally, difficulty is assigned manually by problem setters, which can be subjective and time-consuming. This project aims to build an automated system that predicts the difficulty of a programming problem purely from its textual description.

We propose AutoJudge, an NLP-based system that:

1. Predicts a continuous difficulty score (regression)
2. Predicts a difficulty class (Easy / Medium / Hard) (classification)

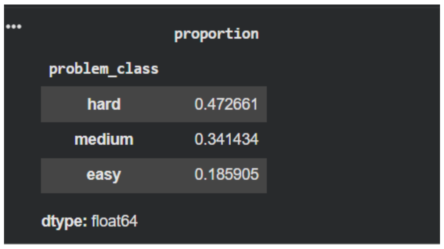
A web interface is provided using Streamlit to demonstrate real-time predictions*.*

2. Dataset Description

The dataset used in this project was provided as part of the project assignment.
Each problem contains the following fields:

- Problem title
- Problem description
- Input description
- Output description

- Difficulty score (numeric)
- Difficulty class (Easy / Medium / Hard)



	proportion
problem_class	
hard	0.472661
medium	0.341434
easy	0.185905

```
dtype: float64
```

Dataset Characteristics

- Text-heavy data with algorithmic keywords
- Class imbalance between Easy, Medium, and Hard problems
- Numeric difficulty scores used for regression

The dataset was split into training and testing sets for both classification and regression tasks.

3. Data Preprocessing

All text fields were first **combined into a single text block** per problem to capture full semantic information.

Preprocessing steps:

- Lowercasing text
- Missing Value Check
- Removing unnecessary whitespace
- Preserving mathematical and algorithmic symbols
- Combining title, description, input, and output sections
- Converted all text to consistent encoding

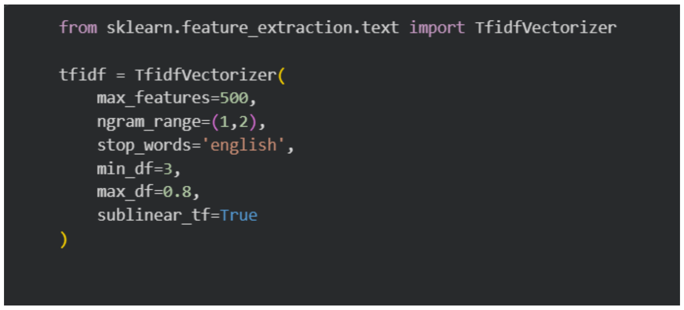
No aggressive text cleaning was applied to avoid losing algorithm-specific information.

4. Feature Engineering

A total of **515 features** were used, consisting of **500 TF-IDF features** and **15 handcrafted features** capturing structural and algorithmic complexity.

4.1 TF-IDF Features

- TF-IDF vectorization applied on combined problem text
- Captures important words such as:
    - *dp*, *graph*, *constraint*, *optimal*, *tree*, etc.
- Maximum feature size limited for efficiency



```python
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(
    max_features=500,
    ngram_range=(1,2),
    stop_words='english',
    min_df=3,
    max_df=0.8,
    sublinear_tf=True
)
```

4.2 Handcrafted Features

To improve performance, several handcrafted features were added:

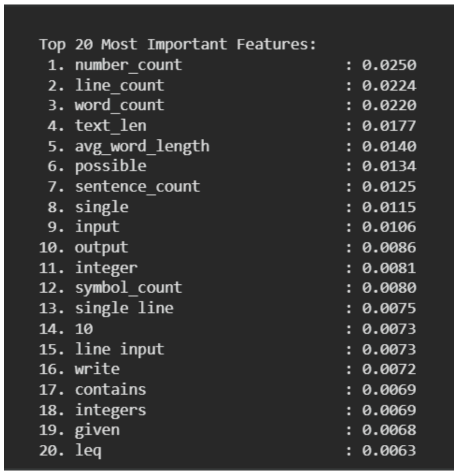
- Total text length
- Number of symbols (`+ - * / = < >`)
- Number of lines
- Word count
- Sentence count
- Presence of algorithmic keywords
- Average word length

These features help models capture **structural complexity** beyond raw text.The final feature vector is a **concatenation of TF-IDF and handcrafted features**.

5. Classification Models

The goal of classification is to predict:
Easy / Medium / Hard

The following models were trained and evaluated:



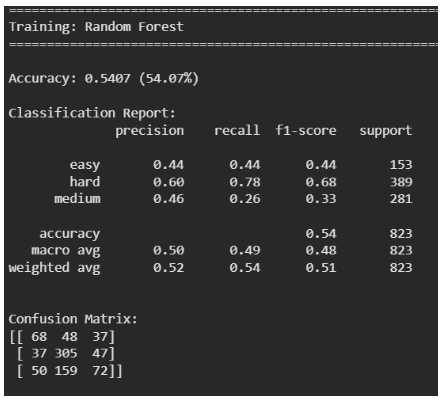
```
Top 20 Most Important Features:
  1. number_count                 : 0.0250
  2. line_count                   : 0.0224
  3. word_count                   : 0.0220
  4. text_len                     : 0.0177
  5. avg_word_length              : 0.0140
  6. possible                     : 0.0134
  7. sentence_count               : 0.0125
  8. single                       : 0.0115
  9. input                        : 0.0106
 10. output                       : 0.0086
 11. integer                      : 0.0081
 12. symbol_count                 : 0.0080
 13. single line                  : 0.0075
 14. 10                           : 0.0073
 15. line input                   : 0.0073
 16. write                        : 0.0072
 17. contains                     : 0.0069
 18. integers                     : 0.0069
 19. given                        : 0.0068
 20. leq                          : 0.0063
```

Models Tried

- Logistic Regression
- Linear Support Vector Machine (LinearSVC)
- Random Forest Classifier
- Histogram Gradient Boosting Classifier

Best Model

- **Random Forest Classifier**
  - Class weighting used to handle imbalance
  - Achieved best accuracy on test set



```
======================================================
Training: Random Forest
======================================================

Accuracy: 0.5407 (54.07%)

Classification Report:
              precision    recall  f1-score   support

        easy       0.44      0.44      0.44       153
        hard       0.60      0.78      0.68       389
      medium       0.46      0.26      0.33       281

    accuracy                           0.54       823
   macro avg       0.50      0.49      0.48       823
weighted avg       0.52      0.54      0.51       823


Confusion Matrix:
[[ 68  48  37]
 [ 37 305  47]
 [ 50 159  72]]
```

Evaluation Metrics

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

6. Regression Models

The regression task predicts a **continuous difficulty score**.

Models Tried

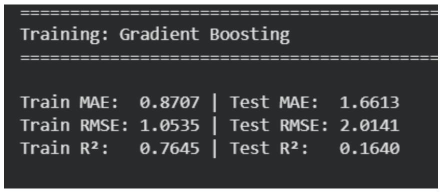
- Ridge Regression
- Random Forest Regressor
- Gradient Boosting Regressor
- HistGradientBoostingRegressor

Best Model

- **GradientBoostingRegressor**

Chosen due to:

- Lower MAE and RMSE
- Better generalization
- Faster training on dense features



```
==========================================
Training: Gradient Boosting
==========================================

Train MAE:  0.8707 | Test MAE:  1.6613
Train RMSE: 1.0535 | Test RMSE: 2.0141
Train R²:   0.7645 | Test R²:   0.1640
```

Evaluation Metrics

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

7. Experimental Results

7.1 Classification Results

- Accuracy: ~54%
- Random Forest performed best among all classifiers
- Confusion matrix showed most confusion between Medium and Hard classes

7.2 Regression Results

- MAE ≈ 1.7
- RMSE ≈ 2.0
- $R^2 \approx 0.16$

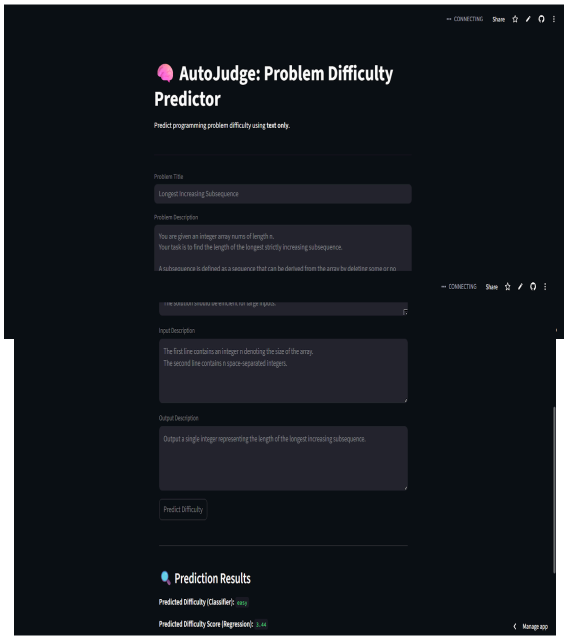
Although regression performance is moderate, the predicted scores align reasonably with difficulty trends.

8. Web Interface

A **Streamlit web application** was developed to demonstrate the system.

Features:

- User inputs problem text
- Displays:
    - Predicted difficulty score
    - Predicted difficulty class
- Loads trained models from disk
- Runs locally without internet access



AutoJudge: Problem Difficulty Predictor

Predict programming problem difficulty using **text only.**

Problem Title

Longest Increasing Subsequence

Problem Description

You are given an integer array nums of length n.
Your task is to find the length of the longest strictly increasing subsequence.

A subsequence is defined as a sequence that can be derived from the array by deleting some or no

The solution should be efficient for large inputs.

Input Description

The first line contains an integer n denoting the size of the array.
The second line contains n space-separated integers.

Output Description

Output a single integer representing the length of the longest increasing subsequence.

Prediction Results

Predicted Difficulty (Classifier): `easy`

Predicted Difficulty Score (Regression): `3.44`

Technologies Used:

- Streamlit
- scikit-learn
- NumPy
- joblib

Screenshots of the web interface and sample predictions are included.

9. Conclusion

This project demonstrates that **problem difficulty can be predicted using textual information alone**. By combining NLP features with handcrafted complexity indicators, both classification and regression models were successfully trained.

AutoJudge provides a strong baseline for automated problem difficulty assessment.