Classification of Presentation MathML Expressions Using Multilayer Perceptron

Classification of Presentation MathML Expressions Using Multilayer Perceptron

Yuma NAGAO

Graduate School of Library, Information and Media Studies
University of Tsukuba

March 2018

Contents

Chapter1

Introduction

MathML (Mathematical Markup Language) is a markup language for describing math expressions. MathML consists of two set of elements: Presentation Markup and Content Markup. The former describes layout structure of math expressions, and is widely used to display math expressions in Web pages. On the other hand, the latter describes semantic structure of math expressions, and is suited to automatic calculation of math expressions. One of the challenging problem related to Presentation MathML is classification, i.e., given a MathML expression e, identify the class (e.g., hypergeometric function, bessel-type function, etc.) that e belongs to. If we can identify the class of a given Presentation MathML expression automatically, it is helpful for various applications, e.g., Presentation to Content MathML conversion, text-to-speech, and so on.

In the notation of math expressions, a token may have multiple meanings. We give two expressions as an example.

$$C_n = \frac{1}{n+1}\binom{2n}{n} \tag{1.1}$$

$$C(x) = \int_0^x \cos(t^2)\,dt \tag{1.2}$$

(1.1) represents nth Catalan number. C is denoted as `<mi>C</mi>` in Presentation MathML and `<ci>Catalan</ci>` in Content MathML. (1.2) represents Fresnel integral. As with (1.1), C is denoted as `<mi>C</mi>` in Presentation MathML but `<ci>FresnelC</ci>` in Content MathML. In the categories of The Wolfram Functions Site[8], (1.1) may belong to Constants and (1.2) may belong to Gamma, Beta, Erf. When converting from Presentation to Content MathML, it is necessary to distinguish such discrimination of meaning appropriately. It is considered that such conversion can be possible by classification of Presentation MathML expressions.

In this thesis, we propose a classification method for Presentation MathML expressions. Our method classifies MathML expressions by using multilayer perceptron, which is a kind of deep learning model having a simple structure. The difficulty in taking such an approach is that the size of MathML expressions are arbitrary, while multilayer perceptron requires input of fixed length. Thus, it is impossible to input MathML expressions to multilayer perceptron directly. To address

this problem, our method converts a Presentation MathML expression into a fixed length vector, which is based on binary branch vector [9]. We train a multilayer perceotron by using such vectors and classify MathML expressions by the multilayer perceptron. Experimental results show that our method classifies math expressions with high accuracy.

Related Work

Kim et al. [1] proposed a classification method for Presentation MathML expressions. They extract features from math expressions and classify them by using support vector machine (SVM). They use labels of nodes and contiguous sequence of leaf nodes as a feature, in which parent-child relationships of tree structures are not considered. A drawback of SVM is that SVM requires proper features which must be extracted and selected according to characteristics of data manually. On the other hand, deep learning (e.g., multilayer perceptron) can extract features automatically from input data.

There are some researches on semantic enrichment from math expressions [6, 3, 5, 4]. The method proposed in [6] extracts semantic meaning of math identifiers from math expressions and texts surrounding the expressions. [3] proposed a method for transforming a Presentation MathML expression to a Content MathML expression. This method uses classes of math expressions as features. On the other hand, we estimate the classes of math expressions without using texts surrounding the expressions. [5] proposed a method for detecting math tokens which have multiple meanings based on Presentation/Content MathML parallel corpora. [4] proposed a method for classifying math documents.

Chapter2

Preliminaries

In this chapter, we present an overview of MathML and Binary Branch Vector.

2.1 Overview of MathML

MathML (Mathematical Markup Language) is an XML-based markup language for describing math expressions which is recommended by W3C. MathML has two kinds of notations. The first is Presentation Markup (Presentation MathML) which is used to display math expressions. The second is Content Markup (Content MathML) which is used to convey the semantic structure of math expressions. Figure 2.1 and Figure 2.2 show math expressions $x^2 + 1$ in Presentation MathML and Content MathML, respectively. In Presentation MathML, `mn` is used for numerical number, `mi` for identifier, `mo` for operator. These elements have a text node as its child. To represent numerical number "2", it is written as `<mn>2</mn>`.

```
<math>
  <msup>
    <mi>x</mi>
    <mn>2</mn>
  </msup>
  <mo>+</mo>
  <mn>1</mn>
</math>
```

Figure2.1 Presentation Markup

```
<math>
  <apply>
    <plus/>
    <apply>
      <power/>
      <ci>x</ci>
      <cn>2</cn>
    </apply>
    <cn>1</cn>
  </apply>
</math>
```

Figure2.2 Content Markup

An XML document is represented as an unranked ordered tree, and thus a MathML expression can also be represented as an unranked ordered tree. Unranked tree is a tree whose node has arbitrary number of children. Ordered tree is a tree which distinguishes order of sibling nodes. An *unranked ordered tree* is denoted $T = (N, E, Root(T))$, where N is the set of nodes, E is the set of edges and $Root(T)$ is the root node of T. By $(u, v) \in E$ we mean an edge from parent node u to child node v.

2.2 Binary Branch Vector

Binary Branch Vector[9] is a kind of vector representation of labeled ordered tree. This is originally proposed for reduction of computing a distance cost between two trees but we apply this to inputs of a multilayer perceptron.

A *full binary tree* (*binary tree* for short) is denoted $B(T) = (N, E_l, E_r, Root(T))$, where E_l and E_r are the sets of left edges and right edges in $B(T)$, respectively. If v is the first child of u, then we write $\langle u, v\rangle_l \in E_l$. Similarly, if v is the second child of u, then we write $\langle u, v\rangle_r \in E_r$.

A *binary branch* is a one level subtree of a binary tree. This consists of root node, left child and right child. Formally, binary branch $BiB(u)$ is defined as $BiB(u) = (N_u, E_{u_l}, E_{u_r}, Root(T_u))$, where

- $N_u = \{u, u_1, u_2\}(u \in N; u_i \in N \bigcup \{\epsilon\}, i = 1, 2)$,
- $E_{u_l} = \{\langle u, u_1\rangle_l\}$,
- $E_{u_r} = \{\langle u, u_2\rangle_r\}$,
- $Root(T_u) = u$.

A *binary branch vector* $BRV(T)$ of a tree T is defined as $BRV(T) = (b_1, b_2, ..., b_{|\Gamma|})$, where Γ is an ordered set of binary branches in a dataset, $|\Gamma|$ is the size of Γ and b_i represents the number of occurrences of the ith binary branch in $B(T)$.

2.3 Multilayer Perceptron

Multilayer perceptron is a kind of deep learning model. Many classification problems have been resolved by SVM previously. However, deep learning models including CNN (Convolutional Neural Network) get better result than SVM in recent years. In this thesis, we use multilayer perceptron which is the simplest deep learning model as a first step of classifying MathML expressions by deep learning model.

A multilayer perceptron is composed of an input layer, any number of hidden layers and an output layer. Each layer consists of units which receive multiple inputs and calculate an output (see Fig. 2.3). The net input $u_j^{(l+1)}$ of jth unit in $l+1$th layer is

$$u_j^{(l+1)} = \sum_i w_{ji}^{(l)} z_i^{(l)} + b_j^{(l+1)}, \tag{2.1}$$

where $w_{ji}^{(l)}$ is a weight, $b_j^{(l+1)}$ is a bias and $z_i^{(l)}$ is the output of $u_i^{(l)}$. Here, the output z_i of u_i is

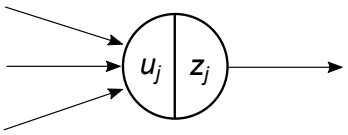

Figure2.3 Construction of a unit

given as $z_i = f(u_i)$, where f is an activation function. ReLU is a popular activation function for multilayer percptron defined as follows.

$$f(u) = \max(0, u). \tag{2.2}$$

For multi-class classification, softmax function

$$z_k = \frac{\exp(u_k)}{\sum_i \exp(u_i)} \tag{2.3}$$

is generally used for output layer. The output z_k of softmax represents the probability that a given input belongs to class C_k. Weights and biases are updated in order to minimize the error between outputs and correct answers of training data. In the case of multi-class classification, the cross entropy error function

$$E = -\sum_k t_k \log z_k \tag{2.4}$$

is generally used, where z_k is an output and t_k is kth element of a correct answer. SGD and Adam are known as typical optimizers to minimize the error (details are omitted).

Chapter3

Classification Method of MathML Expressions

Elements of MathML expressions have arbitrary number of children. On the other hand, a multilayer perceptron requires input of fixed length. Therefore, we need to convert MathML expressions to vectors of fixed length. Figure 3.1 illustrates the overview of our method.

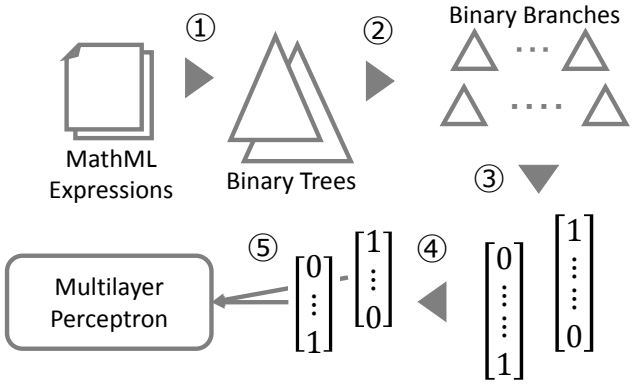

Figure3.1 Overview of Classifying Method

Our method mainly consists of the following 5 steps.

1. Convert MathML expressions to binary trees.
2. Split the binary trees to binary branches.
3. Convert the binary branches to binary branch vector.
4. Reduce the dimension of the binary branch vector.
5. Input the vectors to multilayer perceptron and classify them.

We introduce steps 1 to 3 in Sec. 3.1, step 4 in Sec. 3.2 and step 5 in Sec. 3.3.

3.1 Vector Construction

MathML expressions are represented as trees of various sizes. On the other hand, most classification algorithm (e.g., multilayer perceptron, SVM, etc.) requires input of fixed length. Therefore, we must convert MathML expressions to vectors of fixed length. In this section, we give a procedure for converting MathML expression to binary branch vector.

Let T be an unlanked labeled tree. A binary branch vector $BRV(T)$ is obtained from a binary tree $B(T)$. A binary tree $B(T)$ is obtained from T as follows. Initially, $B(T)$ consists of the same set of nodes as T and no edges.

1. For each node u in T, do the following.
   (a) Let $v_1, v_2, ..., v_n$ be the children of u. Add an edge (v_{i-1}, v_i) to $B(T)$ for every $2 \leq i \leq n$.
   (b) Add an edge (u, v_1) to $B(T)$.
2. Insert empty node ϵ so that all internal nodes in $B(T)$ have exactly two child nodes.

We represent the conversion algorithm in Algorithm 1. The algorithm traverses MathML expressions and constructs binary branch vectors at the same time. The original binary branch vector construction algorithm proposed in [9] includes processes to record positions of branches for computing distance between two trees, which is unnecessary when used for inputs of multilayer perceptron. Therefore, we omitted the processes. For a given dateset of MathML expressions, all MathML expressions in the dataset are traversed and the number of occurrences of binary branches are recorded in an associative array H by *VectorConstruction* function. A key in H is a tuple of a binary branch and the elements of the tuple are of string type. The keys are listed in alphabetical order of first element. A value of H is an array whose size is the number of MathML expressions in the dataset. Each element in the array represents the number of occurrences of binary branch. *Traverse* function traverse a MathML expression recursively and store the number of occurrences of binary branches in H. In this function, *first* and *next* are the first child and the right sibling of the current node, respectively. If the current node does not have any child node or next sibling, $first$ or *next* is assigned an empty node ϵ whose label is also ϵ. Then the function increases the number of occurrences of the binary branch of the current node in the ith MathML expression. After construction of H, each element in the H is pushed to *BRVs*. Finally, we transpose *BRVs* so that we obtain column vectors.

We give an example in Fig. 3.2. In the dataset, there are two binary trees $B(T_1)$ and $B(T_2)$ converted from MathML expressions T_1 and T_2. Each cell in the right table represents the number of occurrences of a binary branch. For example, binary branch (math, mn, ϵ) occurs once in $B(T_1)$ and $B(T_2)$. Therefore, the two columns of the table represent binary branch vectors corresponding to $B(T_1)$ and $B(T_2)$, respectively. That is, $BBV(T_1) = (1, 1, 0, 1, 0, 1, 0, 1, 0)$ and $BBV(T_2) = (1, 0, 1, 0, 1, 0, 1, 0, 1)$.

Algorithm 1 Constructing binary branch vectors

Input:

A set D of MathML expressions

Output:

Binary branch vectors $BRVs$

```
1: function VectorConstruction(D)
2:     Initialize an associative array H to be empty
3:     i ⇐ 0
4:     for all MathML expression e in D do
5:         root ⇐ root node of e
6:         Traverse(i, root, H)
7:         i++
8:     end for
9:     for all value in H do
10:        Push value to BRVs
11:    end for
12:    return transposition of BRVs
13: end function

14: function traverse(i, node, H)
15:    first ⇐ first child of node
16:    next ⇐ next sibling of node
17:    Hash[(node, first, next)][i]++
18:    for all child in children of node do
19:        traverse(i, child, H)
20:    end for
21: end function
```

3.2 Dimensionality Reduction

Each element of a binary branch vector is the number of occurrences of a binary branch in a dataset. Therefore, as the dataset becomes larger, the size of Γ increases accordingly. In particular, the number of binary branches tends to considerably grow when MathML expressions have substructures which are similar but have different numeric values. However, such "non-structural" differences may reduce the accuracy of classification. This is because similar MathML expressions tend to contain similar substructures that have different numeric values, therefore MathML expressions that should be classified into the same class may wrongly be classified into different classes due to substructures

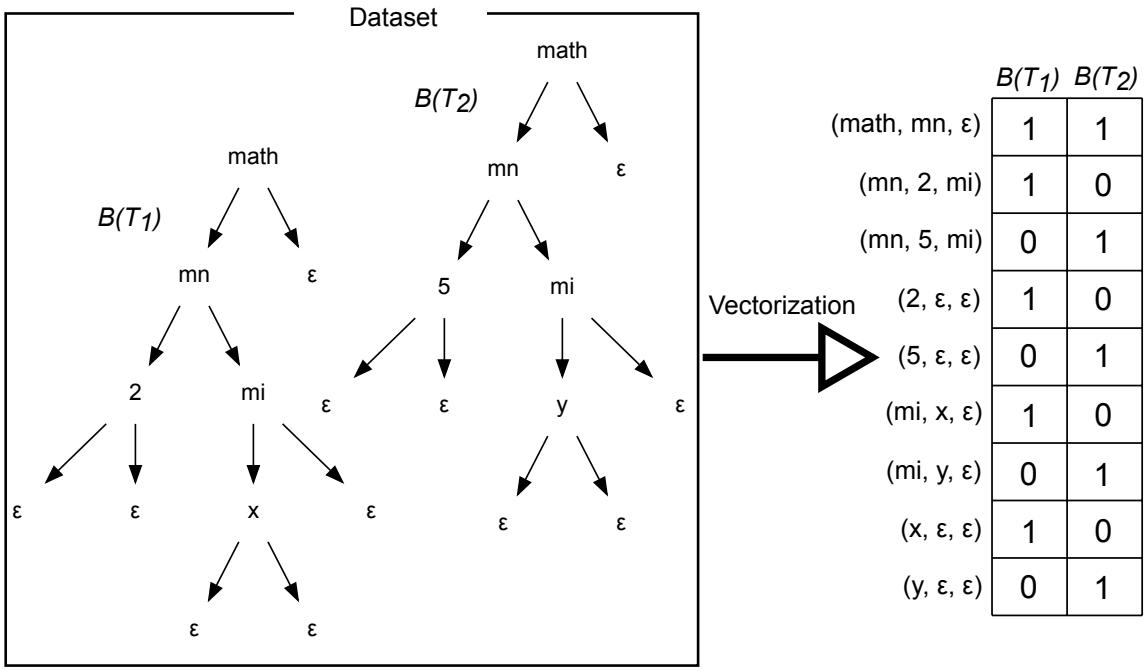

Figure3.2 Converting MathML expressions to binary branch vectors

that are similar but different numeric values. Thus, we reduce the sizes of binary branch vectors by erasing the text of `mn` elements. In Presentation MathML, `mn` elements represent numeric numbers.

We represent the dimensionality reduced vector construction algorithm in Algorithm 2. In this algorithm, *VectorConstruction* function is the same as Algorithm 1, so it is omitted in Algorithm 2. $Traverse$ function in Algorithm 2 also traverses e and stores the number of occurrences of binary branches in H. The major difference is the handling of `mn` elements. If a label of the current node *node* is "mn", the function increases the number of occurrences of binary branches $(node, \epsilon, next)$ and ("mn", ϵ, ϵ) in the ith MathML expression. In other cases, the function traverse the child nodes of *node* in the same way as Algorithm 1.

We give an example of dimensionality reduction. Consider the binary trees in Fig. 3.2. We replace the texts of `mn` elements (2 and 5) with ϵ. As the result, binary branches rooted by `mn` elements are converted to the same binary branches and we obtain two binary trees $B(T_1)'$ and $B(T_2)'$ in Fig. 3.3. In comparison with binary branch vectors in Fig. 3.2, the binary branch vectors in Fig. 3.3 are of smaller length.

Algorithm 2 Constructing dimensionality reduced vector

Input:

A tree id i,

A node of the tree $node$,

An associative array H

```
1: function TRAVERSE(i, node, H)
2:     first ⇐ first child of node
3:     next ⇐ next sibling of node
4:     if label of node is "mn" then
5:         H[(node, ϵ, next)][i] + +
6:         H[("mn", "ϵ", "ϵ")][i] + +
7:     else
8:         H[(node, first, next)][i] + +
9:         for all child in children of node do
10:            TRAVERSE(i, child, H)
11:        end for
12:    end if
13: end function
```

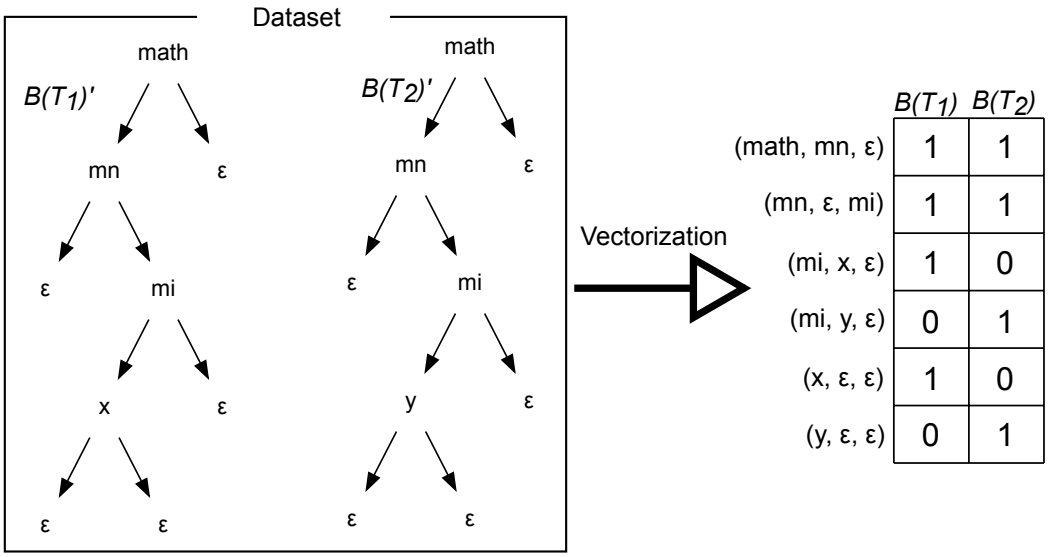

Figure3.3 Dimensionality reduction by converting text of mn element to empty node ϵ

3.3 Classification Model

The vectors obtained by Algorithm 2 are classified by a multilayer perceptron. Our model consists of an input layer, two hidden layers and an output layer. The number of units in the input layer is

same as the size of binary branch vectors. Each hidden layers has 512 units and activated by ReLU. The output layer has the same number of units as MathML classes and activate by softmax. We use Adam as the optimizer to minimize the cross entropy error. We also tried other models (e.g., one with three or more hidden layers) but found no noticeable improvement.

Chapter4

Experiment

In this chapter, we present our experimental results.

4.1 Dataset

In our experiments, we use the following two datasets: The Wolfram Functions Site and MREC (Mathematical REtrieval Collection). In the following, we give the details of the two datasets.

4.1.1 The Wolfram Functions Site

We collected HTML files exhaustively and extracted MathML expressions from The Wolfram Functions Site [8]. We obtained 307,676 MathML expressions and 14 classes. Table 4.1 shows the classes and the numbers of MathML expressions belongs to it.

Expressions in this dataset are Presentation MathML, although the expressions are embedded Content MathML by `semantics` elements and `annotation-xml` elements as shown in Figure 4.1. Therefore, we remove `annotation-xml` elements and unnest `semantics` elements. Furthermore, we remove `mtext` elements and attributes of elements, which do not influence to mathematical meanings. The former are used for display spaces and the latter are used for alignment of rendering of math tokens.

In our experiment, we used 70% of the dataset for training data and used 30 % for test data.

4.1.2 MREC (Mathematical REtrieval Collection)

MREC [2] is a dataset of scientific papers in arxiv.org [*1] translated to XML. The math expressions in the papers are written in Presentation MathML. MREC is composed of MREC2011.3 and MREC2011.4. The expressions in the latter have embedded semantic meanings and the former expressions do not. Since our method does not use specified semantic meaning of expression, we used

[*1] https://arxiv.org/

Table4.1 Number of MathML expressions in each class

Class id	Class Name	# of Expressions
0	Bessel-Type Functions	5,583
1	Complex Components	689
2	Constants	735
3	Elementary Functions	61,454
4	Elliptic Functions	7,248
5	Elliptic Integrals	1,236
6	Gamma, Beta, Erf	5,009
7	Generalized Functions	280
8	Hypergeometric Functions	218,241
9	Integer Functions	1,966
10	Mathieu & Spheroidal Functions	388
11	Number Theory Functions	904
12	Polynomials	2,372
13	Zeta Functions & Polylogarithms	1,571
	total	307,676

the former for the experiment.

MREC contains too short expressions (e.g., x, Δ, etc.) for which classification is meaningless. Therefore, we removed MathML expressions which do not have `mi` elements or less than two elements except `math` and `mrow` elements from the dataset. Note that a `math` element is the root node of a MathML expression and that `mrow` elements influence neither display nor meaning of a MathML expression.

The classes of arxiv.org consist of 8 classes: Physics, Mathematics, Computer Science, Quantitative Biology, Quantitative Finance, Statistics, Electrical Engineering and Systems Science, Economics. We refer to these classes as top-class. The top-classes have child classes, which are subdivided by academic field. For example, the subclasses of top-class Physics are Physics, Astrophysics, Quantum Physics, and so on. We refer to these child class as sub-class. The sub-classes with the same name as its top class are further subdivided. For example, sub-class Physics has Accelerator Physics, Applied Physics, and so on as child classes. We refer to these child classes as sub-sub-class.

The MREC dataset has originally 34 classes based on the sub-classes and sub-sub-classes of arxiv.org. We chose 20 classes that have more than 100,000 expressions. Some classes have parent-child relationships or are very similar to each other, and that these classes should be merged into the same class. We merged similar classes that seem to be the same field. For example, High Energy Physics - Experiment, High Energy Physics - Lattice and High Energy Physics - Theory are sub-class of top-class Physics. These classes include the words “High Energy Physics”. Therefore, they

```
<math xmlns='http://www.w3.org/1998/Math/MathML'
        mathematica:form='TraditionalForm'
        xmlns:mathematica='http://www.wolfram.com/XML/'>
  <semantics>
    <mrow>
      <mrow>
        <mi> F </mi>
        <mo> &#8289; </mo>
        <mo> ( </mo>
        <mrow>
          <mi> z </mi>
          <mo> &#10072; </mo>
          <mn> 0 </mn>
        </mrow>
        <mo> ) </mo>
      </mrow>
      <mo> &#10869; </mo>
      <mi> z </mi>
    </mrow>
    <annotation-xml encoding='MathML-Content'>
      <apply>
        <eq />
        <apply>
          <ci> EllipticF </ci>
          <ci> z </ci>
          <cn type='integer'> 0 </cn>
        </apply>
        <ci> z </ci>
      </apply>
    </annotation-xml>
  </semantics>
</math>
```

Figure4.1 Example of MathML expression in The Wolfram Functions Site

can be considered to be classes which related to the same High Energy Physics. Similarly, Nuclear Experiment and Nuclear Theory are merged. On the other hand, Algebraic Geometry, Differential Geometry and Quantum Algebra are sub-sub-classes of sub-class Mathematics. Therefore, they can

be considered to be classes which related to the same sub-class Mathematics. As the result of merge, we obtained the following 12 classes. Original class names in MREC are in parentheses. For example, class 5 consists of hep-ex, hep-lat and hep-th.

1. Astrophysics (astro-ph)
2. Computer Science (cs)
3. Condensed Matter (cond-mat)
4. General Relativity and Quantum Cosmology (gr-qc)
5. High Energy Physics - Experiment (hep-ex),
   High Energy Physics - Lattice (hep-lat),
   High Energy Physics - Theory (hep-th)
6. Algebraic Geometry (alg-geom),
   Differential Geometry (dg-ga),
   Mathematics (math),
   Quantum Algebra (q-alg)
7. Mathematical Physics (math-ph)
8. Chaotic Dynamics (chao-dyn),
   Nonlinear Sciences (nlin),
   Exactly Solvable and Integrable Systems (solv-int)
9. Nuclear Experiment (nucl-ex),
   Nuclear Theory (nucl-th)
10. Physics (physics)
11. Quantitative Biology (q-bio)
12. Quantum Physics (quant-ph)

We extracted 100,000 MathML expressions from each of the 12 classes. We used 70% of them for training data and used 30 % for test data.

4.2 Experimantal Results

We compare our method and the SVM-based method proposed by Kim et al. [1]. They define five features: labels of nodes (Tag), texts of `mo` elements which represent operators (Operator), texts of `mi` elements which represent identifiers (Identifier), bigram of plain text in expressions (String Bigram) and bigram of identifier and operator (I&O). To classify MathML expressions, they compared several combinations of the features as the inputs of SVM with liner kernel. In their experimental results, the combination of Tag, Operator, String Bigram and I&O shows the highest accuracy. However, the expressions in our datasets contain few plain texts that can be used as the String Bigram feature. Therefore, we use Tag, Operator, Identifier and I&O, which marked the second highest accuracy in their experiment. We adjust a penalty parameter C of SVM ($C = 2^{-10}, 2^{-9}, ..., 2^{10}$) because the

value of C is not specified in their paper.

Tables 4.2 and 4.4 show the results. Table 4.2 shows the accuracies of the three methods in The Wolfram Functions Site. We experimented our method with different two input vectors : dimensionality reduced binary branch vector and binary branch vector without dimensionality reduction processing. We refer to the former as our method without `mn` and the latter as our method with `mn`. The result shows that every method achieves high accuracy. Further, our method slightly outperforms the SVM-based method ($C = 2^8$). The reason why such high accuracies are obtained is that the dataset is "clean", that is, it has well-formed structures (e.g., order of variables, operators, etc.) and unified notations of identifiers. Our method without `mn` slightly outperforms our method with `mn` for the dataset. From this result, it is considered that there are no decrease in accuracy due to our dimensionality reduction processing.

Table4.2 Accuracy for The Wolfram Functions Site

Method	Accuracy
Our method without `mn`	**99.35**
Our method with `mn`	99.24
SVM [1]	99.03

Figure 4.2 and Figure 4.3 show the details of the classification results of our method without `mn` and SVM-based method, respectively. The numbers on each axis represents the class number of The Wolfram Functions Site in Table 4.1. The vertical axis represents correct classes and the horizontal axis represents predicted classes. Let (C_{ans}, C_{prd}) be the cell whose vertical (resp., horizontal) index is C_{ans} (resp., C_{prd}), where C_{prd} and C_{ans} are class ids of correct answer and predicted answer, respectively. The value of (C_{ans}, C_{prd}) is represented as follows.

$$V(C_{ans}, C_{prd}) = \frac{\text{The number of expressions which belong to } (C_{ans}, C_{prd})}{\text{The number of expressions which belong to } C_{ans}}. \tag{4.1}$$

$V(i, i)$ represents recall for each class.

As shown in the figures, there are many misclassification cases in Elementary Functions or Hypergeometric Functions. It is considered that the fact that approximately 90% of training data belong to Elementary Functions or Hypergeometric Functions is one of the causes for the result. Especially, the rate of misclassifying in Elementary Functions is higher than other classes. This is caused by the character of the class. The expressions in Elementary Functions are mathematical basic functions (e.g., power functions, logarithmic functions, etc.). Therefore, it is considered that structural elements such as similar operators appear in other classes and thus classifiers misclassified.

Table 4.3 shows F-measures of each method. Our method achieves F-measures equal to or higher than that of the SVM-based method except for one class (ID:12). Since the gap between the F-measures in the class is small, our method generally shows a good performance for The Wolfram

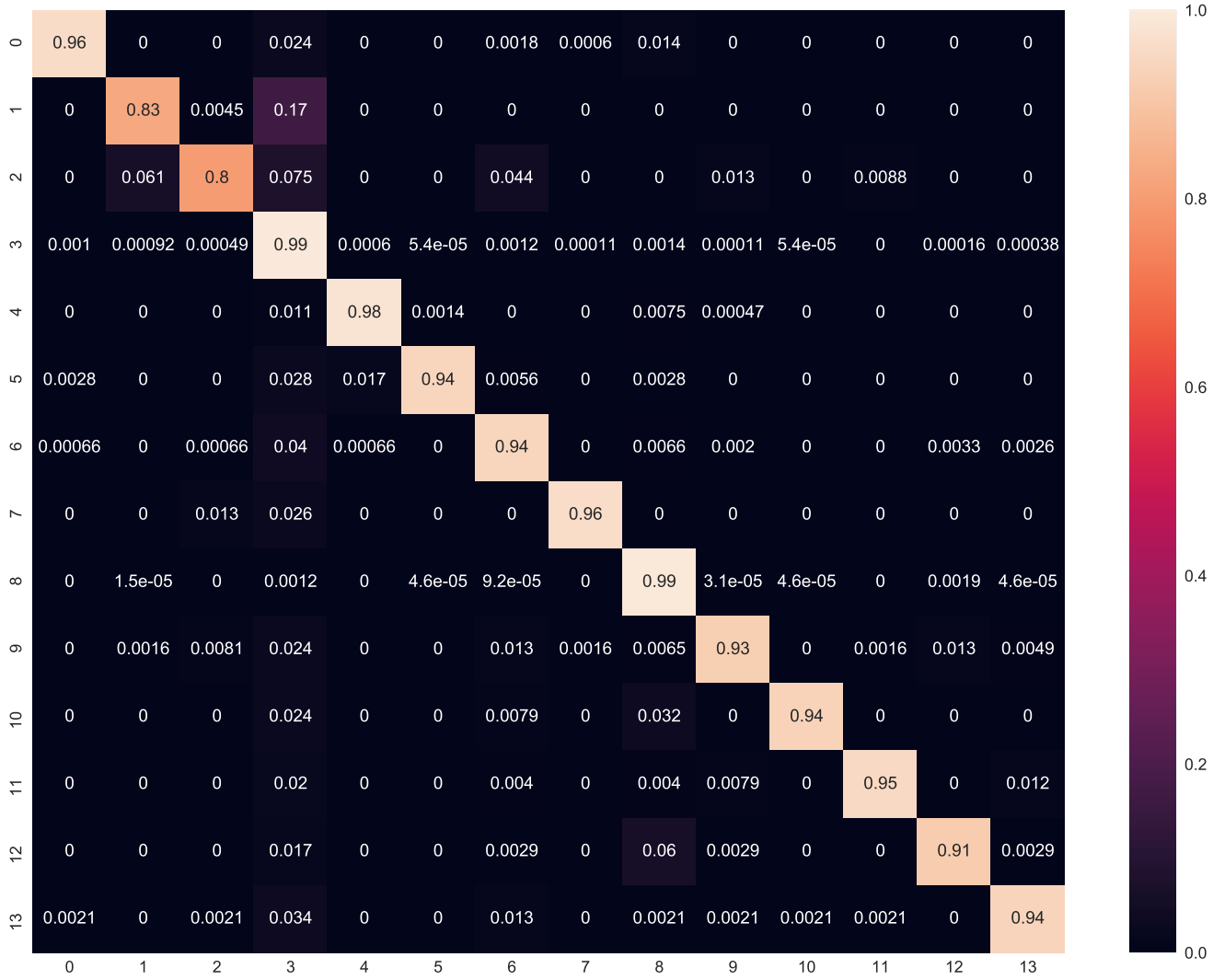

Figure4.2 Normalized confusion matrix (our method without `mn`) for The Wolfram Functions Site

Functions Site.

Table 4.4 shows the accuracies of the two methods for MREC. Both of our methods outperform the SVM-based method ($C = 2^0$). However, every accuracy is lower than its corresponding accuracy of The Wolfram Functions Site. This is because the expressions of MREC are much less "clean" than these of The Wolfram Functions Site. For example, in the expressions of MREC the notation of identifiers are not unified, because the expressions are written by various authors. The result means that our method is much robust and effective to less "clean" expressions than the SVM-based method. In comparison our method (without `mn`) with our method (with `mn`), there is a wide gap in accuracy, and it is a future work to elucidate cause of this.

Finally, let us consider the sizes of input vectors. Here, we denote binary branch vector before dimensionality reduction as *BRV with mn* and binary branch vector after dimensionality reduction

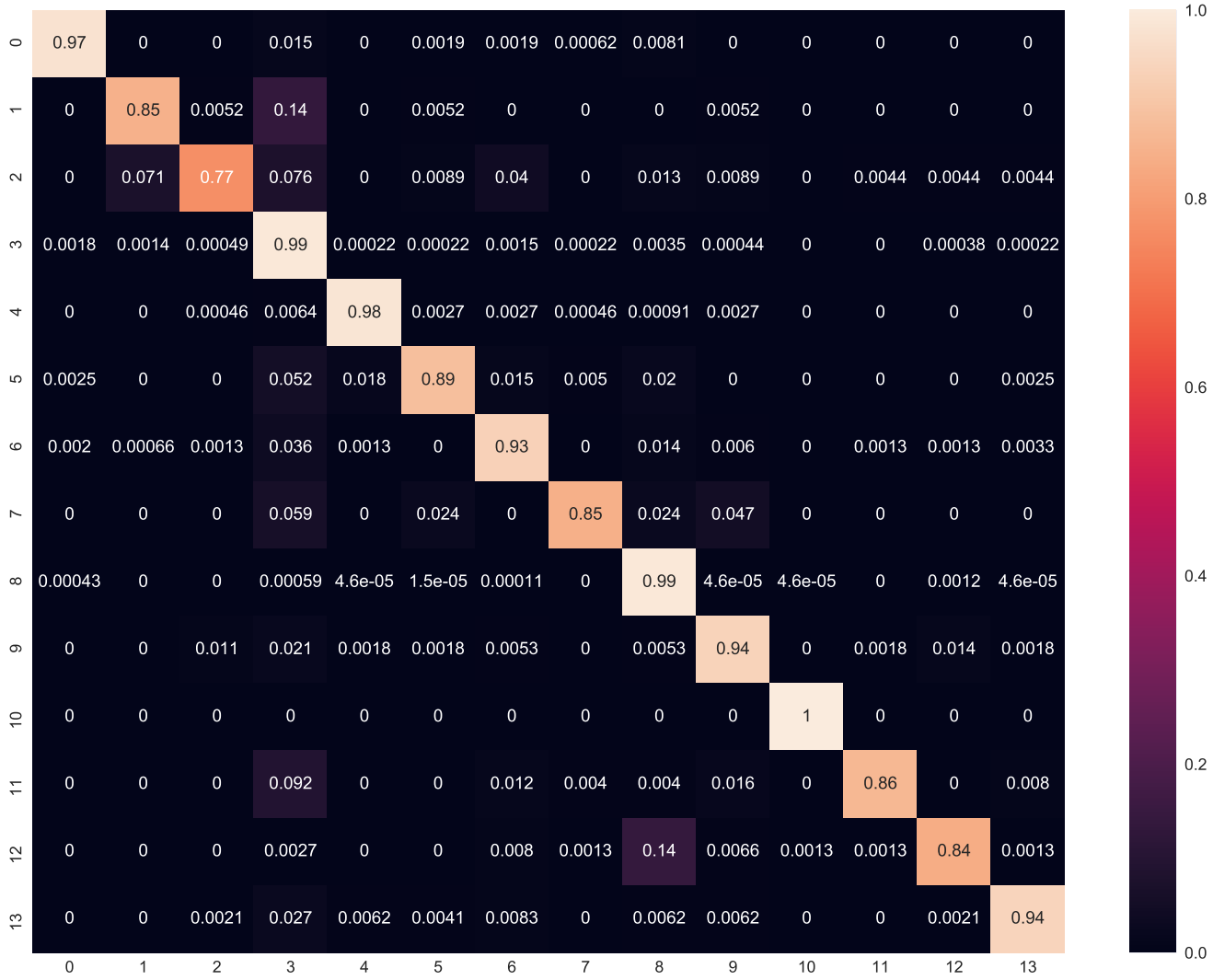

Figure4.3 Normalized confusion matrix (SVM[1]) for The Wolfram Functions Site

as *BRV without mn*. For The Wolfram Functions Site, the sizes of BRVs without `mn` are 2,184 and BRVs with `mn` are 558,777. This means that similar substructures which have different numeric values frequently appear in the dataset. The size of BRVs with `mn` are smaller than the sizes of feature vectors for the SVM-based method (8,113). For MREC, the sizes of BRVs without `mn` are 39,833 and BRVs with `mn` are 72,019. Since the sizes of feature vectors for the SVM-based method are 91,146, our vectors are smaller. Thus, by using our method, we can obtain better classification results with smaller input vectors.

Table4.3 F-measures of our method and SVM-based method

	F-measure	
Class ID	Our method without `mn`	SVM[1]
0	0.97	0.97
1	**0.85**	0.80
2	**0.87**	0.85
3	0.99	0.99
4	0.99	0.99
5	**0.97**	0.93
6	**0.95**	0.94
7	**0.94**	0.90
8	0.99	0.99
9	**0.94**	0.93
10	0.95	0.95
11	**0.96**	0.92
12	0.85	**0.86**
13	**0.95**	0.94

Table4.4 Accuracy for MREC

Method	Accuracy
Our method without `mn`	**83.54**
Our method with `mn`	33.75
SVM [1]	28.16

Chapter5

Conclusion

In this thesis, we proposed a method for classifying MathML expressions based on multilayer perceptron. Experimental results showed that our method can classify MathML expressions with higher accuracy than the SVM-based method and we succeeded in reducing the size of binary branch vectors without a decline in accuracy by our dimensionality reduction processing.

As a future work, it is necessary to clarify the cause of the difference in accuracy in MREC. Furthermore, we need to consider comparing our model with more complex models (e.g., Tree-LSTM[7]). We also have to tune hyperparameters (number of units and layers, etc.) of our model. It is not clear whether the hyperparameters used in our experiment are optimum or not. Due to tuning of hyperparameters, we may expect to higher accuracies.

Acknowledgements

I would like to express my sincere gratitude to constant encouragement and support of my adviser, Associate Professor Nobutaka Suzuki in pursuing this study. Without his guidance and persistent help this thesis would not have been possible. I would like to thank Associate Professor Taro Tezuka and Assistant Professor Kei Wakabayashi. I am also grateful to the members of nslab for their daily supports, suggestions and encouragement.

Bibliography

[1] Kim, S., Yang, S., and Ko, Y. Classifying mathematical expressions written in mathml. *IEICE Transactions on Information and Systems E95.D*, 10 (2012), 2560–2563.

[2] Líška, M., Sojka, P., Růžička, M., and Mravec, P. Web interface and collection for mathematical retrieval: Webmias and mrec. In *Towards a Digital Mathematics Library.* (Bertinoro, Italy, Jul 2011), P. Sojka and T. Bouche, Eds., Masaryk University, pp. 77–84.

[3] Nghiem, M.-Q., Kristianto, G. Y., and Aizawa, A. Using mathml parallel markup corpora for semantic enrichment of mathematical expressions. *IEICE Transactions on Information and Systems E96.D*, 8 (2013), 1707–1715.

[4] Nguyen, T. T., Chang, K., and Hui, S. C. Adaptive two-view online learning for math topic classification. In *Proceedings of the 2012 European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part I* (Berlin, Heidelberg, 2012), ECML PKDD'12, Springer-Verlag, pp. 794–809.

[5] Osedo, R., and Kai, H. Detecting ambiguous formulae using mathematical expression database. In *Proceedings of the 73th National Convention of IPSJ* (mar 2011), vol. 2011, pp. 723–724.

[6] Schubotz, M., Grigorev, A., Leich, M., Cohl, H. S., Meuschke, N., Gipp, B., Youssef, A. S., and Markl, V. Semantification of identifiers in mathematics for better math information retrieval. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2016), SIGIR '16, ACM, pp. 135–144.

[7] Tai, K. S., Socher, R., and Manning, C. D. Improved semantic representations from tree-structured long short-term memory networks. *CoRR abs/1503.00075* (2015).

[8] Wolfram Research. The wolfram functions site, 2017.

[9] Yang, R., Kalnis, P., and Tung, A. K. H. Similarity evaluation on tree-structured data. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2005), SIGMOD '05, ACM, pp. 754–765.