Proc. 14th Int. Conf. on Global Research and Education, Inter-Academia 2015
JJAP Conf. Proc. **4** (2016) 011609
©2016 The Japan Society of Applied Physics

Framework for Sense Disambiguation of Mathematical Expressions

Takayuki Watabe[1,2]*, Yoshinori Miyazaki[3], and Shosaku Tanaka[4]

[1]Graduate School of Science and Technology, Shizuoka University, Shizuoka 432-8011, Japan
[2]Research Fellow of Japan Society for the Promotion of Science
[3]Graduate School of Integrated Science and Technology, Shizuoka University, Shizuoka 432-8011, Japan
[4]College of Letters, Ritsumeikan University, Kyoto 603-8346, Japan

E-mail: dgs13012@s.inf.shizuoka.ac.jp

(Received September 28, 2015)

Mathematical expressions are indispensable for describing mathematical concepts or models. Although these expressions are formal representations, they contain ambiguity, i.e., a single expression could be interpreted as having multiple meanings. This feature prevents the flexible use of mathematical expressions in computation. In this paper, we focus on symbol-level ambiguity and consider the problem of labeling semantic information to each symbol as a classification problem. Then we propose a framework for solving the problem with supervised learning.

1. Introduction

Mathematical expressions are indispensable for describing mathematical concepts or models; they appear in various documents including textbooks, academic papers, and electronic documents. Although mathematical expressions seem to be formal representations, a single expression can be interpreted in multiple ways. For example, consider the use of an ambiguous symbol i in different mathematical expressions. The symbol i in $a + bi$ is likely to represent an imaginary unit, while the i in $\sum x_i$ is interpreted as an index. This symbol-level ambiguity can cause difficulties when determining the meaning of mathematical expressions, preventing the flexible use of mathematical expressions in computational applications. In particular, each symbol in mathematical expressions in applied physics has more various meanings than in mathematics, because applied physics covers a wide range of fields. This implies that the problem of ambiguity is even more serious in processing mathematical expressions in applied physics.

The data described by mathematical expressions in computation can be divided into two categories. The first is the data of the location of symbols (locational data) and the second is that of operators and operands (operational data). An example of locational data is "locating S as a superscript of 2," and an example of operational data is "applying exponentiation to a base 2 and exponent S." Locational data does not correspond one-to-one with an operational data. For example, the aforementioned locational data of "locating S as superscript of 2" can be interpreted as "a power set of S." On the other hand, the operational data of "applying exponentiation to a base 2 and exponent S" can be also represented as "locating 2, ^, and S, horizontally." We refer to this correspondence of a single locational data point to multiple operational data points as the ambiguity of mathematical expressions.

The uses of mathematical expressions in computation can be roughly divided into two categories. The first is their use in describing mathematical expressions in electronic documents such as web pages and e-books; this usage requires locational data. The second use is in applications that transform mathematical expressions and create graphs; these applications need to attach meanings to mathematical expressions, such as the distinction between operators and their operands. The ambiguity of mathematical expressions makes it challenging to combine both usages. This study aims to resolve this problem.

To this end, symbol-level ambiguity should be eliminated. For example, the ambiguity of the mathematical expression $|A|$, which is interpreted as an absolute value of A, cardinality of A, or determinant of A, is caused by the symbol-level ambiguity of '$|$.' We consider the problem of identifying appropriate semantic labels to each ambiguous symbol as a classification problem. Therefore we build classifiers for each ambiguous symbol using an algorithm of supervised learning.

As features for classification, we use the frequency of symbols that appear in the same mathematical expressions as a target ambiguous symbol, because, for instance, it is highly possible that the label of "|" in $|A|$ is cardinality if $|A|$ and $\cap$ appear in the same mathematical expression. Moreover, we define *n*-grams of symbols on mathematical expressions and use them as additional features for classification.

Previously, a disambiguation method for mathematical expressions was proposed by Nghiem et al.[1], which proposed heuristics for alignment between locational data and operational data.

Labeled symbols would help transform locational data into operational data. Also, an advanced search tool for mathematical expressions using semantic information could be developed.

2. MathML

In this study, we adopt MathML[2] as the data format for mathematical expressions. MathML is recommended by W3C and is commonly used. MathML has two types of tag sets, which are called Presentation Markup and Content Markup.

Locational data and operational data could be represented by Presentation Markup and Content Markup, respectively. Fig. 1 shows examples from the expression x^2 (x squared).

Presentation Markup	Content Markup
`<math>`	`<math>`
`   <msup>`	`   <apply>`
`      <mi>x</mi>`	`      <power/>`
`      <mn>2</mn>`	`      <ci>x</ci>`
`   </msup>`	`      <cn>2</cn>`
`</math>`	`   </apply>`
	`</math>`

Fig. 1. Examples of MathML data

As shown in Fig.1, the msup element in Presentation Markup represents "locating its second child as superscript of its first child," which is locational information. The apply element in Content Markup plays the role of "applying an operation as its first child to operands as its following children," which is operational information.

However, both Presentation Markup and Content Markup have further notations for representing additional information other than that which is locational and operational. The rest of this chapter details the focused notations of MathML used in this research.

2.1 Presentation Markup

In Presentation Markup, we focus on elements for specifying locations of symbols, i.e., msub (subscript), msup (superscript), msubsup (subscript and superscript), munder (underscript), mover (overscript), munderover (underscript and overscript), mfrac (fraction), msqrt (square root), mroot

(root with explicit index), and several elements for representing a matrix. Because each symbol in MathML data is represented by unicode that provides a large number of mathematical symbols including ∂ and $\oint$, complex mathematical expressions, including $\int_{-\infty}^{\infty} f(x)\,\mathrm{d}x$ and $\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^{n}$, can only be described using the aforementioned elements and unicode characters.

Although Presentation Markup has notations for font face, font-color, and so on, we ignore this information. The font face of each symbol has the potential to be an effective feature for classification, for example, s, i, and n in $\sin x$ means "sine function" rather than "multiplication of s, i, and n" because their font faces are normal rather than italic. However, incorrect font face is notated frequently. In particular, optical character-recognition for mathematical expressions frequently ignores font face, especially bold. We hope to achieve robust classification, so we ignore font face.

Before learning and classification, we normalize the Presentation Markup data, because an identical mathematical expression could be represented as different data in Presentation Markup. For example, in the mathematical expression ab^2, either ab or only b can be read as being the base of the superscript 2. This diversity makes it difficult to manipulate locational data. To solve this problem, we previously proposed a normalization method for Presentation Markup data [3]. This normalization makes it possible to regard Presentation Markup data as tree structures for representing locational information. The tree structure fulfills the following conditions.

- There are three types of nodes: location nodes, integration nodes, and character nodes. Location nodes represent the location of symbols, e.g., superscripts or fractions. Integration nodes integrate multiple locational nodes and character nodes into a single node.
- The children of a locational node are integration nodes. The number of children is defined with respect to each locational node, e.g., a superscript has only one child and a fraction has two children as its numerator and its denominator.
- The children of an integration node are locational nodes and character nodes.
- A character node has no child.

For example, Fig. 2 depicts the tree structure of the expression $2x^{-a}+3$, which corresponds with normalized Presentation Markup data.

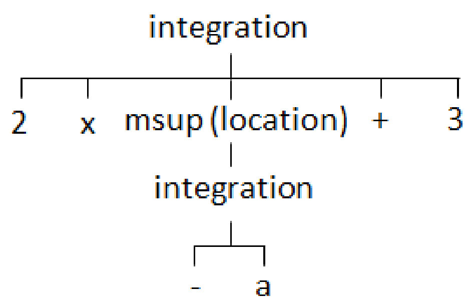

Fig. 2. Tree structure representing the expression $2x^{-a}+3$

In the rest of this paper, we regard the locational data of a mathematical expression as this tree structure.

2.2 Content Markup

In Content Markup, we focus on operators and special symbols including the imaginary unit and Napier's constant. Operators are the first child of an "apply" element and special characters are elements with no child.

For converting Presentation Markup data into Content Markup data, it is indispensable to obtain the operators and special symbols via symbol-level disambiguation. For instance, because Content Markup provides operators of "abs (absolute value)," "card (cardinality)," and "determinant," an ambiguous symbol "|" must be disambiguated. Also, i and e, which can be normal variables or special symbols, need to be disambiguated for elements with no child.

Content Markup provides a notation to describe a type of symbols (e.g., integers, real, function, and vector). However, Content Markup data is valid if it lacks the information of types. Therefore, we ignore the types in this research.

3. Classification

We regard symbol-level disambiguation as multi-label classification for symbols in mathematical expressions. Therefore, we construct classifiers for each symbol with supervised learning. This section details the labels and features for classification.

3.1 Labels

A label is attached to a location node or a character node in a tree in Sect. 2.1. As an example of attaching a label to a location node, a label of "power" is attached to a location node of "superscript" in x^2 because "power" is represented by the location, not x or 2.

Labels are the first child of an apply element and an element with no child in Content Markup. For example, a label for the symbol "|" is any one from "abs," "card," "determinant".

3.2 Features

As mentioned in Sect. 1, we suppose that the label of a symbol is decided by the frequency with which symbols appear in the same mathematical expression as their target ambiguous symbol, i.e., we use the frequency of nodes as features for classification. Here, we not only adopt the frequency of character nodes but also the frequency of location nodes.

We also consider the order of symbols in mathematical expressions. For instance, the character "i" on the right of "s" and the left of "n" is likely to be a part of a sine function. To represent the order of symbols, we use an *n*-gram, which is a sequence of *n* words or characters. However, because the symbols in mathematical expressions are not located horizontally, we define an *n*-gram for a tree structure of locational information on mathematical expressions. Our *n*-gram is the sequence of nodes that are the children of identical integration nodes in the tree structure of the locational data. For example, 2-grams of Fig. 2 are "2 x," "x msup," "msup +," "+ 3," and "− a." Although an *n*-gram for mathematical expressions is already proposed in [4], the *n*-gram considers operational information, for example, in $2y + z$, the 2-gram is only "$2y$." Our 2-grams of $2y + z$ are "$2y$", "$y +$" and "$+z$." The frequencies of the *n*-grams are also features for our classification.

4. Experiment

We conduct a brief experiment to validate our framework for symbol-level disambiguation.

4.1 Data

There are several datasets of mathematical expression, including [5], which provides 200 annotated mathematical expressions; however, it is difficult to directly use such datasets for our classification. Therefore, we create a new dataset as training data for this experiment.

Mathematical expressions are collected from the Wolfram Function Site [6], which provides both Presentation Markup data and Content Markup data of an identical mathematical expression. We normalized the Presentation Markup data, extracted labels from the Content Markup data, and manually attached labels to nodes. We collected 220 labeled mathematical expressions that belong to “Sqrt” of the “Elementary Functions” category on the Wolfram Functions Site. This dataset includes 11 nodes that are attached to more than two types of labels and that appear more than 30 times. We disambiguated these 11 nodes in this experiment.

4.2 Algorithm and Condition

The algorithm for the classification is “random forest [7]”. Random forest is one of the popular classification algorithm, which is based on ensemble learning methodology. The characteristics of random forest as a classifier are low-bias and low-variance, and many experiment reported the high accuracy of random forest as a classifier. We use scikit-learn [8], which is a library for machine-learning in Python, to implement our framework. The parameters of the random forest algorithm are default values in scikit-learn 0.16.1.

Random forest requires a vector expression as features. In our experiment, a feature vector consists of the frequencies of nodes, 2-grams, and 3-grams in the tree of a mathematical expression.

4.3 Results

We conduct two-fold cross-validation. The classifiers for eight symbols achieve better accuracy than the baseline, which is the relative frequency of the most frequent label. The worst classifier is for the symbol “/”, whose accuracy is 73% (the baseline is 74.6%). The Wolfram Functions Site uses the particular notation “/;” to represent “the following mathematical expression of /; indicates the condition.” The “/” in “/;” is labeled with “Condition” because the corresponding Content Markup data contains “Condition” as the first child of an apply element. When “/” represents division, the “/” is labeled as “divide.” The cause of low accuracy is believed to be the frequency of symbols and *n*-grams in mathematical expressions containing “/;” are not important clues, because “/;” is not general notation. The accuracies of the classifications of msub and mfrac, which are location nodes, are also worse than the baseline. This is because most msup are labeled with “power” and a few are “arctan” or “partialdiff,” and most mfrac are labeled with “divide” and a few are “partialdiff.”

The classifiers for symbols that could be a part of operation names show notable results. The classifier “g,” whose labels are “sign,” “arg,” “ln,” and “functionName,” shows 90.8% accuracy and the baseline is 77.8%. The top 5 important features based on mean decrease in Gini impurity are the frequencies of “l”, (“2”, “g”, “(“), (“a”, “r”), (“a”, “r”, “g”), and (“=”, “c”, msub). Here, the sequences enclosed by parentheses indicate *n*-grams. The top 5 important features include the frequencies of four *n*-grams, which show that *n*-grams for mathematical expressions work well for classification.

5. Application

Our main objective of symbol-level disambiguation is the preprocessing of the conversion of Presentation Markup data into Content Markup data. In addition, symbol-level disambiguation would contribute to another application, an advanced search tool for mathematical expressions.

Some search tools allow users to input mathematical expressions as queries [9, 10]. However, the search results may be inappropriate for users because of symbol-level ambiguity. For example, suppose a user who hopes to retrieve an absolute value of A inputs $|A|$ as the query; in this case, the search result includes $|A|$, which represents not only an absolute value, but also a cardinality or a determinant. In addition, $\mathrm{abs}(A)$ is not matched to the query, even if $\mathrm{abs}(A)$ is described as an absolute value of A. If a search tool allows users to input both a mathematical expression and an operator name as a query and attaches labels to each symbol in target mathematical expressions using our classifiers, the search tool will omit $|A|$ as a cardinality or a determinant and provide $\mathrm{abs}(A)$.

6. Conclusion

In this paper, we proposed a framework for symbol-level disambiguation of mathematical expressions to convert Presentation Markup data into Content Markup data. The feature vectors of symbols in mathematical expressions are generated by normalizing the Presentation Markup data to regard the data as a tree and counting the frequency of nodes and *n*-grams in the tree. We implemented the proposed method and experimented briefly with 220 annotated mathematical expressions. As a result, the accuracy of our classification is higher than the relative frequency of a most frequent label, regardless of the limited training data.

In the future, we plan to expand the dataset of annotated mathematical expressions. The mathematical expressions will be collected from various fields of mathematics. Such a large and variable dataset would enable us to enhance the accuracy of the classification and increase the number of types of symbols to classify.

Acknowledgment

This work was supported by Japan Society for the Promotion of Science (JSPS) Grant-in-Aid for JSPS Fellows Grant Number 26-2758.

References

[1] M. Q. Nghiem, G. Y. Kristianto, G. Topic, A. Aizawa, “Sense Disambiguation: from Natural Language Words to Mathematical Terms,” The 6th International Joint Conference on Natural Language Processing (IJCNLP 2013), pp.809-814, (2013).
[2] MathML. http://www.w3.org/TR/MathML3/
[3] T. Watabe, Y. Miyazaki, “Pattern Matching Algorithm for Mathematical Expressions with a Regular Expression,” IPSJ Journal, Information Processing Society of Japan, Vol.56, No.5, pp.1417-1427 (2015).
[4] R. Miner, R. Munavalli, “An Approach to Mathematical Search Through Query Formulation and Data Normalization,” Towards Mechanized Mathematical Assistants, Springer, pp.342-355 (2007).
[5] M. Wolska, M. Grigore, M. Kohlhase, “Using Discourse Context to Interpret Object-denoting Mathematical Expressions,” Towards a Digital Mathematics Library, pp.85-101 (2011).
[6] Wolfram Functions Site. http://functions.wolfram.com/
[7] J. Friedman, T. Hastie, R. Tibshirani, “The Elements of Statistical Learning,” Vol.1, Springer Berlin (2001).
[8] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,” Journal of Machine Learning Research, Vol.12, pp.2825-2830 (2011).
[9] H. Hashimoto, Y. Hijikata, S. Nishida, “A Survey of Index Formats for the Search of MathML Objects,” The Special Interest Group Technical Reports of IPSJ, Information Processing Society of Japan, Vol.2007, No.54, pp.55-5 (2007).
[10] T. Watanabe, Y. Miyazaki, “Development of IR Tool for Tree-Structured MathML-based Mathematical Descriptions,” International Conference on Computers in Education (ICCE2010), pp.310-312 (2010).