BranPy: Classification of mathematical problems in tree-structured domains

Nikolay Georgiev

Abstract

The following paper presents the `branpy` module that provides several methods for classification of mathematical problem statements in tree-structured domains with different depths chosen by the user. Here the methods implemented are described together with an estimation of their accuracy. Problem classification is an important task for automatically evaluating the performance of novel reasoning models in different types of problems and identifying their weaknesses.

1 Introduction

Classification of mathematical problems is an example of a classical text classification task. One specific feature of the classes here is that they are tree-structured. For example the domain *Matrices* is a branch of the domain *Linear algebra*, which is a branch of *Algebra*. Then this class can be structured as $Mathematics -> Algebra -> Linear\ Algebra -> Matrices$. In this paper we evaluate the performance of the `branpy` module on Omni-MATH Benchmark by Gao et al. [1] that consists of 4428 competition-level math problems where each problem is provided with a tree-structured domain in this format. Where more than one domain is provided, only the first (main) one is taken into consideration. This task is important where large reasoning model are trained on other big datasets that lack such feature and an evaluation of performance in different domains is needed. The `branpy` module is accessible at `https://github.com/georcons/branpy`.

2 Task Formulation

Several tree depths of classification are implemented in the `branpy` module:

Main (M): Classify a problem statement into one of the following 5 domains: Algebra, Geometry, Number Theory, Combinatorics or Others.

Reduced (R): Classify a problem statement into one of 24 domains, reduced from the complete list of domains in the Omni-MATH dataset. See Appendix A.

Complete (C): Classify a problem statement into one of the 139 domain labels that appear in the Omni-MATH dataset. See Appendix B for a complete list.

3 Methodology

The module provides two main methods for classification:

Method 1 (*Offline*). This method uses a Naive Bayes Classifier trained on 80% of the Omni-MATH dataset. Statements are preprocessed by removing a set of certain symbols and applying lowercase. A Tfid vectorizer with stop words is used [2].

Method 2 (*OpenAI*). As shown by Brown et al. LLMs are few-shot learners [3]. To classify problems into domain a model (that is either *gpt-4o-mini* or *gpt-4o*) is provided with a list of possible domain labels and with several classification examples. Then it is asked to classify another problem.

To enhance the performance of both approaches another method that we call *tree augmentation* is used. That is when classifying the main or the reduced branch, the model is trained (in the case of Naive Bayes) or prompted (in the case of LLM) with a bigger tree (reduced or complete) and makes a prediction for this bigger tree. Then the result is mapped to the corresponding shallower branch.

4 Results

In this section the results of the different methods are presented. A human performance baseline of 70% for the main branch and 43.3% for the reduced one was established. To evaluate the Naive Bayes approach the Omni-MATH dataset is randomly split into a training and test set with ratio 9:1 several times. Each model is trained and the accuracy evaluated and then the results are averaged. Each column is the tree on which the model was trained and the rows present the results on different tasks.

TASK	**Main Train**	**Reduced Train**	**Complete Train**
Main	77.34%	78.84%	78.70%
Reduced	-	56%	55.1%
Complete	-	-	38.33%

In the `branpy` module for offline classification of both main and reduced branch the reduced branch is used for training. For complete branch classification the model is trained on the complete tree. There in this method tree augmentation is applied only for classification of the main branch.

To evaluate the performance of the LLM approach 10% of the samples in the Omni-MATH dataset are randomly selected. In the table below the results for *GPT-4o-mini* are presented.

TASK	**Main Train**	**Reduced Train**	**Complete Train**
Main	74.94%	82.84%	83.97%
Reduced	-	42.21%	53.5%
Complete	-	-	22.8%

In the module for all tree depths tree augmentation to complete depth is used to maximize results.

5 Conclusions

Tree Augmentation may improve performance of Naive Bayes Classification on the Main Depth Classification task but always improves the performance of the LLM approach. Given the low human level baseline one may hypothesize the high accuracy of the Naive Bayes approach on the Reduced and Complete Depth tasks is a result of overfitting and further research is needed to resolve that.

References

[1] B. Gao et al. "Omni-MATH: A Universal Olympiad Level Mathematic Benchmark For Large Language Models". In: (2024). arXiv: 2410.07985.

[2] Joel Nothman, Hanmin Qin, and Roman Yurchak. "Stop Word Lists in Free Open-source Software Packages". In: Proceedings of Workshop for NLP Open Source Software (NLP-OSS) (2018).

[3] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: (2020). DOI: https://doi.org/10.48550/arXiv.2005.14165.

Appendix A.

Here is a list of all 24 *Reduced* Classes:

```
1. "Mathematics -> Combinatorics -> Probability",
2. "Mathematics -> Combinatorics -> Statistics",
3. "Mathematics -> Combinatorics -> Graph Theory",
4. "Mathematics -> Combinatorics -> Logic",
5. "Mathematics -> Combinatorics -> Algorithms",
6. "Mathematics -> Combinatorics -> Others",
7. "Mathematics -> Algebra -> Math Word Problems",
8. "Mathematics -> Algebra -> Expressions",
9. "Mathematics -> Algebra -> Polynomials",
10. "Mathematics -> Algebra -> Sequences and Series",
11. "Mathematics -> Algebra -> Complex Numbers",
12. "Mathematics -> Algebra -> Linear Algebra",
13. "Mathematics -> Algebra -> Differential Calculus",
14. "Mathematics -> Algebra -> Integral Calculus",
15. "Mathematics -> Algebra -> Other Calculus",
16. "Mathematics -> Algebra -> Others",
17. "Mathematics -> Number Theory -> Prime Numbers",
18. "Mathematics -> Number Theory -> Factorization",
19. "Mathematics -> Number Theory -> Congruences",
20. "Mathematics -> Number Theory -> Others",
21. "Mathematics -> Geometry -> Plane Geometry",
22. "Mathematics -> Geometry -> Stereometry",
23. "Mathematics -> Geometry -> Others",
24. "Mathematics -> Others"
```

Appendix A.1

For a complete list of all 139 complete classes check
`https://github.com/georcons/branpy/blob/main/paper/classes.txt`.

Appendix B

For the few-shot prompting of the GPT model the system prompts looks like this:

```
I am a teacher, and have some high-level olympiad math
problems. I want to categorize the domain of these math
problems. Here is a list of all possible 139 domains:
{List of The Domains from Appendix A.1}
Here are a few references to the domain.
{List of Several Example Problems with Domains}
```

Then the use prompt is as follows:

```
Give me the domain of the following problem: {Statement}.
Prompt only the domain without any other words.
```