Classifying Tags for Codeforces Problems

Manas Mulpuri
mmulpuri@ucsd.edu

1 Introduction

Competitive programming is a famous sport in the world of algorithms, and recent advancements in large language models (LLMs) have demonstrated potential in solving complex programming problems. However, I got the question: **Can we use classifiers to classify problems by their topics based their problem statements?**

Codeforces problems are tagged with topics such as *"graphs", "dp" (dynamic programming),* and *"greedy"*, which indicate the core concepts or algorithms necessary to solve them. Automating the classification of these tags can offer clues for anyone unable to solve these problems and it can also filter problems and send them to downstream specialized ML models which are trained for solving problems related to a specific topic.

In this project, I explored whether problem tags could be accurately predicted from problem statements using NLP techniques. Here is the summary of tasks:

- Web-scraped Codeforces and processed problems for the dataset: DONE.
- Finetuned a BERT model on the collected dataset to examine its performance: DONE.
- Finetuned AlBERT and RoBERTa models on the dataset to examine if they outperform the baseline: DONE.

My main goal was to see if a classifier can analyze the textual content of problem statements to detect patterns and phrases which are related to a specific tag, where human programmers like us, often rely on our intuition and our experience to identify these topics.

2 Related work

A lot of research is being conducted on leveraging machine learning to solve competitive programming problems. Notably, deep learning techniques have been explored to automate and improve processes such as problem tagging, code generation, and solution analysis. (Li et al., 2022) demonstrated how advanced language models could generate code and even compete on platforms like Codeforces.

Transformer-based models like BERT and its variants, such as ALBERT and RoBERTa, have shown great promise across various natural language processing tasks. The study by (Azizah et al., 2023) compared these models, highlighting their strengths and weaknesses for tasks requiring contextual understanding and fine-grained classification. The insights from such studies suggest that these models could be highly effective for understanding and classifying competitive programming problem descriptions.

BERT, in particular, has been successfully applied to multilabel classification tasks. In (Schonlau et al., 2023), researchers demonstrated how BERT's bidirectional contextual embeddings could handle complex and nuanced text classification, making it an ideal candidate for the task of tagging Codeforces problems with multiple labels.

However, not all prior research has focused on Transformer models. Some studies have explored alternative deep learning architectures for similar tasks. For instance, (Lokat et al., 2023) utilized LSTMs, GRUs, and MLPs for tagging coding problems, demonstrating the effectiveness of these recurrent architectures in capturing sequential dependencies in problem descriptions. Additionally, (TS et al., 2023) experimented with boosting models for classification, achieving competitive performance through model stacking techniques.

These diverse approaches highlight the growing interest in using machine learning for competitive programming tasks. Building on these works, this project aims to explore BERT and its siblings for multilabel classification of Codeforces problems.

3 Dataset

As my objective was to predict tags from problem statements, using Jina and Codeforces API, I scraped through all of the Codeforces problems and got the problem statements, tags, and ratings for each problem.

Here is an example problem which I solved a while ago. 432A

```
The Saratov State University
   Olympiad Programmers Training
   Center (SSU OPTC) has n
   students. For each student you
    know the number of times he/
   she has participated in the
   ACM ICPC world programming
   championship. According to the
    ACM ICPC rules, each person
   can participate in the world
   championship at most 5 times.
The head of the SSU OPTC is
   recently gathering teams to
   participate in the world
   championship. Each team must
   consist of exactly three
   people, at that, any person
   cannot be a member of two or
   more teams. What maximum
   number of teams can the head
   make if he wants each team to
   participate in the world
   championship with the same
   members at least k times?
```

Here is a snippet of my solution

```
while(test--){
   int n,k;
   cin >> n >> k;
   int arr[n];
   for(int i =0;i<n;i++)
       cin >>arr[i];
   sort(arr,arr+n);
   int teams = 0;
   for(int i = 0;i<n;i++){
       if ((5-arr[i]) >= k)
           teams++;
       else break;
   }//endfor
   cout << teams/3 << endl;
}//endwhile
```

This solution required sorting the input array and greedily counting the number of teams. Therefore, the tags Codeforces has assigned to this problem are *["greedy", "implementation", "sortings"]*.

math	2915
greedy	2909
implementation	2729
dp	2135
data structures	1751
constructive algorithms	1744
brute force	1700
binary search	1086
sortings	1072
graphs	1049
dfs and similar	930
trees	818
number theory	753
strings	718
combinatorics	677
bitmasks	572
two pointers	549
geometry	380
dsu	359
*special	325
divide and conquer	295
shortest paths	264
games	236
probabilities	233
interactive	219
hashing	215
flows	146
matrices	122
fft	96
string suffix structures	92
graph matchings	90
ternary search	58
meet-in-the-middle	49
2-sat	35
expression parsing	32
chinese remainder theorem	18
schedules	10

Above is the distribution of all the tags for Codeforces problems. Only the top 10 occurring tags were considered in the dataset, which are *[math, gray, implementation, dp, data structures, construction algorithms, brute force, binary search,*

sortings, graphs].

I also saved the ratings for each problem as later we could build a rating predictor based on the problem statement.

Dataset Details:
Size: 9,076 rows (problems).
Tags: [math, gray, implementation, dp, data structures, construction algorithms, brute force, binary search, sortings, graphs]
Splits: The dataset was split into 80% for training, 10% for validation, and 10% for testing.
Average text length (in characters): 2067.18
Average number of words per text: 358.20
Average word length: 4.74

The challenges I faced when working with Codeforces problems were:

- The Jina reader occasionally extracted extra information from webpages, such as contest descriptions or unrelated text, with the actual problem statement, introducing noise into the dataset.
- Sometimes Codeforces uses stories to explain the problem statement causing them to be long sometimes. This poses a challenge when tokenizing and processing the data, especially for transformer-based models with input length limitations.

You can also find the dataset I created on HuggingFace: CodeforcesProblems.

3.1 Data preprocessing

To ensure the dataset was clean and usable, I:

- Retained only rows with problem statements and tags within the top 10 most frequent categories, aligning with the scope of the task.
- Cleaned out irrelevant data extracted by the Jina reader, such as URLs and unrelated webpage elements.

These steps were required to eliminate noise and ensure the dataset centered exclusively on problem statements and the most common tags.

4 Baselines

For this project, the baseline model I used a multilabel classification model based on BERT, fine-tuned for the classifying Codeforces problems. I used a pretrained BERT model and fine-tuned a classification head, where the output layer is adjusted for multilabel classification. The dataset was split into 80% training, 10% validation, and 10% test sets. I used the validation set to tune hyperparameters and monitor performance during training. The hyper parameters I used were

- Learning Rate: 2e-5
- Batch Size: 8
- Number of Epochs: 3
- Weight Decay: 0.01

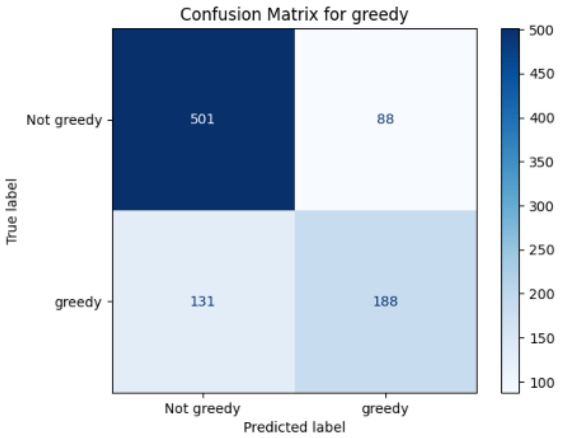

Figure 1: Using BERT

The baseline on the test set demonstrated reasonable overall accuracy: 0.8292 and precision: 0.6972, but the recall: 0.3661 and F1 score: 0.4801 were low.

5 Your approach

5.1 Conceptual Approach

BERT has set the benchmark for several NLP applications, including text classification, question answering, and more. However, newer models like ALBERT and RoBERTa have been introduced to address certain limitations in BERT and enhance performance in specific use cases.

My approach was to leverage these advancements to build a multilabel classifier using ALBERT, and RoBERTa. I wanted to explore how these models perform in a multilabel classification setting, comparing their accuracy.

5.2 Working Implementation

Yes, I was able to complete a working implementation for all of the models. You can find the following at:

- Scraping script: *scraper.py*
- BERT model: *Project_Bert.ipynb*
- AlBERT model: *Project_Albert.ipynb*
- RoBERTa model: *Project_Roberta.ipynb*
- Evaluation script: *Project_Evaluator.ipynb*

5.3 Compute

I used Colab(T4 GPU) and Kaggle(GPU T4x2) notebooks for this project, where I used the GPUs provided to finetune my models.

5.4 Runtime

To scrape problems from Codeforces website it took around 12 hours.
Each model took around 35-40 minutes to train.

5.5 Results

Metric	BERT	RoBERTa	AlBERT
Accuracy	0.8292	0.8311	0.8018
Recall	0.3661	0.3778	0.1876
F1 Score	0.4801	0.4907	0.2897
Precision	0.6972	0.6998	0.6349

Table 1: Comparison of Overall Metrics for BERT, RoBERTa, and AlBERT

The performance of the BERT, RoBERTa, and ALBERT models shows different strengths and weaknesses, with RoBERTa emerging as the best performer overall. RoBERTa achieves the highest F1 score and recall performing slightly better than BERT. On the other hand, AlBERT struggles significantly, with a much lower F1 score, recall, and precision, indicating that it is unable to generalize well across most labels.

RoBERTa's improved performance could be because of its robust pretraining methodology. Unlike BERT, RoBERTa uses a larger batch size, more training data which helps it learn better representations for textual features.

The reason AlBERT's performance couldn't match BERT could be because of its small model size. This could be partly the reason it didn't perform as well for complex tasks like multilabel classification in this case.

6 Error analysis

I will be using the links of the problems instead of copying the whole problem statements here.

Here's an example where the baseline(BERT) failed: 1487A
The BERT model misclassified the tags for this problem by predicting "math" and "implementation" instead of the correct tags, "implementation" and "sortings." This could be as the problem involves reasoning about the progression of hero levels, which might have been interpreted by the model as a mathematical process, leading to the inclusion of "math."

Here's an example where the RoBERTa failed: 1088F
Here RoBERTa misclassified the tags for this problem, predicting "greedy" and "data structures" instead of the correct "data structures" tag alone. This error likely arises because the problem involves constructing an optimal tree, which might look like problems solved with greedy techniques. RoBERTa may have associated phrases like "minimizing the weight" or "choosing neighbors" with greedy strategies. This indicates a potential over-reliance on surface-level cues rather than fully capturing the core structural and algorithmic requirements of the problem.

Here's an example where AlBERT failed: 1990C.
Here AlBERT predicted only the "greedy" tag instead of the correct tags "math," "greedy," and "brute force". AlBERT might also rely heavily on dominant features, such as the sequential updates in the problem, which align with "greedy" strategies, overlooking secondary features indicating "math" or "brute force."

From these examples, I observed that there is often an overlap between tags. For instance, a problem with a greedy solution can also be interpreted as having a mathematical approach, leading to ambiguity for the model. Additionally, the narrative or context provided in the problem statement can mislead the model, as the descriptive text may contain keywords or phrases that incorrectly influence the tag prediction. These factors highlight the challenges of accurately capturing the problem's essence, as models can sometimes focus on unnecessary details rather than the underlying algorithmic requirements.

7 Conclusion

My key takeaway from this project was that BERT and its variants performed effectively in classifying tags for Codeforces problems. However, the most challenging aspect of the project was scraping the Codeforces website for problem statements. Since there were no APIs available to provide the problem statements and the site blocked access via Python scripts, I had to use Jina, which internally navigates each Codeforces link, extracts the HTML content, and leverages LLMs to retrieve the problem statements.

Looking ahead, I have a few ideas to enhance this project. One approach would be to use a text summarizer to condense the problem statements before passing them into a transformer. Codeforces problems often contain a lot of unnecessary information, which increases the length of the input and poses challenges for models like BERT, which have a token limit of 512. Another idea is to experiment with larger models like Llama-3.2, which support longer inputs, thereby eliminating the need to truncate problem statements. Lastly, it would be interesting to explore predicting the problem ratings in addition to classifying tags. Since the dataset includes ratings for each problem, using transformers with a regression head could provide insights into how well these models perform in predicting problem difficulty.

8 Acknowledgements

I used ChatGPT to proof-read and paraphrase, which suggested minor changes.

References

Azizah, S. F. N., Cahyono, H. D., Sihwi, S. W., and Widiarto, W. (2023). Performance analysis of transformer based models (bert, albert and roberta) in fake news detection.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., Hubert, T., Choy, P., de Masson d'Autume, C., Babuschkin, I., Chen, X., Huang, P.-S., Welbl, J., Gowal, S., Cherepanov, A., Molloy, J., Mankowitz, D. J., Sutherland Robson, E., Kohli, P., de Freitas, N., Kavukcuoglu, K., and Vinyals, O. (2022). Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Lokat, T., Prajapati, D., and Labde, S. (2023). Tag prediction of competitive programming problems using deep learning techniques.

Schonlau, M., Weiß, J., and Marquardt, J. (2023). Multi-label classification of open-ended questions with bert.

TS, S. K., Pandian, S. L., and Shunmugapriya, P. (2023). Stacking of hyperparameter tuned models for tagging coding problems.