AutoJudge - Problem Difficulty Prediction from Text

By Prachi Bhagat

Intro to problem :

Programming problem difficulty is one of the most important features in online judges and learning platforms. It helps users select problems fitting their level. So far, difficulty labels are usually assigned by human problem setters or inferred from user statistics manually, which may be arbitrary and time-consuming.
This project automatically predicts the difficulty of programming problems using only textual information in problem statements. That is, the system carries out two tasks:
Classification-Predicting the difficulty class(Easy, Medium, Hard)
Regression – Predicting a continuous difficulty score

The difficult part is that usually the class of problem is predicted by humans by their knowledge about the concept used in the problem , which was nearly impossible for machines to figure out just from the textual data . So this project just provides an estimate of the type of problem .

Objective :

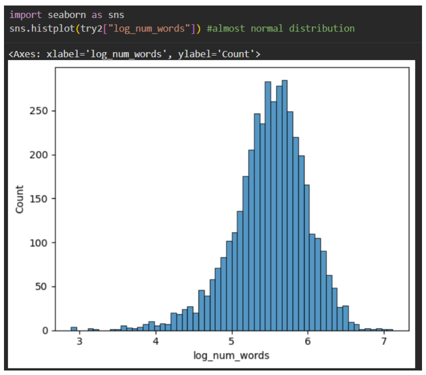
- To build a machine learning system that predicts problem difficulty using text alone
- To explore effective feature engineering techniques for small textual datasets
- To evaluate both classification and regression models
- To deploy the final system through a simple web-based interface

Dataset Description :

The dataset consists of approximately 4000 programming problems each with title , description , input/output description and problem url , this dataset was in the problem description -> here
The dataset is small , preventing use of deep learning models , problem scores are in a small band making it extremely difficult to predict actual problem scores , class distribution is imbalanced with more hard problems than easy and medium .

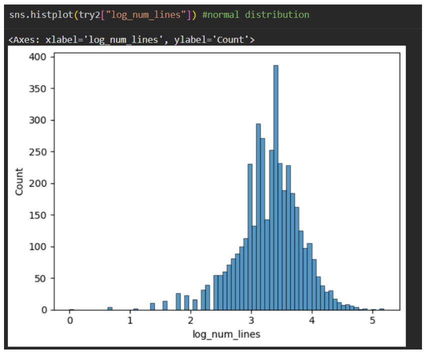
Data Preprocessing :

1. All the main text fields were combined (problem description , input and output).
2. Basic NLP steps were performed like converting to lowercase , removing extra spaces, removing dollar signs , stopword removal , lemmatization was performed .
3. TF-IDF was used to convert this processed text to vectors which can be fed to ML models.
4. Then various trials were performed to figure out which structural features actually mattered . Structural features like log number of words captures problem length , number of sentences approximates explanation complexity , some keywords showed improved results were also included , max numeric value was used as a proxy of constraints .



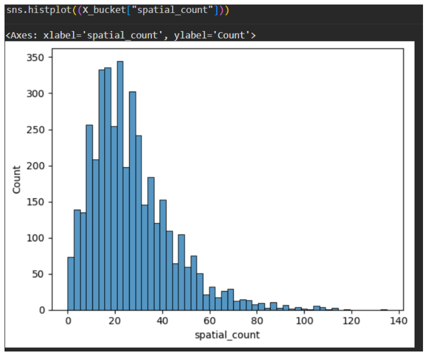
```
import seaborn as sns
sns.histplot(try2["log_num_words"]) #almost normal distribution
```

```
<Axes: xlabel='log_num_words', ylabel='Count'>
```



```
sns.histplot(try2["log_num_lines"]) #normal distribution
```

```
<Axes: xlabel='log_num_lines', ylabel='Count'>
```



```
sns.histplot((X_bucket["spatial_count"]))
```

```
<Axes: xlabel='spatial_count', ylabel='Count'>
```

Models Used :

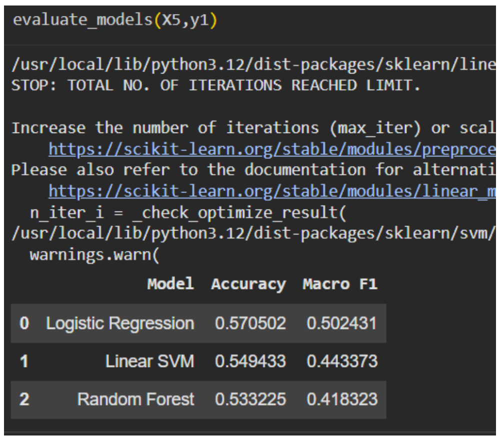
Classification Model

- **Logistic Regression** was used for difficulty classification
- It performed best among tested models such as SVM and Random Forest
- It handled sparse TF-IDF features efficiently and generalized well

Regression Model

- **Ridge Regression** was used for score prediction
- It was preferred over Linear regression due to better generalization .

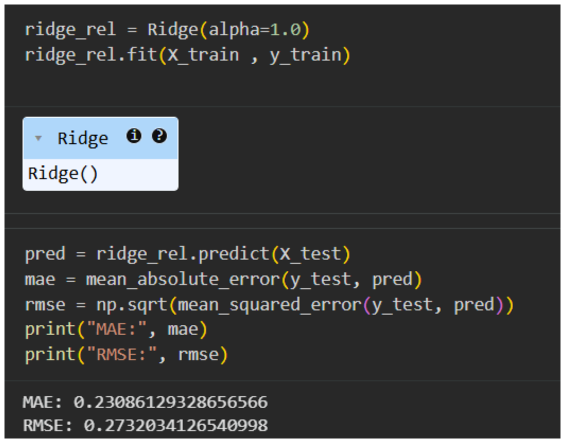
This is the final model that I used and its accuracy :



```
evaluate_models(X5,y1)

/usr/local/lib/python3.12/dist-packages/sklearn/line
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scal
    https://scikit-learn.org/stable/modules/preproce
Please also refer to the documentation for alternati
    https://scikit-learn.org/stable/modules/linear_m
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.12/dist-packages/sklearn/svm/
  warnings.warn(
                 Model  Accuracy  Macro F1
0  Logistic Regression  0.570502  0.502431
1           Linear SVM  0.549433  0.443373
2        Random Forest  0.533225  0.418323
```



```
ridge_rel = Ridge(alpha=1.0)
ridge_rel.fit(X_train , y_train)

Ridge
Ridge()

pred = ridge_rel.predict(X_test)
mae = mean_absolute_error(y_test, pred)
rmse = np.sqrt(mean_squared_error(y_test, pred))
print("MAE:", mae)
print("RMSE:", rmse)

MAE: 0.23086129328656566
RMSE: 0.2732034126540998
```

Hierarchical Regression Strategy

To address instability in score prediction, a **class-conditional regression approach** was used:

1. Predict difficulty class
2. Normalize scores within each class
3. Train Ridge Regression on normalized scores
4. Map predicted values back to class-specific score ranges

This approach improved score consistency and interpretability.

Classification Results

- Accuracy was used as the primary metric , best accuracy achieved was 57%

Regression Results

Regression performance was evaluated using:

- Mean Absolute Error (MAE) ~ 0.23
- Root Mean Squared Error (RMSE) ~0.27

Hierarchical regression showed improved stability compared to direct score prediction.

Web Interface

A web interface was developed using Streamlit.

Interface Description

The interface allows users to enter:

- Problem description
- Input description
- Output description

The system outputs:

- Predicted difficulty class
- Predicted difficulty score

Conclusion & Future scope

This project shows that it is possible to estimate programming problems' difficulties based solely on text information. The key is to extract both structural and numeric features from problems and combine them with TF-IDF features. Further optimizations could be made by utilizing large data , by inputting the topics that might be used in the problem and also by using pretrained language models.