Algorithm for Predicting Mathematical Formulae from Linear Strings for Mathematical Inputs

Tetsuo Fukui

Abstract Recently, computer-aided assessment (CAA) systems have been used for mathematics education, with some CAA systems capable of assessing learners' answers using mathematical expressions. However, the standard input method for mathematics education systems is cumbersome for novice learners. In 2011, we proposed a new mathematical input method that allowed users to input mathematical expressions through an interactive conversion of mathematical expressions from colloquial-style linear strings in WYSIWYG. In this study, we propose a predictive algorithm to improve the input efficiency of this conversion process by using machine learning to determine the score parameters with a structured perceptron similar to natural language processing. In our experimental evaluation, with a training dataset comprising 700 formulae, the prediction accuracy was 96.2% for the top ten ranking by stable score parameter learning; this accuracy is sufficient for a mathematical input interface system.

Keywords Mathematical input interface · Predictive algorithm · Machine learning · Mathematical formula editor

1 Introduction

In recent years, computer-aided assessment (CAA) systems have been used for the purpose of mathematics education. Some CAA systems enable users to directly enter mathematical expressions such that their answers can be evaluated automatically by using a computer algebra system (CAS). These systems have also been used to provide instructions to students at universities. However, the procedure through which answers are entered into the system, using a standard input method for mathematics education, is still cumbersome [10, 11].

T. Fukui (✉)
Mukogawa Women's University, Nishinomiya, Japan
e-mail: fukui@mukogawa-u.ac.jp

I.S. Kotsireas and E. Martínez-Moro (eds.), *Applications of Computer Algebra*, Springer Proceedings in Mathematics & Statistics 198,
DOI 10.1007/978-3-319-56932-1_9

In 2011, we proposed a new mathematical input method through the conversion of colloquial-style mathematical text (string) [1, 3]. This method is similar to those used for inputting Japanese characters in many operating systems. In this system, the list of candidate characters and symbols corresponding to the desired mathematical expression, as obtained through the user input, is displayed in WYSIWYG format; once all elements required to be included by the user are selected, the process of formatting of the expressions is complete. This method enables the user to input almost any mathematical expression without having to learn a new language or syntax [12]. However, the disadvantage of the above-mentioned method is that the user has to convert each element in the colloquial-style mathematical string proceeding from left to right in order [13].

This study aims to address this shortcoming by improving the input efficiency of such systems through intelligent predictive conversion of a linear mathematical string to an entire expression instead of converting each element individually.

2 Related Works

In this section, we describe related works on natural language processing, along with other predictive inputs for mathematical formulae using an N-gram model.

Input-word prediction has been studied since the 1980s in the field of natural language processing. Input characters are usually predicted for a word unit [5]. An N-gram model is typically used to predict text entries in popular probabilistic language models. For example, one typical system for word prediction, Reactive Keyboard, uses an N-gram model for augmentative and alternative communication (AAC) [7]. In such systems, a tree is built for prediction, where each alphabetical character corresponds to a node. Priority is assigned to each node based on the number of occurrences in the N-gram. When a user inputs characters, the system matches them with tree nodes, and the words in the child nodes of the matched node are provided as proposed predictions. A structured perceptron in machine learning for natural language processing has been used to input Japanese characters since the 1990s. As explained in Sect. 4.1, Algorithm 1 is similar to machine learning. It uses a structured perceptron for natural language processing [9]. However, mathematical formulae have tree structures, rather than the sentential chain structures of natural language. Indeed, none of the above-mentioned methods consider the structure of a sentence; however, our method considers the 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 WYSIWYG by selecting an icon corresponding to the structure of the expression. User do so using a GUI template, e.g., a fraction bar and an exponent form, into which the mathematical elements can be entered. Hijikata et al. from Osaka University improved the input efficiency of mathematical formulae by proposing an algorithm for predicting mathematical elements using an N-gram model [6]. However, their proposal is nevertheless a structure-based interface in the sense that users must understand the entire structure of a desired mathematical formula before selecting the corresponding icons.

By contrast, our predictive conversion method predicts such mathematical structures from a linear string of the mathematical formulae, rendering it significantly different from structure-based input methods.

3 Predictive Conversion

In this section, we define the linear string of a mathematical expression to be input by the user and describe the design of an intelligent predictive conversion system of such linear strings in Sect. 3.2. In Sect. 3.3, we formulate a predictive algorithm by using machine learning.

3.1 Linear String Rules

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

Set the key letters (or words) corresponding to the elements of a mathematical expression linearly in the order of the colloquial (or reading) 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 the clipped form (such as the LaTeX -form) of the objective mathematical symbol. Therefore, a single key often supports many mathematical symbols. For example, when a user wants to input α^2, the linear string is denoted by "a2", where "a" represents the "alpha" symbol and it is unnecessary to include the power sign (i.e., the caret letter (ˆ)). In the case of $\frac{1}{\alpha^2+3}$, the linear string is denoted by "1/a2 + 3", where it is not necessary to put the denominator (which is generally the operand of an operator) in parentheses, because those are never printed.

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

3.2 Design of an Intelligent Predictive Conversion System

In this paper, we propose a predictive algorithm to convert a linear string s into the most suitable mathematical expression y_p. For prediction purposes, we devise a method through which each candidate to be selected would be ranked in terms of

Table 1 Examples of mathematical expressions using linear string rules

Category	Linear strings	Math formulae
Variable	*a*	a or α
Polynomial	3x2+4x+1	$3x^2 + 4x + 1$
Fraction	2/3	$\frac{2}{3}$
Equation	(x–1/2)2=x2–x+1/4	$(x - \frac{1}{2})^2 = x^2 - x + \frac{1}{4}$
Square root	root3	$\sqrt{3}$
Trigonometric	sin2x	$\sin^2 x$
Logarithm	log10x	$\log_{10} x$
Exponent	epx	$e^{\pi x}$
Summation	sumk=1nk2	$\sum_{k=1}^{n} k^2$
Integral	intabfdx	$\int_a^b f dx$

its suitability. Our method uses a function $\mathrm{Score}(y)$ to assign a score proportional to the probability of occurrence of the mathematical expression y, which would enable us to predict the candidate y_p by using Eq. (1) as being the most suitable expression with the maximum score. Here, $Y(s)$ in Eq. (1) represents the totality of all possible mathematical expressions converted from s.

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

A mathematical expression consists of mathematical symbols, such as numbers, variables, and *operators*,[1] together with the operating relations between an operator and an element. Therefore, we decided to represent a mathematical expression by a tree structure consisting of nodes and edges corresponding to the symbols and operating relations, respectively.

First, all node elements of the mathematical expressions are classified into nine categories, as listed in Table 2 in this mathematical conversion system. Therefore, 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 2 is characterized as ("2",2,N) and similarly a variable x as ("x", x, V) and as for the Greek letter α, it can either be characterized as ("$alpha$", α, V) or ("a", α, V). In the case of an operator, the character ("/", $\frac{\triangle_1}{\triangle_2}$, C) represents a fractional symbol with input key "/", where $\triangle_1$, $\triangle_2$ represents arbitrary operands.

In this study, a total of 510 mathematical symbols and 597 operators in node element table $\mathscr{D}$ are implemented by our prototype system.

[1]In this article, "operator" is used in the sense of operating on, i.e. performing actions on elements in terms of their arrangements for two-dimensional mathematical notation.

Table 2 Nine types of mathematical expressive structures

Math element	Codes of type	Examples ($\triangle_1, \triangle_2, \triangle_3$ represent operands)
Number	N	2, 128
Variable, symbol	V	x, α
Prefix unary operator	P	$\sqrt{\triangle_1}$, $\sin\triangle_1$
Postfix unary operator	A	$\triangle_1'$
Bracket	B_L, B_R	$(\triangle_1)$
Infix binary operator	C	$\triangle_1 + \triangle_2$, $\frac{\triangle_1}{\triangle_2}$
Prefix binary operator	Q	$\log_{\triangle_1}\triangle_2$
Prefix ternary operator	R	$\int_{\triangle_1}^{\triangle_2}\triangle_3$
Infix ternary operator	T	$\triangle_1 \overset{\triangle_2}{\rightarrow} \triangle_3$

The totality $Y(s)$ of the mathematical expressions converted from s is calculated by using the following procedure **Proc. 1–Proc. 3** (cf. [2, 4]) referring to node element table $\mathscr{D}$.

Proc. 1 A linear string s is separated in the group of keywords defined in Eq. (2) by 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 $\mathscr{D}$.

$$s = k_1 \uplus k_2 \uplus \cdots k_K \text{ where } (k_i, v_i, t_i) \in \mathscr{D}, i = 1, ..., K \tag{2}$$

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

Proc. 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 $\mathscr{D}$.

Complexity of $Y(s)$

Generally, the number of elements in $Y(s)$, denoted by $n(Y(s))$, becomes enormous corresponding to the increase in the length of s. For example, because the key letter "a" corresponds to seven symbols, namely $Y(\text{"}a\text{"}) = \{a, \alpha, \mathrm{a}, \mathbf{a}, \boldsymbol{a}, \mathsf{a}, \aleph\}$, and the invisible times between a and b corresponds to $Y(\text{"}ab\text{"}) = \{ab, a^b, a_b, {}^ab, {}_ab\}$, then $n(Y(\text{"}abc\text{"})) = 7^3 \times 5^2 = 8575$. However, for the purpose of a mathematical input interface, it is enough to calculate the N-best high score candidates in $Y(s)$ as shown in Eq. (1). Therefore, for improving the efficiency of calculations we obtain the N-best candidates in $Y(s)$ as follows:

1. In **Proc. 1**, all the key separation vectors $(k_1, k_2, \cdots, k_K)$ of s are sorted in ascending order of the number K in Eq. (2), i.e. in an order starting from higher probability.
2. In **Proc. 2**, we set upper limit L of the number of loops for breaking down all the possible calculations of the predictive expressive structures.

3. In **Proc. 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.

3.3 Predictive Algorithm

Let us assume that the probability of occurrence of a certain mathematical element is proportional to its frequency of use. Then, the probability of occurrence of mathematical expression y, which is possibly converted from a given 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 F_{total}, which is the total number of elements, 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 Eq. (1) is estimated by Eq. (3), where $\boldsymbol{\theta}^T = (\theta_1, \cdots, \theta_{F_{total}})$ denotes the score vector and $\mathbf{X} = (x_f(y)),\ f = 1, \cdots, F_{total}$ is the F_{total}-dimensional vector.

$$h_\theta\left(\mathbf{X}(y)\right) = \boldsymbol{\theta}^T \cdot \mathbf{X}(y) = \sum_{f=1}^{F_{total}} \theta_f x_f(y) \tag{3}$$

Equation (3) is in agreement with the hypothesis function of linear regression, and $\mathbf{X}(y)$ is referred to as the characteristic vector of y. To solve our linear regression problem and predict the probability of occurrence of a mathematical expression, we conduct supervised machine learning on the m elements of a training dataset $\{(s_1, y_1), (s_2, y_2), \cdots, (s_m, y_m)\}$. Our learning algorithm to obtain the optimized score vector is performed through the following four-step procedure:

Step 1 Initialization: $\boldsymbol{\theta} = \mathbf{0}$, $i = 1$
Step 2 Decision regarding a candidate: y_p s.t. $h_\theta\left(\mathbf{X}(y_p)\right) = \max\{h_\theta\left(\mathbf{X}(y)\right) | y \in Y(s_i)\}$
Step 3 Training parameter: if($y_p \neq y_i$) {

$$\begin{array}{ll} \theta_f := \theta_f + 1 & \text{for } \{f \leq F_{total} | x_f(y_i) > 0\} \\ \theta_{\bar{f}} := \theta_{\bar{f}} - 1 & \text{for } \{\bar{f} \leq F_{total} | x_{\bar{f}}(y_p) > 0\} \end{array} \tag{4}$$

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

This learning algorithm is very simple, and similar to machine learning using a structured perceptron for natural language processing [9].

4 Main Algorithm

In this section, we experimentally investigate the prediction accuracy by using the algorithm described in the previous section. Then, we discuss the results of the evaluation in Sect. 4.1 and propose the main algorithm of this study in Sect. 4.2.

4.1 *Experimental Evaluation*

We examine the prediction accuracy using two score learning parameter sets on an evaluation dataset $\mathscr{E} = \{(s_i, y_i) | i = 0, \ldots, 799\}$ containing 800 mathematical formulae from a mathematics textbook [8]. As the scope of the evaluation dataset $\mathscr{E}$, we adopted the mathematical subjects: "Quadratic-polynomials, -equations, -inequalities and -functions" that are studied in the tenth grade in Japan. The dataset $\mathscr{E}$ has generated manually with our previous system [3] in the order of appearance from the textbook by choosing individual expressions y_i with length of s_i, which is less than 16. Some samples of the dataset $\mathscr{E}$ are shown in Table 3.

Two parameter sets of $\boldsymbol{\theta}$ for scoring were trained by using the following two algorithms programmed in Java on a desktop computer (MacOS 10.9, 3.2 GHz Intel core i3, 8 GB memory):

Algorithm 1 **Step 1**–**Step 4**, using Eq. (4).
Algorithm 2 **Step 1**–**Step 4**, with **Step 3** using

$$\begin{array}{ll} \theta_f := \theta_f + 2 & \text{for } \{f \le F_{total} | x_f(y_i) > 0\} \\ \theta_{\bar{f}} := \theta_{\bar{f}} - 1 & \text{for } \{\bar{f} \le F_{total} | x_{\bar{f}}(y_p) > 0\} \end{array} \tag{5}$$

in place of Eq. (4).

Table 3 Samples of the evaluation dataset $\mathscr{E}$

Input strings (s_i)	Length of s_i	Formulae (y_i)
a/=0	4	$a \neq 0$
A (3,–2)	7	A $(3, -2)$
3 <= y <= 7	7	$3 \leqq y \leqq 7$
7/9=0.7.	8	$\frac{7}{9} = 0.\dot{7}$
root32=[3]	10	$\sqrt{3^2} = \|3\|$
y=1/2x2–2x–1	12	$y = \frac{1}{2}x^2 - 2x - 1$
(a4)3=a4*3=a12	14	$\left(a^4\right)^3 = a^{4\times3} = a^{12}$
3x2y4*(–2x4y)3	14	$3x^2y^4 \times \left(-2x^4y\right)^3$

Table 4 Prediction accuracy using Algorithms 1 and 2

Training number	Best 1 (%)		Best 3 (%)		Best 10 (%)		Correct score	
	Algo. 1	Algo. 2	Algo. 1	Algo. 2	Algo. 1	Algo. 2	Algo. 1	Algo. 2
0	25.9 (3.8)	25.9 (3.8)	41.3 (4.4)	41.3 (4.4)	52.3 (4.3)	52.3 (4.3)	2.8 (0.1)	2.8 (0.1)
100	62.7 (14.7)	53.3 (14.6)	75.5 (9.2)	82.5 (6.4)	81.5 (6.8)	88.5 (4.3)	15.4 (1.2)	307.0 (58.1)
200	75.6 (6.6)	60.3 (5.0)	82.7 (5.0)	86.1 (4.2)	86.6 (4.3)	91.7 (3.2)	18.0 (1.5)	568.9 (99.4)
300	79.3 (4.1)	64.1 (5.1)	85.2 (4.3)	89.1 (3.2)	88.1 (4.3)	93.8 (2.9)	20.4 (1.9)	964.3 (186.8)
400	79.2 (3.8)	67.7 (5.7)	85.1 (3.8)	90.1 (3.1)	88.2 (3.5)	94.1 (3.1)	21.0 (2.2)	1103.4 (75.2)
500	80.0 (4.4)	67.6 (5.7)	86.7 (4.0)	90.6 (2.9)	89.5 (3.3)	94.5 (2.8)	23.1 (2.2)	1290.6 (99.8)
600	79.5 (3.7)	69.1 (4.6)	85.9 (3.4)	90.8 (2.7)	89.2 (3.8)	94.3 (2.5)	22.4 (2.4)	1492.2 (106.5)
700	79.1 (5.7)	68.5 (6.0)	85.7 (5.3)	91.1 (2.5)	89.2 (4.2)	95.0 (2.5)	22.9 (1.7)	1692.9 (114.7)

Numbers within parentheses denote the *SD*

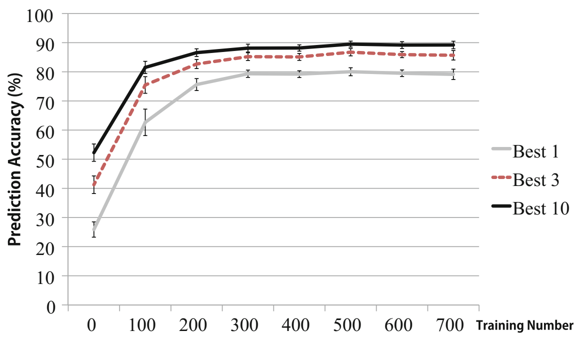

Fig. 1 The result by Algorithm 1 (training number–prediction accuracy)

In the experimental evaluation, we measured the proportion of correct predictions from among 100 test datasets after learning the parameters through Algorithms 1 and 2 using a training dataset consisting of 700 formulae by eightfold cross-validation.

The machine learning results using Algorithms 1 and 2 are given in Table 4 for each training number. By using Algorithm 1, the prediction accuracy of "Best 1" is about 79.1% after being trained 700 times. In the top ten ranking ("Best 10"), it achieves about 89.2%. Figure 1 shows the change in the prediction accuracy as a result of Algorithm 1 for each training number.

On the other hand, the result obtained by using Algorithm 2 with another learning weight shows that the prediction accuracy of "Best 1" is approximately 68.5% after being trained 700 times. It achieves about 95.0% in the top ten ranking. The change in the prediction accuracy as a result of Algorithm 2 is shown in Fig. 2.

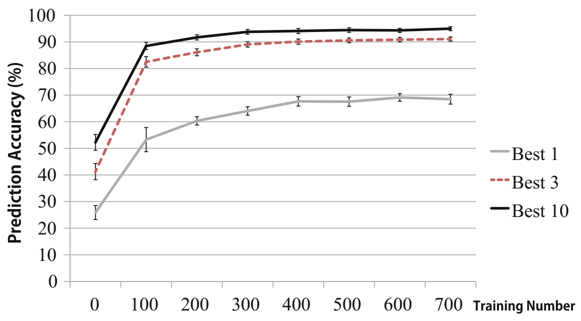

Fig. 2 The result by Algorithm 2 (training number–prediction accuracy)

4.2 Discussion

The mean scores for the correct expressions (“correct score” in short) in the test dataset for each training number are shown in the fifth column of Table 4 and illustrated in Fig. 3. The prediction accuracy of “Best 1” by using Algorithm 1 becomes sufficiently high, i.e., approximately 80%, with the mean correct score approximately equal to 23 after being trained 500 times. However, this is disadvantageous for a mathematical input interface, because the correct expressions out of the top ten ranking are more than 10%. One of the causes of this 10% leak is because the priorities of some correct expressions are not reflected in their occurrence frequency. In the case when two different candidates belonging to the same key appear from the training data, e.g., the pair a and α and the pair p and π, their scores change into a positive value from a negative value or vice versa. This means that even if a candidate with negative score occurred many times in $\mathscr{E}$, it has lower priority than the one with zero score because the increase and decrease in the weights of the score in Eq. (4) are mutually the same. For example, changes in score parameters (a and α) are shown in Fig. 4.

To avoid such problems, we have modified Algorithm 2 such that the increase in weight for the correct candidate is greater than the decrease in weight for the incorrect one as shown in Eq. (5). From the results of experimental evaluation of the

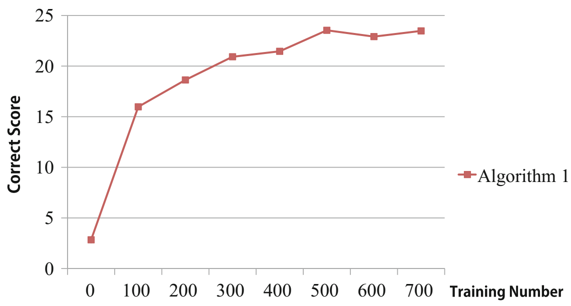

Fig. 3 Change in correct score given by Algorithm 1

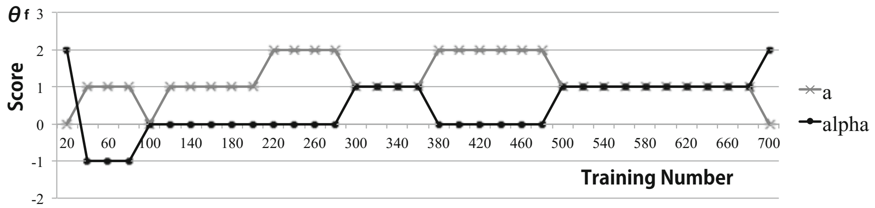

Fig. 4 Change in score parameters (a and α)

prediction accuracy by using Algorithm 2, the ratio of correct expressions from the top ten ranking is less than 5%, which is sufficient for a mathematical input interface system. However, we remark that the score parameter continues to increase while Algorithm 2 is learning.

In this study, we propose the following algorithm, Algorithm 3, to overcome the problems encountered in Algorithm 2.

Algorithm 3 **Step 1**–**Step 4**, where **Step 3** using

$$\begin{aligned} \text{if}(\theta_f < S_{\max})\{\theta_f &:= \theta_f + 2 \quad \text{for } \{f \leq F_{total} | x_f(y_i) > 0\}\} \\ \theta_{\bar{f}} &:= \theta_{\bar{f}} - 1 \quad \text{for } \{\bar{f} \leq F_{total} | x_{\bar{f}}(y_p) > 0\} \end{aligned} \tag{6}$$

in place of Eq. (5).

Here, $S_{\max}$ in Eq. (6) is a suitable upper bound for any mathematical element score. Because the result of Algorithm 1 provides good precision with a mean score of approximately 23, we set the upper bound $S_{\max}$ to 20 for any mathematical element score θ_f.

The machine learning results for Algorithm 3 for the case $S_{\max} = 20$ are given in Table 5 for various sizes of the training dataset. It can be seen that the accuracy of "Best 1" with Algorithm 3 was approximately 68.3% after being trained 700 times. This algorithm achieved an accuracy of 90.5% for the top three ranking, and 96.2% for the top ten ranking. With a training set of size 700, there is no statistically significant difference (at the 5% level) between the results for Algorithm 2 and those for Algorithm 3 for the "Best 1," "Best 3," or "Best 10" cases. Additionally, the learning curves for both algorithms change at the same skill rate for each of these cases. The mean correct scores in the test dataset for each training number are presented in the fifth column of Table 5 and illustrated in Fig. 5. The correct score with Algorithm 2 (shown in the fifth column of Table 4) increases proportionally with training number n (decision coefficient: $R^2 = 0.98$); however, the correct score with Algorithm 3 increases only at a rate of $\log n$ ($R^2 = 0.96$).

Comparing case $S_{\max} = 20$ with $S_{\max} = 50$, we conclude that precision properties of both are almost similar while the mean correct score for the test data when $S_{\max} = 20$ is 14% lower than that when $S_{\max} = 50$. However, if we set $S_{\max}$ to less than 20, the scores of the individual elements belonging to any one key are not much different

Table 5 Prediction accuracy using Algorithm 3

Training no.	Best 1 (%)	Best 3 (%)	Best 10 (%)	Correct score
0	25.9 (3.8)	41.3 (4.4)	52.3 (4.3)	2.8 (0.1)
100	54.2 (13.8)	82.6 (6.4)	88.7 (4.3)	283.0 (74.0)
200	65.7 (6.8)	87.7 (3.4)	93.0 (2.7)	428.6 (110.6)
300	69.5 (6.1)	88.3 (3.1)	94.0 (3.0)	494.1 (134.7)
400	67.9 (6.3)	88.8 (2.4)	94.3 (2.8)	536.7 (148.8)
500	69.2 (5.6)	89.8 (3.0)	95.2 (2.7)	566.2 (162.7)
600	70.6 (5.2)	90.9 (2.7)	95.9 (2.5)	590.0 (169.4)
700	68.3 (6.1)	90.5 (2.8)	96.2 (2.3)	608.0 (180.0)

Numbers within parentheses denote the *SD*

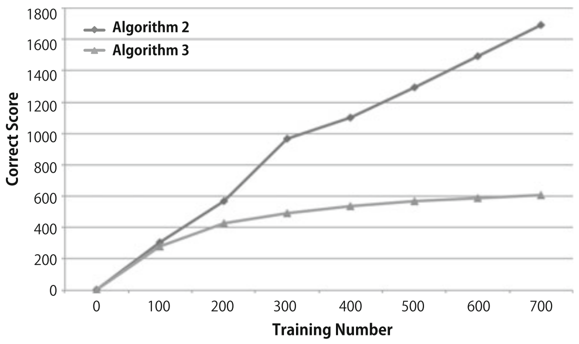

Fig. 5 Change in correct score given by Algorithms 2 and 3

in the machine training because the maximum number of elements belonging to any one key is equal to 20 in our key dictionary $\mathscr{D}$. Therefore, we propose $S_{\max} = 20$ to be the most suitable value in this study.

5 Conclusion and Future Work

In this paper, we proposed a predictive algorithm with an accuracy of 96.2% for the top ten ranking by improving upon a previously proposed algorithm in terms of a structured perceptron for stable score parameter learning. The mean CPU time for predicting each mathematical expression with corresponding linear string of length less than 16 obtained from a mathematics textbook was 0.44 s (SD = 0.61).

Because the linear strings for mathematical expressions are easily recognized from both handwritten image data and voice data for such mathematical expressions, it is possible that the desired mathematical expression is predicted with high accuracy from such linear strings by using this predictive algorithm. We believe that there is a possibility to apply this predictive algorithm to not only a mathematical input method on a PC with the keyboard but also for recognizing handwritten mathematical expressions and voice for mathematical expressions.

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

Acknowledgements This work was supported by JSPS KAKENHI Grant Number 26330413.

References

1. Fukui, T.: An intelligent method of interactive user interface for digitalized mathematical expressions. RIMS Kokyuroku **1780**, 160–171 (2012) (in Japanese)
2. 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)
3. Fukui, T.: An intelligent user interface technology for easy formatting of digitalized mathematical expressions–a mathematical expression editor on web-browser. Interaction 2013 IPSJ symposium, No.1, 2EX13-50, pp. 537–540 (2013) (in Japanese)
4. Fukui, T.: Prediction for converting linear strings to mathematical formulae using machine learning. In: Proceedings of ARG WI2, No. 6, pp. 67–72 (2015) (in Japanese)
5. Garay-Vitoria, N., Abascal, J.: Text prediction systems: a survey. Univers. Access. Inf. Soc. **4**(3), 188–203 (2006)
6. Hijikata, Y., Horie, K., Nishida, S.: Predictive input interface of mathematical formulas Human-Computer Interaction-INTERACT2013, Vol. 8117 of the series Lecture Notes in Computer Science. Springer, New York (2013)
7. Hunnicutt, S.: Input and output alternative in word prediction. STL/QPRS **28**(2–3), 15–29 (1987)
8. Iidaka, S., Matsumoto, Y., et al.: Mathematics I, **001**, TOKYO SHOSEKI (2012) (in Japanese)
9. Manning, C.D., Scheutze, H.: Foundations of Statistical Natural Language Processing. The MIT Press, London (2012)
10. Pollanen, M., Wisniewski, T., Yu, X.: XPRESS: a novice interface for the real-time communication of mathematical expressions, In: Proceedings of MathUI (2007)
11. Sangwin, CJ.: Computer aided assessment of mathematics using STACK, In: Proceedings of ICME, vol. 12 (2012)
12. Shirai, S., Fukui, T.: Development and evaluation of a web-based drill system to master basic math formulae using a new interactive math input method, Mathematical Software—ICMS2014, vol. 8592 of the Series Lecture Notes in Computer Science, pp. 621–628. Springer, New York (2014)
13. Shirai, S., Fukui, T.: Improvement in the input of mathematical formulae into STACK using interactive methodology. Comput. Educ. **37**, 85–90 (2014) (in Japanese)