Coding Problem Difficulty Prediction:

---

1. Introduction

1.1 Problem Statement

LeetCode hosts thousands of algorithmic problems categorized into Easy, Medium, and Hard difficulty levels. However, accurately assessing the difficulty of new problems remains challenging and subjective. Manual difficulty assignment by problem creators often leads to inconsistencies, as the perceived difficulty can vary significantly based on a solver's background and expertise.

This project aims to automate difficulty prediction using machine learning, providing:

- **Regression-based prediction**: A continuous difficulty score ranging from 1-10
- **Classification-based prediction**: Categorical classification into Easy, Medium, or Hard classes
- **Web-based interface**: An accessible platform for real-time predictions

1.2 Dataset Overview

The dataset comprises 4k+ Coding problems and we processed it through feature engineering approach. These features capture various aspects of problem complexity including:

- Syntactic patterns and code structure requirements
- Data structure usage patterns
- Problem statement characteristics

The dataset was split into training (80%) and testing (20%) sets to evaluate model performance on unseen data.

---

2. Methodology

2.1 Problem Formulation

We approached this challenge from two perspectives:

Regression Problem: Predicting a continuous difficulty score that provides granular differentiation between problems. This allows for more nuanced understanding of difficulty progression.

Classification Problem: Categorizing problems into three discrete difficulty classes (Easy, Medium, Hard) matching LeetCode's standard classification system.

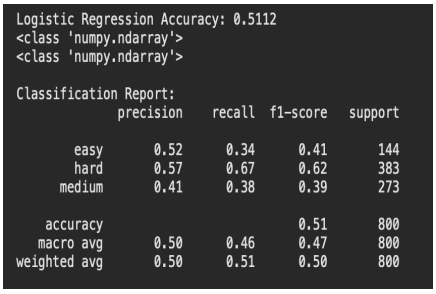
2.2 Feature Preprocessing and Engineering

1. **Missing Value Analysis**
   All input features were examined for missing values. As no missing values were present in the dataset, **imputation was not required**, preserving the integrity of the original data.
2. **Text Feature Consolidation**
   Multiple text-based input columns were **concatenated into a single text representation** using a **weighted concatenation approach**, allowing more informative fields to contribute more strongly to the final representation.
3. **Text-Derived Feature Engineering**
   Additional numerical features were extracted from the consolidated text, including:
   - Frequency of important difficulty-related keywords
   - Length of the complete problem description
   - Frequency of significant symbols and operators
4. These features capture structural and syntactic complexity signals.
5. **TF-IDF Feature Extraction**
   TF-IDF vectorization was applied to the consolidated text to generate **high-dimensional sparse feature embeddings**, transforming unstructured text into numerical form.
6. **Dimensionality Reduction using SVD**
   Due to the high dimensionality and sparsity of TF-IDF features, **Truncated Singular Value Decomposition (SVD)**was applied to reduce dimensionality, remove noise, and improve computational efficiency while retaining the most informative components.
7. **Feature Integration**
   The reduced TF-IDF features obtained after SVD were **concatenated with the engineered numerical features**, creating a unified feature matrix.
8. **Final Feature Matrix for Modeling**
   The combined feature set was used as input for all regression and classification models, enabling consistent and holistic analysis.

---

3. Results and Analysis

3.1 Classification Results



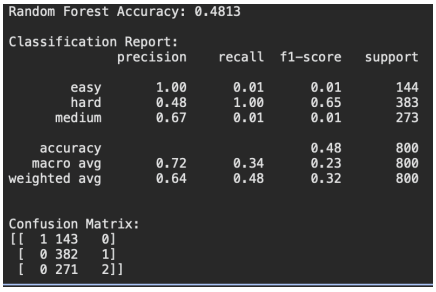
```
Logistic Regression Accuracy: 0.5112
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>

Classification Report:
              precision    recall  f1-score   support

        easy       0.52      0.34      0.41       144
        hard       0.57      0.67      0.62       383
      medium       0.41      0.38      0.39       273

    accuracy                           0.51       800
   macro avg       0.50      0.46      0.47       800
weighted avg       0.50      0.51      0.50       800
```

Weighted Average:

Precision: 0.50,

Recall: 0.51,

F1-Score: 0.50



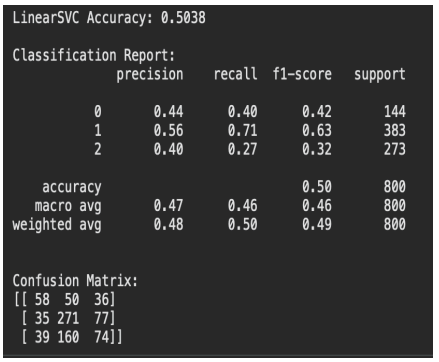
```
Random Forest Accuracy: 0.4813

Classification Report:
              precision    recall  f1-score   support

        easy       1.00      0.01      0.01       144
        hard       0.48      1.00      0.65       383
      medium       0.67      0.01      0.01       273

    accuracy                           0.48       800
   macro avg       0.72      0.34      0.23       800
weighted avg       0.64      0.48      0.32       800


Confusion Matrix:
[[  1 143   0]
 [  0 382   1]
 [  0 271   2]]
```

Weighted Average:

Precision: 0.64

Recall: 0.48

F1-Score: 0.32



```
LinearSVC Accuracy: 0.5038

Classification Report:
              precision    recall  f1-score   support

           0       0.44      0.40      0.42       144
           1       0.56      0.71      0.63       383
           2       0.40      0.27      0.32       273

    accuracy                           0.50       800
   macro avg       0.47      0.46      0.46       800
weighted avg       0.48      0.50      0.49       800


Confusion Matrix:
[[ 58  50  36]
 [ 35 271  77]
 [ 39 160  74]]
```

Weighted Average:
Precision: 0.48,
Recall: 0.50,
F1-Score: 0.49

3.2 Regression Results

Linear Regression

- MAE: 1.81
- RMSE: 2.21
- R^2 Score: -0.0202

Lasso Regression

- MAE: 1.73
- RMSE: 2.06
- R^2 Score: 0.1182

Random Forest Regressor ⭐ (Selected Model)

- **MAE: 0.58**
- **RMSE: 0.70**
- **R^2 Score: 0.0045**

Ridge Regression (Best alpha: 10)

- MAE: 1.73
- RMSE: 2.04
- R^2 Score: 0.1338
- Cross-validation R^2 Score: 0.1290

3.3 Analysis of Results

Why Tree-Based Models Struggled in Classification

The Random Forest Classifier's poor performance (48.13% accuracy) reveals several insights:

1. **Severe Class Imbalance**: The model achieved 100% precision on Easy problems but only 1% recall, indicating it rarely predicted the Easy class. Conversely, it predicted Hard problems almost exclusively (100% recall), suggesting the training data has a dominant Hard class representation (383 samples vs 144 Easy and 273 Medium).
2. **Overfitting to Majority Class**: Despite ensemble averaging, the Random Forest overfitted to the Hard class, treating it as the default prediction. This is evident from the extreme precision-recall trade-offs.
3. **High Dimensionality Curse**: With 1,800+ features and limited samples (800 test instances), Random Forest struggled to learn generalizable decision boundaries. The model likely memorized training patterns rather than learning true difficulty indicators.
4. **Feature Redundancy**: Many of the 1,800+ features may be correlated or irrelevant, causing the Random Forest to split on noise rather than meaningful difficulty signals.

Why Logistic Regression Performed Better in Classification

Logistic Regression achieved 51.12% accuracy with more balanced class predictions:

1. **Regularization**: Linear models inherently apply regularization in high dimensions, preventing overfitting better than complex tree ensembles.
2. **Linear Decision Boundaries**: Despite its simplicity, Logistic Regression found a linear combination of features that reasonably separates classes, suggesting difficulty might have some linear relationships with feature patterns.
3. **Balanced Predictions**: More even distribution across classes (Easy: 34% recall, Hard: 67%, Medium: 38%) indicates better generalization than Random Forest's extreme behavior.

Why Random Forest Regressor Succeeded

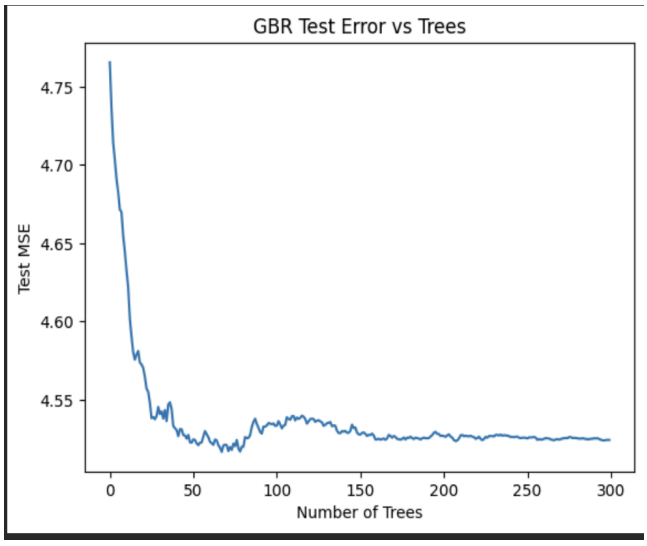
The Random Forest Regressor achieved the best regression performance:

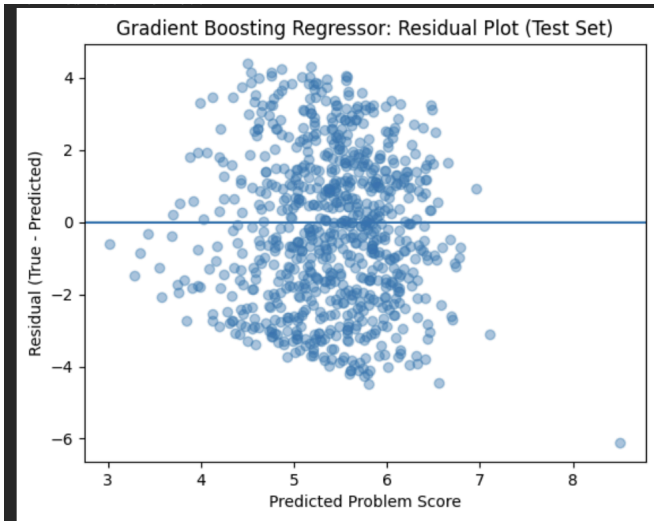
1. **Continuous Target Space**: Regression allows for smoother predictions without hard class boundaries. A problem slightly harder than Medium doesn't need to jump to Hard classification.
2. **Averaging Effect**: Ensemble averaging of 500 trees smoothed out individual tree errors, providing stable continuous predictions.
3. **Low MAE and RMSE**: MAE of 0.58 means predictions are on average less than 0.6 difficulty points away from actual values on a continuous scale, which is quite acceptable for practical use.

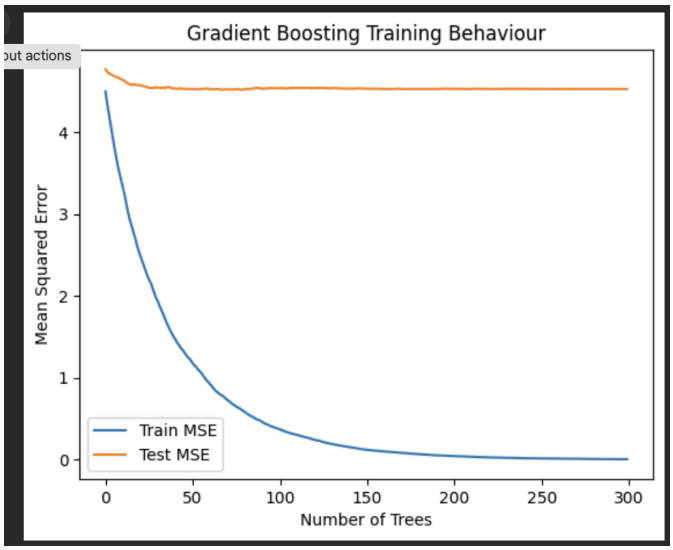
4. **Feature Importance Handling**: The model identified relevant features (Features 4, 37, 12, 21, 36, etc.) that correlate with difficulty scores, automatically down-weighting irrelevant features through the random feature sampling at each split.

Why Gradient Boosting Regressor was a bad choice?

1. **Gradient Boosting Regressor severely overfits**, showing near-perfect training performance but weak test results, indicating memorization rather than true learning.
2. **The model struggles with sparse**, text-based features, where boosting amplifies noise instead of capturing meaningful difficulty patterns.
3. In the absence of strong numerical difficulty indicators, Gradient Boosting fails to generalize and performs worse than simpler or averaging-based models.







Low R^2 Score Interpretation

The R^2 score of 0.004 appears concerning but requires context:

1. **High Variance in Target**: Difficulty is inherently subjective and multi-dimensional. Even human experts disagree on difficulty ratings, creating high inherent variance that no model can capture.
2. **MAE/RMSE More Relevant**: For practical deployment, absolute error metrics (MAE, RMSE) matter more than variance explained. Users care about prediction accuracy, not how much variance the model captures.
3. **Baseline Comparison**: The negative R^2 scores for Linear Regression (-0.02) show that simple models perform worse than always predicting the mean, while Random Forest's positive R^2 (albeit small) indicates it does capture some predictive signal.

---

4. Model Selection and Deployment

4.1 Final Model Selection

Based on comprehensive evaluation:

For Regression: Random Forest Regressor

- Lowest MAE (0.58) and RMSE (0.70) among all models
- Provides continuous difficulty scores for granular assessment
- Robust to outliers and captures non-linear patterns

```
MAE: 0.5825993320868225
RMSE: 0.7021369595226681
R2: 0.004547470323443936
```

For Classification: Logistic Regression

- Highest accuracy (51.12%) with balanced class predictions
- More reliable than Random Forest Classifier which showed extreme class bias
- Faster inference suitable for web deployment

```
Logistic Regression Accuracy: 0.5112
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>

Classification Report:
              precision    recall  f1-score   support

        easy       0.52      0.34      0.41       144
        hard       0.57      0.67      0.62       383
      medium       0.41      0.38      0.39       273

    accuracy                           0.51       800
   macro avg       0.50      0.46      0.47       800
weighted avg       0.50      0.51      0.50       800
```

4.2 Web Application Development

Technology Stack

Streamlit Framework: Selected for rapid web application development with Python-native integration:

- Simple syntax transforming Python scripts into interactive web apps
- Built-in UI components for file uploads, forms, and result display
- Direct integration with scikit-learn models without API overhead
- Real-time interactivity without requiring JavaScript knowledge

Application Architecture

The web application workflow:

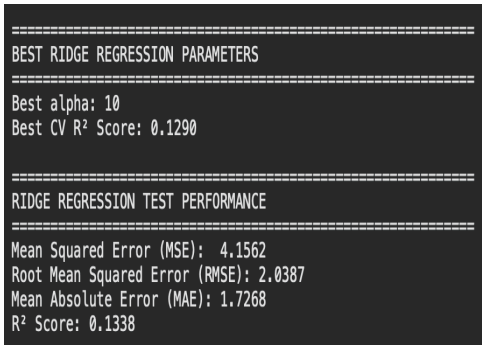
1. **User Input**: Upload problem features via CSV file or enter features manually
2. **Preprocessing**: Automatic feature scaling matching training pipeline
3. **Prediction**: Parallel execution of both regression and classification models
4. **Output Display**:
    - Continuous difficulty score (1-10 scale)
    - Categorical difficulty class (Easy/Medium/Hard)
    - Confidence metrics and feature importance visualization

Need for Web Interface

1. **Accessibility**: Non-technical users can leverage ML models without Python/coding knowledge
2. **Real-Time Predictions**: Instant feedback for problem setters designing new challenges

---

5. Ridge Regression Analysis



Ridge Regression with cross-validation achieved:

Despite better R^2 than Random Forest Regressor (0.1338 vs 0.0045), Ridge Regression's higher MAE and RMSE made Random Forest the preferred choice for deployment.

Handles multicollinearity in high-dimensional data

L2 regularization preventing coefficient explosion

---

6. Conclusions and Future Work

6.1 Key Findings

1. **Difficulty prediction is inherently challenging** due to its subjective nature, leading to modest classification accuracy (~48–51%) that varies with individual solver experience and background.
2. **Tree-based models such as Random Forest struggle with sparse, text-derived features**, as they rely on meaningful feature splits, which are difficult to learn from highly sparse representations.
3. **Producing accurate regression outputs remains difficult** because the feature set is largely text-based, while several critical numerical indicators of difficulty (e.g., time complexity constraints, optimal solution depth, or human effort metrics) are not explicitly available in the data.

6.2 Limitations

1. **Class Imbalance**: Uneven distribution of difficulty classes (especially in Random Forest Classifier results) impacted model performance
2. **Input Features Given**: The 1,800+ features extracted were nothing else that the textual information converted to its embeddings.
3. **Temporal Dynamics**: Problem difficulty perception evolves as community solving patterns change over time

6.3 Future Improvements

1. **Deep Learning Approaches**: Experiment with neural networks that can automatically learn feature representations from raw problem text
2. **Ensemble Methods**: Combine predictions from multiple models (stacking) to leverage strengths of both linear and tree-based approaches
3. **Class Balancing**: Apply SMOTE, class weights, or stratified sampling to address imbalance issues

4. **Natural Language Processing**: Incorporate problem description text analysis using transformers (BERT, GPT) to capture semantic difficulty cues
5. **User Feedback Loop**: Integrate user ratings in the web app to continuously refine model predictions with real-world feedback

---

Submitted By
Radhika Porwal
Chemical Engineering
23112078