AutoJudge: Predicting Programming Problem Difficulty

NAME: Akhil kumar

ENROLLMENT NO: 24117009

BRANCH :Mechanical Engineering

Problem Statement

PROBLEM STATEMENT

AUTOJUDGE

The problem statement of this problem gives us a challenge to classify (easy, medium and hard) and rate programming problems similar to given on platform like leetcode, codeforces, codechef etc and make a simply web interface for the program. To complete this task Machine learning based algorithms are to be used. While use of deep learning is not allowed. So algorithms like random forest, boosting algorithm etc will be used.

DATASET USED

- The dataset used for the training was taken from Hugging Face . https://huggingface.co/datasets/open-r1/codeforces
- It can also be download using the datasets library directly in python using the given below code.

```python
from datasets import load_dataset

# Login using e.g. `huggingface-cli login` to access this dataset
ds = load_dataset("open-r1/codeforces", "default")
```

- The dataset **contains three subsets: `default`, `verifiable`, and `verifiable-prompts`**.
- **This project uses the **default** subset** with about 10k data points.
- **Original dataset had separate train and test splits for this project, both splits were combined and a custom 80/20 train-test split was created**.

	Column Heading	Column Heading
Row Heading	Text	123.45
Row Heading	Text	123.45
Row Heading	Text	123.45

- *As the dataset did not originally contain a class label, a categorical class feature was derived based on the Codeforces problem ratings. The problems were classified according to the following rating thresholds:*

  Easy: rating<1200

  Medium:1200<rating<1900

  Hard: rating>1900

- After forming the classes, we saw that the distribution of data is as follows

  Hard 43%

  Medium 32%

  Easy 24%

DATA PREPROCESSING

The raw dataset contains several metadata fields that are not directly useful for difficulty prediction.

ORIGINAL FEATURES

id, aliases, contest_id, contest_name, contest_type, contest_start, contest_start_year, index, time_limit, memory_limit, title, description, input_format, output_format, interaction_format, note, examples, editorial, rating, tags, testset_size, official_tests, official_tests_complete, input_mode, generated_checker, executable, generated_tests

FEATURES USED

time_limit, memory_limit, title, description, input_format, output_format, rating, tags

All of the text fields are then concatenated and classes are formed using ratings as shown

page 2.

TEXT CLEANING

- Removal of special characters and underscores
- Conversion to lowercase
- Tokenization using the NLTK word tokenizer
- Removal of stopwords, while retaining important words for prediction such as *not*, *if*, *find*, and *graph*, as these influence problem difficulty
- Lemmatization using **WordNet** Lemmatizer to reduce words to their base forms

FEATURE ENGINEERING

Construction of basic numerical feature from text data which are word count ,token count, Complexity words, and algorithm count.

While the meaning word count and token count are clear from the name the complexity count and algorithm count counts words like given below .

Complexity word- limit, efficient, optimize etc.

Algorithm count – dp, graph, binary, tree, bit etc.

As we can not find all such words only some of them are taken in care while addition of more such words may improve model performance.

Each problem contains multiple tags indicating relevant algorithms or concepts. The data contains more than 30 unique tags so a **MultiLabelBinarizer** was applied to convert tags into a binary vector representation. This allows models to learn relationships between problem topics and difficulty levels.

All the numerical features were scaled using StandardScaler. (Word count ,Token count Algorithm count ,Complexity term count , Time limit, Memory limit)

TEXT VECTORIZATION

To convert text into numerical form, **TF-IDF (Term Frequency–Inverse Document Frequency)** vectorization was applied with the following configuration:

- N-gram range: 1 to 4
- Maximum features: 3000
- Sublinear term frequency scaling
- English stopword removal

Final Feature Matrix Construction

The final feature representation was created by horizontally stacking:

1. TF-IDF sparse vectors
2. Scaled numerical features
3. Binary tag vectors

And finally, the train_test_split was used for a 80/20 split.

CLASSIFICATION MODELS

Predicting the difficulty class of problems.

Logistic Regression

- Multinomial logistic regression, Uses L2 regularization
- Accuracy obtained: 67.55%
- Confusion matrix:

Actual \ Predicted	Easy	Hard	Medium
Easy	341	16	123
Hard	25	671	150
Medium	131	189	308

Linear Support Vector Classifier (Linear SVC)

- Effective for large sparse feature spaces
- Accuracy obtained: 64.12%
- Confusion matrix:

Actual \ Predicted	Easy	Hard	Medium
Easy	345	27	108
Hard	34	630	182
Medium	162	188	278

Random Forest Classifier

- Ensemble of decision trees
- Accuracy obtained: 64.69%
- Confusion matrix:

Actual \ Predicted	Easy	Hard	Medium
Easy	301	29	150
Hard	24	690	132
Medium	135	220	273

XGBoost Classifier

- Gradient boosting framework
- Accuracy obtained: 67.15%
- confusion matrix:

Actual \ Predicted	Easy	Hard	Medium
Easy	340	13	127
Hard	22	654	170
Medium	144	166	318

REGRESSION MODELS

Predicting the numerical rating of problems.

Random Forest Regressor

- Captures non-linear relationships
- R2 score: 0.5227
- Mean absolute error: 385.78

XGBoost Regressor

- Gradient boosting with decision trees
- R2 score: 0.5819
- Mean absolute error: 355.26

Results and Evaluation

Multiple models were tried in classification Xgboost and logistic regression performed very well and were close in performance. In regression model Xgboost performed good but still on 0.58 R2 score was achieved. For future works and improvement of the tool Deep Learning should be tried.

DISCUSSION

- Use of tags, time limit and memory limit improved the model significantly from almost 60% accuracy .
- While class imbalance remains a challenge and also the use of artificially made class may have cause misclassification on the border ratings .
- For regression tasks, the available data is often insufficient to achieve significantly accurate results, making it difficult for classical machine learning methods to effectively differentiate between neighbouring classes. Deep learning approaches, however, may provide better performance in such scenarios.

Web Interface

Streamlit Application Design

The web interface allows users to:

- Enter problem title and description
- Specify input/output formats
- Select relevant tags
- Provide time and memory limits

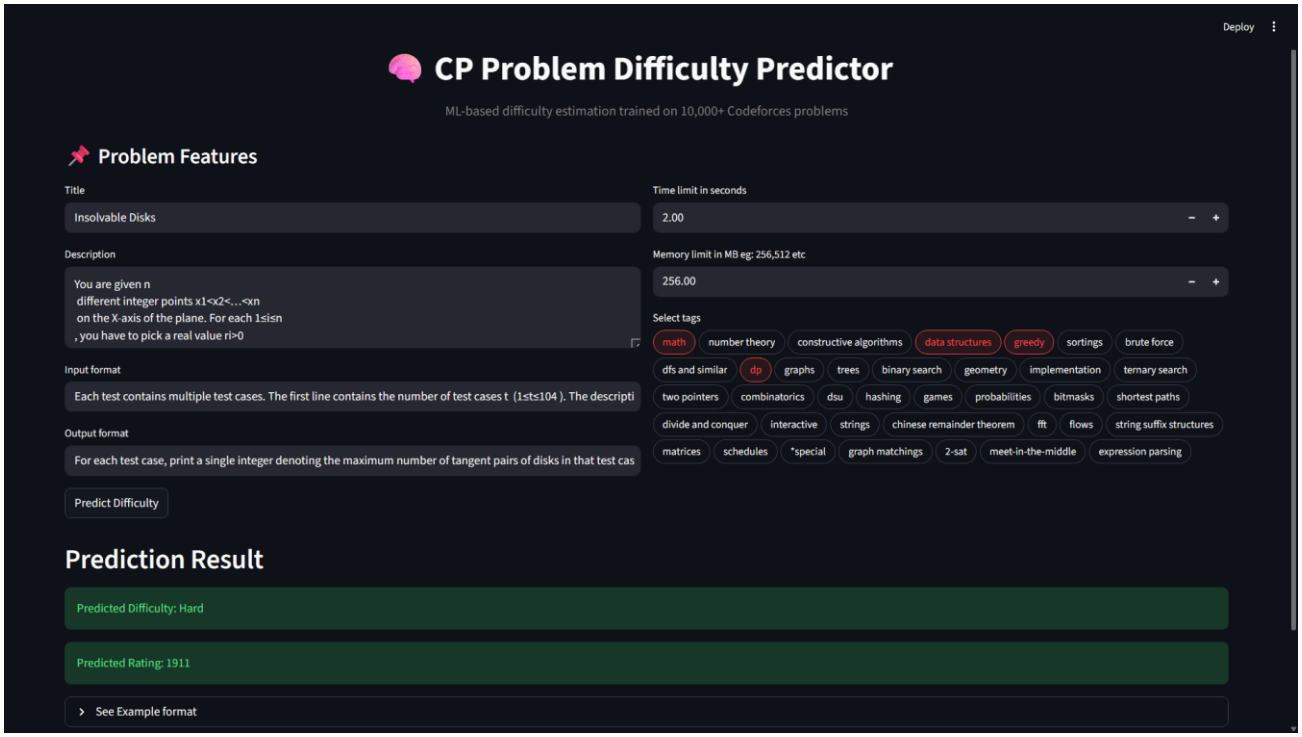
A separate `preprocessing.py` module is implemented to ensure that input data provided through the web interface is preprocessed in exactly the same manner as the training data. This module exposes a preprocessing function that is imported and used during inference to maintain consistency between training and prediction pipelines.

To replicate the original training environment, the preprocessing function relies on several serialized objects stored as pickle files. These include the **MultiLabelBinarizer**, which ensures that problem tags are encoded in the same fixed order as during training, preventing feature misalignment. Additionally, other preprocessing artifacts such as the **TF-IDF vectorizer**, **scaler**, and feature configuration objects are loaded to apply identical transformations to incoming data.

All required pickle files are loaded **only once** during application startup and stored in cache. This design significantly improves the performance of the web application by avoiding

repeated disk access and redundant object initialization during each prediction request. As a result, the system ensures both **accuracy and efficiency** during real-time inference.

Sample Prediction



The question was used for this is a recent question of a contest and is rated 1900.

Link to the original problem : https://codeforces.com/contest/2180/problem/D