Predictive Algorithm for Converting Linear Strings to General Mathematical Formulae

Tetsuo Fukui(✉) and Shizuka Shirai

Mukogawa Women's University, Nishinomiya, Japan
fukui@mukogawa-u.ac.jp

Abstract. Standard input methods for entering mathematical expressions on digital devices are cumbersome. Our goal is to develop an intelligent mathematical input method that allows users to input mathematical expressions using colloquial linear strings. This paper presents the evaluation of an improved predictive algorithm for converting linear strings into general mathematical formulae. The results of our evaluation show that the prediction accuracy of the top ten ranking for our method is 85.2%.

1 Introduction

In recent years, computer-aided assessment (CAA) systems have been used in mathematics education. However, current mathematical input methods for digital devices are cumbersome for novice learners [1,2].

To reduce the effort required to input mathematical expressions, we propose an interactive conversion input method that uses linear strings in a colloquial style [3,4]. In this method, a list of candidates for the desired mathematical expression is shown in a "what you see is what you get" (WYSIWYG) editor. After all the elements are interactively chosen, the desired expression is formed. We have previously shown that an interface implementing this method is 1.2 to 1.6 times faster than standard interfaces [5]. However, users must convert each element into the appropriate colloquial-style mathematical strings in the correct order from left to right [6].

We have previously addressed this shortcoming and proposed a predictive algorithm [7–9] for converting linear strings into complete mathematical expressions using a perceptron [10]. The results of an evaluation that involved the entry of quadratic polynomials achieved a prediction accuracy of 96.2% for the top-ten ranking. The mean CPU time for predicting each mathematical expression was 0.44 s.

However, it is not clear whether this algorithm results in the same high level of prediction accuracy for more general mathematical formulae. In fact, the results of our previous investigation showed that the prediction accuracy for complicated mathematical formulae decreased to approximately 70% and the prediction time increased remarkably, to more than 6 min in some cases.

This study aims to address these shortcomings by extending the previous algorithm to a wider field of mathematics. We present the improved algorithm

S. Yamamoto (Ed.): HIMI 2017, Part II, LNCS 10274, pp. 15–28, 2017.
DOI: 10.1007/978-3-319-58524-6_2

and the results of its evaluation using a dataset containing 4,000 mathematical formulae. The prediction accuracy of the top-ten ranking for this improved method is 85.2%.

2 Predictive Conversion

In this section, we review our previously proposed predictive conversion system [7–9]. First, we define the linear string of a mathematical expression to be input by the user. We describe an intelligent predictive conversion system of such linear strings in Sect. 2.2. In Sect. 2.3, we formulate a predictive algorithm using machine learning.

2.1 Linear String Rules

The rules for a linear mathematical string for a mathematical expression are as follows:

Definition 1 (Linear String Rules). *Set the key letters (or words) corresponding to the elements of a mathematical expression linearly in the order of the colloquial (read or spoken) style, without considering two-dimensional placement and delimiters.*

In other words, a key letter (or word) consists of the ASCII code(s) corresponding to the initial or clipped form (such as the LaTeX form) of the objective mathematical symbol; a single key often supports many mathematical symbols. For example, when a user wants to input θ^2, the linear string is denoted by "t2," where "t" represents the "theta" symbol. It is unnecessary to include the power sign (i.e., the caret letter (^)). The linear string denoting $\frac{3}{x^2-1}$ is "3/x2−1," where it is not necessary to include the denominator (which is generally the operand of an operator) in parentheses, because they are not printed.

Other representative cases are shown in Table 1. For example, the linear string for $e^{\pi x}$ is denoted by "epx." However, the linear string of the expressions $e_p x$, e^{px}, and $e^{\pi} x$ are also denoted by "epx." Hence, there are some ambiguities when representing mathematical expressions as linear strings using these rules.

2.2 A Predictive Conversion System

In 2015, we proposed a predictive algorithm [7,8] to convert linear string s into the most suitable mathematical expression y_p. For prediction purposes, we devised a method in which each candidate to be selected is ranked by its suitability. Our method uses the function Score(y) to assign a score proportional to the occurrence probability of mathematical expression y, which enables us to predict candidate y_p, using (1), as the most suitable expression with the maximum score. Here, $Y(s)$ in (1) represents all possible mathematical expressions converted from s.

$$y_p \quad \text{s.t.} \quad \text{Score}(y_p) = \max\{\text{Score}(y) | y \in Y(s)\} \tag{1}$$

Table 1. Examples of mathematical expressions using linear string rules.

Category	Linear strings	Math formulae
Variable	t	t or θ
Polynomial	x2 + 2x + 1	$x^2 + 2x + 1$
Fraction	3/4	$\frac{3}{4}$
Equation	(x−3)2 = 0	$(x-3)^2 = 0$
Square root	root2	$\sqrt{2}$
Trigonometric	cos2t	$\cos^2\theta$
Logarithm	log10x	$\log_{10} x$
Exponent	epx	$e^{\pi x}$
Summation	sumk = 1nak	$\sum_{k=1}^{n} a_k$
Integral	intabf(x)dx	$\int_a^b f(x)dx$

Generally, any mathematical expression is represented by a tree structure consisting of nodes and edges, which correspond to the symbols and operating relations, respectively. In other words, any mathematical expression y is characterized by all the nodes and edges included in y. We identify each node or edge as a mathematical element in the formula.

First, all node elements of the mathematical expressions are classified into the nine categories listed in Table 2. Thus, a node element is characterized by (k, e, t), where k is the key letter (or word) of the mathematical symbol e that belongs to type $t(= N, V, P, A, B_L, B_R, C, Q, R,$ or $T)$ in Table 2. For example, the number 12 is characterized as ("12",12, N), the variable a as ("a", a, V), and the Greek letter π can either be characterized as ("pi", π, V) or ("p", π, V). As an example of an operator, ("/", $\frac{\triangle_1}{\triangle_2}, C$) represents a fraction symbol with input character "/", where $\triangle_1$ and $\triangle_2$ represent arbitrary operands.

An edge element, i.e. an operating relation, is characterized by (e_p, i, e_c), where the parent operator e_p operates the i-th operand whose top element is e_c. For example, expression $\frac{\pi}{12}$ consists of three nodes,

$$\{e_1, e_2, e_3\} := \{(\text{“}p\text{”}, \pi, V), (\text{“/”}, \frac{\triangle_1}{\triangle_2}, C), (\text{“12”}, 12, N)\}, \tag{2}$$

and the following two edges:

$$\{(e_2, 1, e_1), (e_2, 2, e_3)\}. \tag{3}$$

In this study, our prototype system implements a total of 509 mathematical symbols and 599 operators in node element table $\mathcal{D}$.

The totality $Y(s)$ of the mathematical expressions converted from s is calculated using Procedures 1–3 (cf. [4,7]), referring to node element table $\mathcal{D}$.

Table 2. Nine types of mathematical expressive structures

Math element type	Type codes	Examples
Number	N	3, 256
Variable, Symbol	V	a, x, α, θ, π
Prefix unary operator	P	$\sqrt{\triangle_1}$, $\sin\triangle_1$
Postfix unary operator	A	$\triangle_1'$, $\triangle_1^\circ$
Bracket	B_L, B_R	$(\triangle_1)$, $\{\triangle_1\}$, $\lvert\triangle_1\rvert$
Infix binary operator	C	$\triangle_1+\triangle_2$, $\triangle_1\times\triangle_2$, $\frac{\triangle_1}{\triangle_2}$
Prefix binary operator	Q	$\log_{\triangle_1}\triangle_2$
Prefix ternary operator	R	$\int_{\triangle_1}^{\triangle_2}\triangle_3$
Infix ternary operator	T	$\triangle_1 \overset{\triangle_2}{\rightarrow} \triangle_3$

$\triangle_1, \triangle_2$, and $\triangle_3$ represent operands.

Procedure 1. A linear string s is separated in the group of keywords defined in (4) using the parser in this system. All possible key separation vectors $(k_1, k_2, \cdots, k_K)$ are obtained by matching every part of s with a key in $\mathcal{D}$.

$$s = k_1 \uplus k_2 \uplus \cdots k_K \text{ where } (k_i, v_i, t_i) \in \mathcal{D}, i = 1, ..., K \tag{4}$$

Procedure 2. Predictive expressive structures are fixed by analyzing all the key separation vectors of s and comparing the nine types of structures in Table 2.

Procedure 3. From the fixed structures corresponding to the operating relations between the nodes, we obtain $Y(s)$ by applying all possible combinations of mathematical elements belonging to each keyword in $\mathcal{D}$.

2.3 Predictive Algorithm

Let us assume that the occurrence probability of a certain mathematical element is proportional to its frequency of use. Then, the occurrence probability of mathematical expression y, which is a possible conversion from string s, is estimated from the total score of all the mathematical elements included in y. Given the numbering of each element from 1 to the total number of elements F_{total}, let θ_f be the score of the $f(=1, \cdots, F_{total})$-th element, and let $x_f(y)$ be the number of times the f-th element is included in y. Then, Score(y) in (1) is estimated by (5), where $\boldsymbol{\theta}^T = (\theta_1, \cdots, \theta_{F_{total}})$ denotes the score vector and $\boldsymbol{X} = (x_f(y)),\ f = 1, \cdots, F_{total}$ is an F_{total}-dimensional vector.

$$h_\theta\left(\boldsymbol{X}(y)\right) = \boldsymbol{\theta}^T \cdot \boldsymbol{X}(y) = \sum_{f=1}^{F_{total}} \theta_f x_f(y) \tag{5}$$

Equation (5) is in agreement with the hypothesis function of linear regression and $\boldsymbol{X}(y)$ is referred to as the characteristic vector of y. To solve our linear regression

problem and predict the occurrence probability of a mathematical expression, we conduct supervised machine learning on the m elements of training dataset $\{(s_1, y_1), (s_2, y_2), \cdots, (s_m, y_m)\}$. To obtain the optimized score vector, our learning algorithm utilizes the following four-step procedure:

Step 1. Initialization $\boldsymbol{\theta} = \mathbf{0}$, $i = 1$
Step 2. Decision regarding a candidate.

$$y_p \ \text{ s.t. } h_\theta\left(\boldsymbol{X}(y_p)\right) = \max\{h_\theta\left(\boldsymbol{X}(y)\right) | y \in Y(s_i)\} \tag{6}$$

Step 3. Training parameter.
if$(y_p \neq y_i)$ {

$$\begin{aligned} \text{if}(\theta_f < S_{\max})\{\theta_f &:= \theta_f + 2 \ \text{ for } \ \{f \leq F_{total} | x_f(y_i) > 0\}\} \\ \theta_{\bar{f}} &:= \theta_{\bar{f}} - 1 \ \text{ for } \ \{\bar{f} \leq F_{total} | x_{\bar{f}}(y_p) > 0\} \end{aligned} \tag{7}$$

}

Step 4. if$(i < m)\{$ $i = i + 1$; repeat from **Step 2**$\}$
else { Output $\boldsymbol{\theta}$ and end}

This learning algorithm is simple and is similar to a structured perceptron used for natural language processing (NLP) [10]. However, in our previous study [7], we revised the increase weight from one to two in (7) to avoid negative score learning. When two different candidates belonging to the same key appear in the training dataset, e.g., the pair a and α, their scores change into a positive value from a negative value or vice versa; even if a candidate with a negative score has occurred many times, it has lower priority than one with a score of zero. Here, $S_{\max}$ in (7) is a suitable upper bound for any mathematical element score, preventing the score parameter from continuing to increase as the algorithm runs.

3 Main Algorithm

In previous investigations [7–9] of the learning algorithm in Sect. 2.3, limiting the entered expressions to quadratic polynomials resulted in a prediction accuracy of 96.2% for the top-ten ranking. The mean CPU time for predicting each mathematical expression was 0.44 s.

However, the algorithm in Sect. 2.3 did not provide the same high performance given more general mathematical formulae. In fact, the results of our investigation using a dataset of 4000 math formulae from broader fields of mathematics (cf. Sect. 4.1) showed that prediction accuracy decreased to approximately 70% and prediction time increased considerably; for a complicated mathematical formula, prediction took more than 6 min.

As the reason for the decrease in prediction accuracy and increase in prediction time, we found the following three challenges in predicting general mathematical expressions.

1. Scores can increase based on the number of elements, instead of their priority.
2. If $Y(s)$ includes the same mathematical expressions, but has different internal tree structures, machine learning does not work well.
3. The number of complicated mathematical expressions, which lead to long prediction times, increases when general mathematics fields are considered.

To overcome these shortcomings, we have revised the weight calculations of the score in Sect. 3.1, introduce a normal form for mathematical tree expressions in Sect. 3.2, and improved the search routine, breaking it off after ten seconds that is described in Sect. 3.3.

3.1 Balancing Scores with the Key Character Length

An expression' score can increase based on the number of its elements, instead of their priority. For example, the score of $sinx(= s \otimes i \otimes n \otimes x)$ is higher than the score of $\sin x$ because $sinx$ includes 7 nodes and 6 edges, compared to the 3 elements in $\sin x$. (Here, in this paper, the symbol $\otimes$ is used for recollection of an invisible multiplication that is also treated as an operator or node element.)

Generally, when $Y(s)$ from linear string s with character length n is obtained by Procedures 1–3, s can be decomposed into a key separation vector with n elements, per (4), because the character keys of almost all ASCII codes are registered in our system's key dictionary $\mathcal{D}$.

For example, if s = "pi", there exist two key separation vectors ("pi") and ("p","i") from s, and the totality of candidates $Y(\text{“}pi\text{”})$ is estimated to be

$$Y(\text{“}pi\text{”}) = \{\pi, pi, p^i, p_i, \cdots\}. \tag{8}$$

The score of the first candidate in (8) is estimated only by parameter θ_π, whereas the score of the second candidate $pi(= p \otimes i)$ is summed among the five parameters as

$$h_\theta\left(\boldsymbol{X}(pi)\right) = \theta_p + \theta_\otimes + \theta_i + \theta_{(\otimes,1,p)} + \theta_{(\otimes,2,i)}. \tag{9}$$

Therefore, h_θ does not become proportional to the occurrence probability of such a mathematical expression, because its score can increase according to its number of elements if the value of each score parameter has the same degree. This shortcoming also occurs for multidigit figures (e.g., 123) and some operators (e.g., sin).

To overcome this issue, we revise the score's weight calculations by adding a suitable weight to the score parameters depending on the length of key characters. Let K be the length of linear string s when a candidate expression $y_p(s)$ is formed from s. The tree structure of a product of K variables consists of $2K-1$ nodes and $2(K-1)$ edges. In this study, we have adopted the following score functions (10–12) for numbers N, variables/symbols V, and operators $\mathcal{O}$ onto $m(= 1, 2, 3)$ numbers of operands as a weight balance, respectively.

$$\text{Score}(N) = \text{Len}(N) + 3S_{\max}\{\text{Len}(N) - 1\} \tag{10}$$

$$\mathrm{Score}(V) = \theta_V \times \mathrm{Len}(V) + 3S_{\max}\{\mathrm{Len}(V) - 1\} \tag{11}$$

$$\mathrm{Score}(\mathcal{O}) = \theta_{\mathcal{O}} \times \mathrm{Len}(\mathcal{O}) + 3S_{\max}\{\mathrm{Len}(\mathcal{O}) - 1\} + 2mS_{\max} \tag{12}$$

Here, $\mathrm{Len}(e)$ stands the length of the key string of element e, and θ_V and $\theta_{\mathcal{O}}$ are the score parameters for nodes V and $\mathcal{O}$, respectively. Any score parameter $\theta_f \leqq S_{\max}$, as described in (7). Using (8), $\mathrm{Score}(\pi) = 2\theta_\pi + 3S_{\max}$, per (11), and if $\theta_\pi \approx \theta_p \approx \theta_i$, then $\mathrm{Score}(\pi) \geqq h_\theta\,(\boldsymbol{X}(pi))$.

Therefore, we propose to revise the algorithm in **Steps 1**–**4**, by altering **Step 2** to

$$y_p \ \text{ s.t. } \mathrm{Score}(y_p) = \max\{\mathrm{Score}(y) | y \in Y(s_i)\}. \tag{13}$$

3.2 Unique Normal Form for Mathematical Expressions

There exist innumerable tree representations of mathematical expression with the same notation but different internal structures. For example, $a + b + c$ is denoted without any parentheses, given the associativity of addition (14). However, it can be represented by two different tree structures, shown in Fig. 1.

$$(a + b) + c = a + (b + c) \tag{14}$$

This becomes a challenge in the machine learning stage; in **Step 3** in Sect. 2.3, candidate y_p is judged to be different from the correct formula y_i, despite the same notation in both expressions. Thus, our system cannot definitively predict the desired mathematical expression.

To overcome this shortcoming, we define a normal form for mathematical notation.

Definition 2 (Normal form for mathematical notation). *A normal form for a mathematical notation ensures that all trees based on a mathematical expression with the same notation are recorded uniquely.*

Normal form was discussed using "term rewriting system" by Knuth and Bendix [11] in 1967 and a unique normal form in computer algebra systems was discussed in [12], in particular, that there exists a unique normal form for polynomial systems. We proposed a normal form for mathematical notation to use

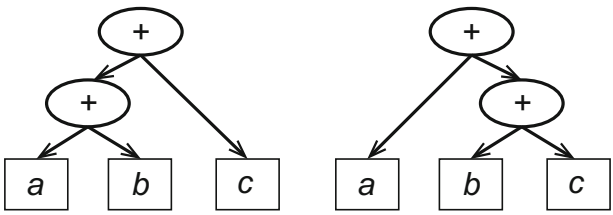

Fig. 1. The different tree structures of $a + b + c$

in a math input interface in [4], in which we concretely define the normal form for mathematical notation to implement into our prediction system for general mathematical formulae.

First, we define the order for all formulae per algebraic rules in Table 3 to prescribe the unique normal form for mathematical notation. The definition for all elements that constitute a formula using Buchus' normal form (BNF) in Table 3 stands for a recursive definition from a elements of low rank to those of high rank. The order of operators are conducted as in (15), per algebraic rules.

$$\otimes < \oplus < \text{Comparison op.} < \text{Relational op.} < \text{Multiple op.} \tag{15}$$

Namely, any formula can be a high-rank multiple form ⟨Multiple⟩. Here, $\otimes$ stands for an invisible multiplication, $\oplus$ for an addition-type binary operator ($+$ or $-$) and $\pm$ for a prefix unary sign operator ($+$,$-$ or $\pm$). Let ⟨Term⟩ and ⟨Factor⟩ be an algebraic term (e.g., xy^2z) and a factor (e.g., x, y^2, z), respectively. As prescribed by Table 3, the right-hand side of an invisible multiplication $\otimes$ can be a ⟨Factor⟩ and a factor ⟨Right factor⟩ prefixed by sin, lim, log, $\sum$, $\int$, etc., can be arranged on the right side of $\otimes$ but is forbidden on the left side of $\otimes$. For example, $\int f(x)dx$ is a ⟨Right factor⟩ because if the factor y was arranged on the right side of such an integral, y could not be distinguished from an integral calculus variable. Similarly, a ⟨Left factor⟩ like a signed factor (e.g., $-x$) is forbidden on the right-hand side of $\otimes$.

For example, the tree structure on the right in Fig. 1, which is formed as ⟨Factor⟩ $\otimes$ ⟨Term⟩, is excluded because such a form is not included in the definition equation for ⟨Term⟩ in Table 3.

Therefore, we treat only the normal form defined by Table 3 for mathematical tree expressions in this study. We converted the 4000 math formulae in the training dataset to the normal form and use only the normal form to predict

Table 3. The definition equation for the normal form of mathematical formulae using BNF

⟨Number⟩	::=	⟨Value⟩ | ⟨Value⟩.⟨Value⟩;
⟨Non number symbol⟩	::=	⟨Symbol⟩ | ⟨Text⟩ | ⟨$\widehat{\text{Symbol}}$⟩ | ⟨$\dot{\text{Symbol}}$⟩ | ⟨$\widetilde{\text{Symbol}}$⟩ | ⟨$\overrightarrow{\text{Symbol}}$⟩;
⟨Factor⟩	::=	⟨Non number symbol⟩ | (⟨Factor⟩⟨Postfix unary op.⟩) | (⟨Bracket⟩⟨Multiple⟩) | $\sqrt{\langle\text{Polynomial}\rangle}$ | ⟨Factor⟩$^{\langle\text{Polynomial}\rangle}$ | ⟨Factor⟩$_{\langle\text{Polynomial}\rangle}$ | $\frac{\langle\text{Polynomial}\rangle}{\langle\text{Polynomial}\rangle}$;
⟨Term⟩	::=	⟨Number⟩ | ⟨Factor⟩ | ⟨Term⟩ $\otimes$ ⟨Factor⟩;
⟨Right factor⟩	::=	(⟨Prefix op.⟩⟨Expressions⟩);
⟨Right term⟩	::=	⟨Right factor⟩ | ⟨Term⟩ $\otimes$ ⟨Right factor⟩ | ⟨Right term⟩ $\otimes$ ⟨Right factor⟩;
⟨Left factor⟩	::=	$\pm$⟨Factor⟩ | $\pm$⟨Left factor⟩;
⟨Left term⟩	::=	⟨Left factor⟩ | ⟨Left term⟩ $\otimes$ ⟨Factor⟩;
⟨Left-right term⟩	::=	$\pm$⟨Right factor⟩ | ⟨Left term⟩ $\otimes$ ⟨Right factor⟩ | ⟨Left-right term⟩ $\otimes$ ⟨Right factor⟩;
⟨Polynomial⟩	::=	⟨Term⟩ | ⟨Right term⟩ | ⟨Left term⟩ | ⟨Left-right term⟩ | ⟨Polynomial⟩ $\oplus$ ⟨Term⟩ | ⟨Polynomial⟩ $\oplus$ ⟨Right term⟩;
⟨Comparable⟩	::=	⟨Polynomial⟩ | (⟨Comparable⟩⟨Comparable op.⟩⟨Polynomial⟩);
⟨Relational⟩	::=	⟨Comparable⟩ | (⟨Relational⟩⟨Relational op.⟩⟨Comparable⟩);
⟨Multiple⟩	::=	⟨Relational⟩ | ⟨Matrix⟩ | (⟨Multiple⟩, ⟨Relational⟩) | (⟨Multiple⟩⟨Space⟩⟨Relational⟩);

the N-best candidates from a linear string using the algorithm proposed in this section.

3.3 Complexity of Candidate Math Expressions and Calculation Time

Generally, the number of elements in $Y(s)$, denoted by $n(Y(s))$, increases rapidly corresponding to the increase in the length of s. For example, because the key character "a" corresponds to seven symbols ($Y(\text{“}a\text{”}) = \{a, \alpha, \mathrm{a}, \mathbf{a}, \boldsymbol{a}, \mathtt{a}, \aleph\}$) and the invisible multiplication between a and b corresponds to $Y(\text{“}ab\text{”}) = \{ab, a^b, a_b, {}^a b, {}_a b\}$, then $n(Y(\text{“}abc\text{”})) = 7^3 \times 5^2 = 8575$. However, for a mathematical input interface, it is sufficient to calculate the N-best high score candidates in $Y(s)$ as shown in (1).

Therefore, we improve the search routine, breaking it off after ten seconds, because the mean runtime for cases in which the length of s was less than 16 required less than 10 s (cf. Sect. 4.3). To improve the efficiency of calculation, we obtain the N-best candidates in $Y(s)$ as follows:

1. In Procedure 1, all the key separation vectors $(k_1, k_2, \cdots, k_K)$ of s are sorted in ascending order of K in (4), i.e., in order starting from higher probability.
2. In Procedure 2, we set an upper limit $T = 10$ s for breaking down all possible calculations of the predictive expressive structures.
3. In Procedure 3, to obtain the N-best candidates in $Y(s)$, we apply only the N-best mathematical elements for operand expressions related to an operator, instead of all possible combinations.

4 Experimental Evaluation

In this section, we experimentally investigate the prediction accuracy of the algorithm described in the previous section. Then, we present the results of this evaluation in Sect. 4.2 and discuss the results of this study in Sect. 4.3.

4.1 Method

We examined prediction accuracy using a dataset of 4000 mathematical formulae $\mathcal{E} = \{(s_i, y_i) | i = 0, \cdots, 3999\}$ from a five-volume mathematics textbook [13], organized into the following categories: algebra, geometry, vector algebra, sets/logic, numbers, probability/statistics, trigonometry, exponents/logarithms, sequences, limits, and calculus, which are studied in the tenth through twelfth grades in Japan. Dataset $\mathcal{E}$ was generated manually with our previous system [14] in the order of appearance in the textbook by choosing individual expressions y with the corresponding linear string s, the length of which is less than 16.

In the experimental evaluation, we measured the proportion of correct predictions for 500 test datasets after learning the parameters using the predictive algorithm described in Sect. 3 on a training dataset consisting of 3500 formulae by 8-fold cross-validation.

4.2 Results

The machine learning results for our predictive algorithm are given in Table 4 for various training set sizes. Figure 2 illustrates this result and shows that 3500 training data are sufficient for performing our machine learning algorithm.

The accuracy of "Best 1" with our predictive algorithm was approximately 51.5% after being trained 3500 times and that of "Best 3" was 72.4%. However, for its top-ten ranking, this algorithm achieved an accuracy of 85.2%.

The mean CPU time for predicting each mathematical expression with corresponding linear string of length less than 16 was 2.85 s ($SD = 0.16$).

The search ratio omitting the best score was only 0.6%; nevertheless, the calculation break ratio was 16.5%, which indicates that the improvements described in Sect. 3.3 worked well for finding the N-best candidates.

Table 4. Prediction accuracy using our predictive algorithm (%)

Training no.	Best 1	Best 3	Best 10
0	16.0 (0.7)	20.2 (0.9)	23.8 (1.5)
500	36.4 (2.9)	49.9 (5.2)	61.3 (7.9)
1000	45.3 (2.6)	63.6 (1.8)	77.2 (1.9)
1500	47.5 (4.2)	67.4 (2.6)	80.7 (1.6)
2000	49.9 (3.3)	69.7 (2.2)	82.7 (1.6)
2500	51.6 (4.5)	71.2 (3.6)	84.0 (1.7)
3000	50.9 (3.9)	71.5 (3.3)	84.0 (1.7)
3500	51.5 (3.1)	72.4 (2.2)	85.2 (1.8)

Numbers in parentheses denote *SD*.

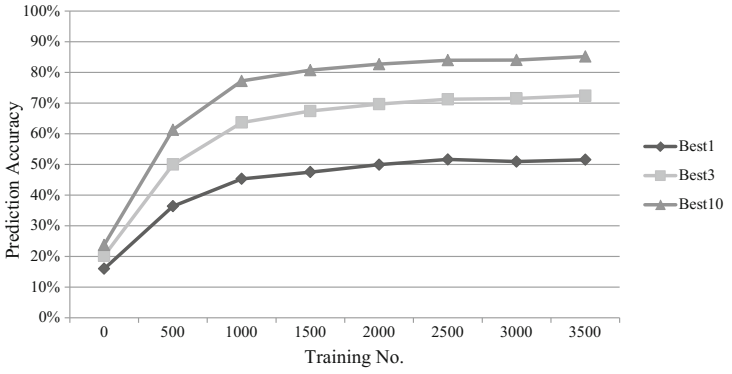

Fig. 2. Prediction results for varying numbers of training examples

4.3 Discussion

Analysis on Length of *s*. The 4000 test data consist of approximately 66% of the mathematical expressions included in a five-volume textbook [13]. There is a negative association ($R^2 = 0.83$) between the length of s and prediction accuracy for top-ten rankings (Fig. 3). The mean CPU time to obtain y_p from various lengths of string s is illustrated in Fig. 4. The mean runtime when the length of s is less than 16 required less than 10 s, although when the length is greater than 10, the mean runtime increases rapidly, because the complexity of candidate math expression increases with the length of string s (c.f. Sect. 3.3). Therefore, this algorithm is insufficient for predicting the N-best candidates from strings whose length is greater than 16. In this case, we need another strategy, like a predictive conversion limited from a part of s.

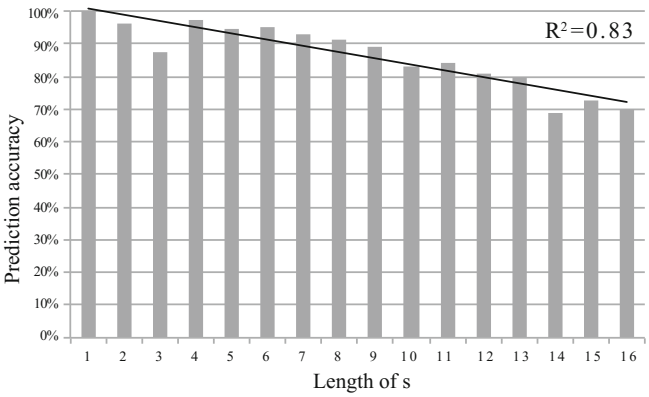

Fig. 3. Length of s effects prediction accuracy

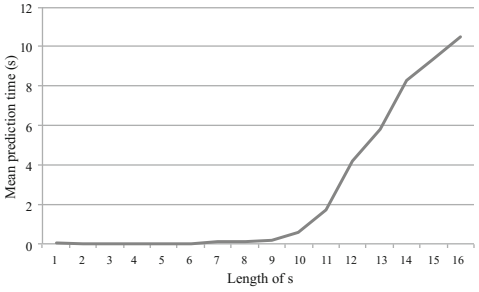

Fig. 4. CPU time for varying lengths of s

Differences Across Mathematics Fields. Figure 5 shows the difference in projection accuracy for each field of mathematics after learning with 3500 training data. The line indicates the data distribution of each field. The prediction accuracies for vector algebra and trigonometry are greater than 90% for the top-ten ranking, but are low for exponents and sequences because mathematical expressions, such as recursive formulae can be complex.

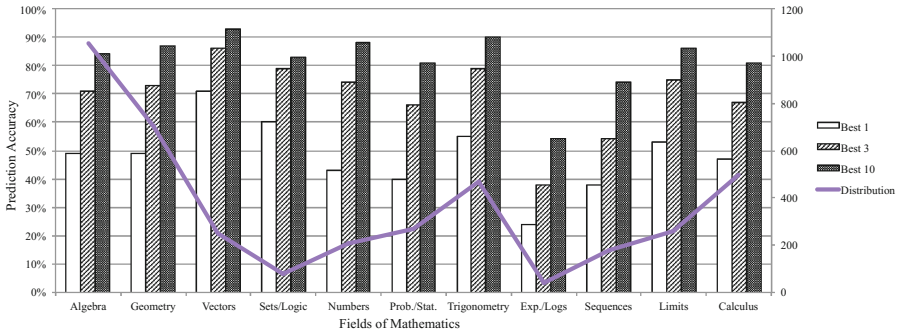

Fig. 5. The difference in projection accuracy for each field of mathematics

5 Related Works

In this section, we describe related works on NLP, along with other predictive inputs for mathematical formulae that use an N-gram model.

Input-word prediction has been studied in NLP since the 1980s, often to predict input characters for a word unit [15]. Structured perceptrons have been used in NLP to input Japanese characters since the 1990s. As explained in Sect. 2, our predictive algorithm uses a structured perceptron; however, mathematical formulae have tree structures, rather than the sentential chain structures of natural language. Indeed, none of the aforementioned methods consider sentence structures, although our method considers the tree structure of mathematical formulae.

Structure-based user interfaces for inputting mathematical formulae are popular. They enable users to format a desired mathematical formula on a PC in a WYSIWYG manner by selecting an icon corresponding to the structure of the expression. Users do so using a GUI template, e.g., a fraction bar or an exponent form, into which the mathematical elements can be entered. Hijikata et al. improved the input efficiency of mathematical formulae by proposing an algorithm for predicting mathematical elements using an N-gram model [16]. However, their proposal is still structure-based in the sense that users must understand the entire structure of a desired mathematical formula before selecting the corresponding icons.

In contrast, our predictive conversion method predicts mathematical structures from a linear string in a colloquial style, separating it from structure-based input methods.

6 Conclusion and Future Work

In this study, we proposed a predictive algorithm for linear string conversion to the N-best candidate mathematical formulae, with an accuracy of 85.2% for the top-ten ranking, by improving upon a previously proposed structured perceptron algorithm to apply to general categories of mathematics. The mean CPU time for predicting each mathematical expression with a corresponding linear string of length less than 16 (obtained from a five-volume mathematics textbook) was 2.84 s.

The most important avenues for future research are to reduce prediction time and to develop an intelligent mathematical input interface by implementing our proposed predictive algorithm.

This work was supported by JSPS KAKENHI Grant Number 26330413.

References

1. Pollanen, M., Wisniewski, T., Yu, X.: Xpress: a novice interface for the real-time communication of mathematical expressions. In: Proceedings of the Workshop on Mathematical User Interfaces (2007)
2. Sangwin, C.: Computer aided assessment of mathematics using STACK. In: Cho, S.J. (ed.) Selected Regular Lectures from the 12th International Congress on Mathematical Education, pp. 695–713. Springer, Cham (2015). doi:10.1007/978-3-319-17187-6_39
3. Fukui, T.: An intelligent method of interactive user interface for digitalized mathematical expressions. RIMS Kokyuroku **1780**, 160–171 (2012). (in Japanese)
4. Fukui, T.: The performance of interactive user interface for digitalized mathematical expressions using an intelligent formatting from linear strings. RIMS Kokyuroku **1785**, 32–44 (2012). (in Japanese)
5. Shirai, S., Nakamura, Y., Fukui, T.: An interactive math input method for computer aided assessment systems in mathematics. IPSJ Trans. Comput. Educ. **1**(3), 11–21 (2015). (in Japanese)
6. Shirai, S., Fukui, T.: Improvement in the input of mathematical formulae into STACK using interactive methodology. Comput. Educ. **37**, 85–90 (2014). (in Japanese)
7. Fukui, T.: Prediction for converting linear strings to mathematical formulae using machine learning. In: Proceedings of ARG WI2, vol. 6, pp. 67–72 (2015). (in Japanese)
8. Fukui, T., Shirai, S.: Predictive algorithm from linear string to mathematical formulae for math input method. In: Proceedings of the 21st Conference on Applications of Computer Algebra, pp. 17–22 (2015)
9. Shirai, S., Fukui, T.: Evaluation of a predictive algorithm for converting linear strings to mathematical formulae for an input method. In: Kotsireas, I.S., Rump, S.M., Yap, C.K. (eds.) MACIS 2015. LNCS, vol. 9582, pp. 421–425. Springer, Cham (2016). doi:10.1007/978-3-319-32859-1_36
10. Manning, C.D., Scheutze, H.: Foundations of Statistical Natural Language Processing. The MIT Press, London (2012)
11. Knuth, D.E., Bendix, P.B.: Simple word problems in universal algebra. In: Proceedings of Oxford, vol. 67, pp. 263–298 (1967)

12. Sasaki, T., Motoyoshi, F., Watanabe, S.: Computer Algebra System, vol. 36, Shoukoudo (1986). (in Japanese)
13. Matano, H., et al.: Vols. I301, A301, II301, B301, and III301 of Mathematics, Japan. Tokyo Shoseki (2014). (in Japanese)
14. Shirai, S., Fukui, T.: MathTOUCH: mathematical input interface for e-assessment systems. MSOR Connections **15**(2), 70–75 (2016)
15. Garay-Vitoria, N., Abascal, J.: Text prediction systems: a survey. Univers. Access Inf. Soc. **4**(3), 188–203 (2006)
16. Hijikata, Y., Horie, K., Nishida, S.: Predictive input interface of mathematical formulas. In: Kotzé, P., Marsden, G., Lindgaard, G., Wesson, J., Winckler, M. (eds.) INTERACT 2013. LNCS, vol. 8117, pp. 383–400. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40483-2_27

http://www.springer.com/978-3-319-58523-9

Human Interface and the Management of Information: Supporting Learning, Decision-Making and Collaboration
19th International Conference, HCI International 2017, Vancouver, BC, Canada, July 9–14, 2017, Proceedings, Part II
Yamamoto, S. (Ed.)
2017, XXV, 636 p. 321 illus., Softcover
ISBN: 978-3-319-58523-9