7 Jan' 26

TAVISHEE AGRWAL
23119050 (PI – 3Y)

AUTO JUDGE

INTRODUCTION

Online programming platforms categorize problems by difficulty level and assign difficulty scores, which are typically determined through human judgment and user feedback. This project proposes **AutoJudge**, an automated system that predicts the difficulty class (Easy, Medium, Hard) and a numerical difficulty score for programming problems using only their textual descriptions.

The system leverages **TF-IDF embeddings** to capture semantic information from problem statements and combines them with **domain-specific engineered features** reflecting algorithmic complexity, constraints, and structural characteristics. Separate machine learning models are trained for classification and regression tasks.

DATASET DESCRIPTION

The dataset consists of programming problems collected from online competitive programming platforms. Each data sample contains the following fields:

- "title" Problem name
- "description" Detailed textual description of the problem
- "input_description" Textual Description of the input
- "output_description" Textual Description of the output
- "sample_io" Sample for how input/output will be entered
- "problem_class" Problem classification into easy/medium/hard
- "problem_score" Score allotted based on difficulty
- "url" Link to the problem

DATA PREPROCESSING

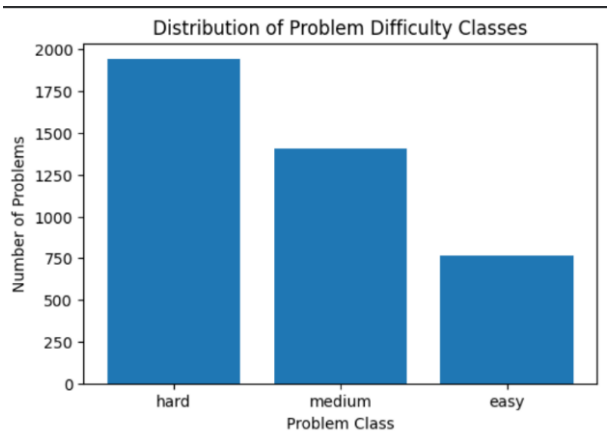
- Missing Values
  I checked for missing values but found none.

```
data.isnull().sum()
```

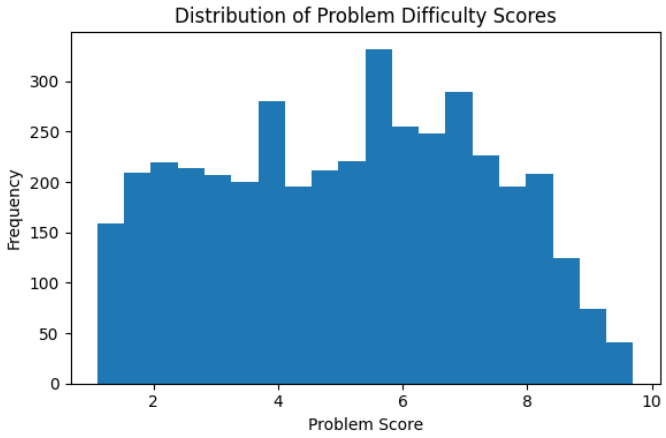
	0
title	0
description	0
input_description	0
output_description	0
sample_io	0
problem_class	0
problem_score	0
url	0

dtype: int64

- Class distribution
  It shows that data has class imbalance . There are more hard and medium problems than easy ones. Hence it might happen during prediction that our model is classifying hard problems accurately but it is misclassifies the easy problems a lot .We will handle it later and see if applying class imbalance is effective or not.



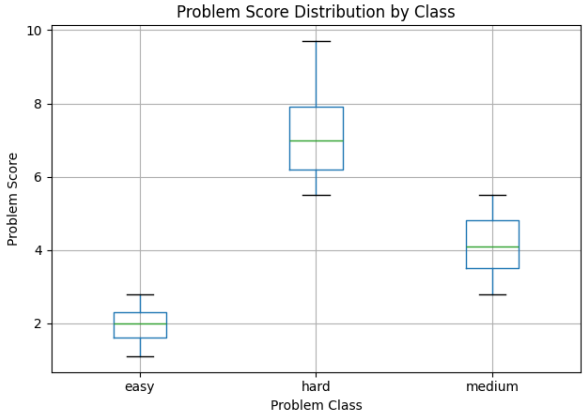
- Score Distribution



The scores range from approximately 1 to 10, with the majority of problems concentrated between scores 4 and 7. This indicates that most problems fall within the medium difficulty range, while very easy and very hard problems are comparatively fewer.

The distribution appears approximately symmetric and centered around a score of 5–6, suggesting a balanced dataset without extreme skewness. This makes it suitable for regression based models.

- Problem Score vs Problem Class



Easy problems exhibit low score values with a median around 2, while Medium problems occupy an intermediate range with a median near 4. Hard problems show significantly higher scores, with a median around 7. The clear separation between classes demonstrates consistency between categorical difficulty labels and numerical difficulty scores. Some overlap is observed between adjacent classes, which explains occasional misclassifications during model training.

FEATURE ENGINEERING

- I dropped all the unwanted columns like url and sample_io.
- Combing textual columns
  Columns like title , description and input and output description were combined into one new column 'full_text' so that model can see the complete context at once. This helps the model better understand the problem's complexity and difficulty.
- Then I dropped all the individual text columns.
- Then I engineered some domain specific features in a new dataset called heuristic_data:
  i. 'text_len'
  Harder programming problems usually need longer explanations because they include more constraints and edge cases. The text length helps the model capture this complexity directly.
  ii. 'math_density'
  This helps capture the mathematical complexity of the problem. Generally more complex problems often have more use of symbols.
  iii. 'has_kw'
  I created a list of keywords like graph , tree, dp , modulo and assigned a column for each. Now for a rows if the text contains the keyword it is assigned 1 else 0.
  Presence of the keywords can strongly be related to the difficulty of the problem.

Heuristic features extracted:

	text_len	math_density	has_graph	has_tree	has_dp	has_modulo
0	1672	24	1	0	0	0
1	1422	37	0	0	0	0
2	1334	37	0	0	0	0
3	1390	27	0	0	0	0
4	2261	26	0	1	0	0

- The combined textual column that I created cant be directly used , it first has to be converted to a vector embedding , I first used transformer based sentence embeddings but later on switched to TF-IDF embeddings as it gave much better results.

```
tfidf = TfidfVectorizer(max_features=10000, stop_words='english', ngram_range=(1, 2))
X_tfidf = tfidf.fit_transform(data['full_text'])
```

- Then I concatenated both the heuristic_data features and the newly formed vector embedddings of the textual data.This is going to be the final data that I'll be using for training.

MODELS AND EXPERIMENTAL SETUP

- Applied label encoder to the 'problem_class'.
- Applied train-test splitting.

Classification Model

The problem difficulty classification task was formulated as a multi-class classification problem with three classes: Easy, Medium, and Hard. After experimenting with multiple baseline classifiers, including Logistic Regression and Support Vector Machines, **XGBoost (Extreme Gradient Boosting)** I selected as the final classification model due to its superior performance.

XGBoost is an ensemble-based gradient boosting algorithm that builds decision trees sequentially and is capable of capturing complex non-linear relationships between features. It is particularly effective when working with high-dimensional feature spaces, such as those produced by TF-IDF representations combined with engineered features. Additionally, XGBoost provides inherent robustness to noise and feature interactions, making it well-suited for textual classification tasks.

I applied class weighting to handle imbalance but the results that I found with it were slightly below those that I obtained without it so I decided not to keep class weights for now.

```
clf_model = xgb.XGBClassifier(
    n_estimators=300,
    learning_rate=0.1,
    max_depth=6,
    use_label_encoder=False,
    eval_metric='mlogloss',
    random_state=42
)
```

Regression Model

The numerical difficulty score prediction was treated as a regression task. For this purpose, the **XGBoost Regressor** was employed. XGBoost Regression was chosen due to its strong performance in modeling non-linear relationships and its robustness when handling sparse, high-dimensional feature sets derived from textual data.

The use of gradient boosting with regularization helps reduce overfitting while capturing complex patterns between problem descriptions and their associated difficulty scores. This made XGBoost a suitable choice for predicting continuous difficulty values accurately.

```
reg_model = xgb.XGBRegressor(
    n_estimators=200,
    learning_rate=0.1,
    max_depth=6,
    random_state=42
)
```

RESULTS AND EVALUATION

Classification Result

```
--- Classification Results ---
              precision    recall  f1-score   support

        easy       0.57      0.35      0.43       136
        hard       0.62      0.76      0.68       425
      medium       0.40      0.34      0.37       262

    accuracy                           0.56       823
   macro avg       0.53      0.48      0.49       823
weighted avg       0.54      0.56      0.54       823

Accuracy: 0.56
```

I obtained an overall accuracy of 0.56 . The rest of the matrix is presented here.

Regression Result

```
--- Regression Results ---
Mean Absolute Error (MAE): 1.65
Root Mean Squared Error (RMSE): 2.01
```

I obtained MAE as 1.65 and RMSE as 2.01

Prediction errors were lower for mid-range difficulty scores, consistent with the score distribution observed during exploratory analysis.

WEB INTERFACE DESCRIPTION

The web interface is very simple and has been developed using streamlit.

The user will input the corresponding details into the following text boxes:

- Title box
- Description box
- Input Description box
- Output Description box

Then the user will click on the 'PREDICT' button and the system will output the predicted difficulty class and numerical difficulty score in real time.

SAMPLE PREDICTION

Title - Maximum Pair Sum

Description - You are given an array of integers of length n. Your task is to find the maximum possible sum of two **distinct** elements from the array.

The array may contain both positive and negative integers. You must choose two different indices $i \neq j$and compute the sum $a[i] + a[j]$. Output the maximum such sum.

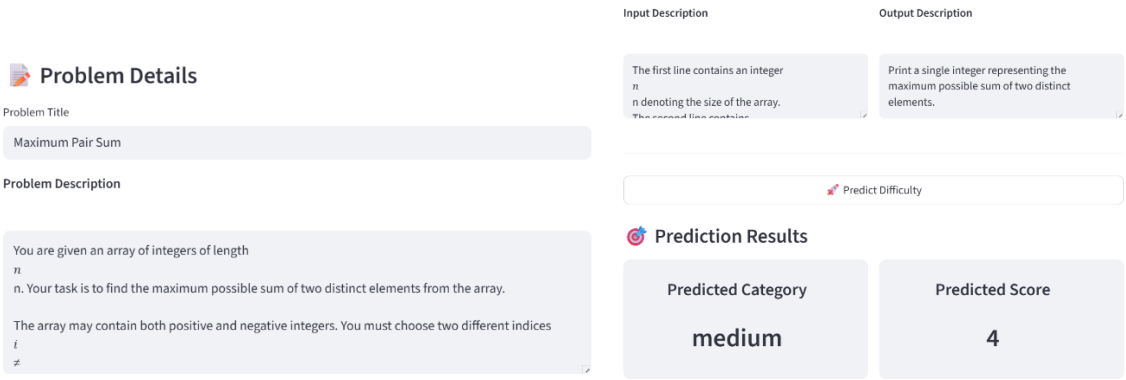
Input Description -

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

Constraints:

$$2 \leq n \leq 10^5$$
$$-10^9 \leq a[i] \leq 10^9$$

Output Description - Print a single integer representing the maximum possible sum of two distinct elements.

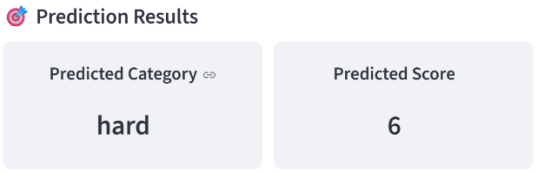


You can see the result is accurately as how it should be for this given problem .

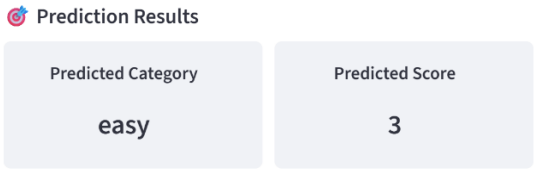
Medium with score 4 .

I predicted some other problems also like

- **Minimum Cost Path in Directed Graph**
  The result is :



- **Even or Odd**
  The result is :



CONCLUSION

This project presented an automated approach for predicting programming problem difficulty using textual descriptions. By combining TF-IDF features, engineered features, and XGBoost models, the system successfully predicted both difficulty class and numerical score. The results demonstrate the feasibility of automating difficulty assessment for programming problems.

THANK YOU