社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

# サポートベクターマシンによる数学記号認識

クリストファー　マロン[†]　　鈴木　昌和[††]　　内田　誠一[†††]

† マサチューセッツ工科大学数学部
†† 九州大学大学院数理学研究院　〒812–8581 福岡市東区箱崎 6–10–1
††† 九州大学大学院システム情報科学研究院 〒812–8581 福岡市東区箱崎 6–10–1
E-mail: †malon@mit.edu, ††suzuki@math.kyushu-u.ac.jp, †††uchida@is.kyushu-u.ac.jp

**キーワード**　数式認識、記号認識、文字認識、サポートベクターマシン

# Support Vector Machines for Mathematical Symbol Recognition

Christopher MALON[†], Masakazu SUZUKI[††], and Seiichi UCHIDA[†††]

† MIT Department of Mathematics
77 Massachusetts Ave., Cambridge, MA 02139 USA
†† Engineering Division, Faculty of Mathematics, Kyushu University
6–10–1 Hakozaki, Higashi-ku, Fukuoka-shi, Fukuoka, 812–8581 Japan
††† Faculty of Information Science and Electrical Engineering, Kyushu University
6–10–1 Hakozaki, Higashi-ku, Fukuoka-shi, Fukuoka, 812–8581 Japan
E-mail: †malon@mit.edu, ††suzuki@math.kyushu-u.ac.jp, †††uchida@is.kyushu-u.ac.jp

**Abstract**　Mathematical formulas challenge an OCR system with a range of similar-looking characters whose bold, calligraphic, and italic varieties must be recognized distinctly, though the fonts to be used in an article are not known in advance. We describe the use of support vector machines (SVM) to learn and predict about 300 classes of styled characters and symbols.

**Key words**　mathematical formula recognition, symbol recognition, character recognition, support vector machines

## 1. Introduction

Optical character recognition problems were considered very early in the development of support vector machines, with promising results [1]. However, the problem of OCR for mathematical documents is substantially more difficult than standard OCR problems for three principal reasons:

（1） Although a variety of fonts is used in mathematical literature, when reading any single paper, it is important to keep appearances of italic, bold, roman, calligraphic, typewriter, and blackboard bold letters distinguished.

（2） A rich set of symbols is used, and distinctions between letters may be more subtle than within the character set of a typical human language.

（3） The symbols are not arranged in a simple one–dimensional pattern. Subscripts, superscripts, fractional relationships, and accents occur, and may be nested [2].

The *Infty Project [7]* in the Suzuki Laboratory at Kyushu University is developing *Infty Reader* software [5] to perform OCR of scanned mathematical journal articles, and produce output in languages that allow symbol relationships to be properly encoded, including TEXand MathML. Although Infty Reader nominally achieved 99 percent accuracy of single–letter recognition before this investigation (October 2005), its failure to distinguish certain common symbols would be bothersome to any serious user.

The Infty Project defined entities for about 600 characters and symbols used in mathematical research, and created a ground truth database identifying their appearances in page–by–page scans of hundreds of journal articles. Many character pairs could be distinguished in different styles by simple clustering techniques applied to directional features measured in a mesh grid. Runtime accuracy exceeded 99% , but hundreds of letter pairs remained consistently problematic. We aim to improve the accuracy of single–character recognition through the use of support vector machines.

## 2. Test Data

The Infty character set comprises 1,571 Roman and Greek letters, numbers, and mathematical symbols, divided into 46 categories according to purpose and style. Further details of these characters appear in [8].

The Infty Project has selected journal articles representing diverse fields of higher mathematics, taken from a thirty year period. These articles were scanned page by page at 600 dpi to produce bitmap image files. The Infty OCR engine extracted the symbols from each page and recognized each symbol as a character from the Infty character set. College and graduate mathematics students manually inspected and corrected the results.

The results of this process appear in a "ground truth" database. Namely, for each character on a scanned page, a bitmap framed by the bounding box of that character is taken from the page. This bitmap is tagged with the correct identification of the symbol. (注1) "Link information" describing the character's relationship to others on the page (subscripts, superscripts, limits, portions of a fraction, *etc.*) is also available in the database, but it is not utilized in the present study.

In fact, the Infty project has produced two databases of this kind. One, called InftyCDB-1, is freely available for research purposes upon request, and is summarized in [8]. The other is used internally by the Infty Reader OCR engine. We use the latter database in this experiment, because it has more data, and because it makes it easier to compare our results with those of the actual Infty Reader. Our data sample consists of 284,739 character symbols extracted from 363 journal articles. There are 608 different characters represented.

At random, we divide the 363 articles into three parts consisting of 121 articles each. The data from the corresponding articles is marked as "training", "selection", or "testing" accordingly. To make sure we had enough data to train and evaluate our classifiers, we examined only the characters with at least ten samples in training, selection, and testing portions of the database. This left 297 characters, pictured in Figure 1.

(注1) : Some bitmaps would not be identifiable solely on the basis of this bitmap. For example, a hyphen could not be distinguished from an underscore, without knowing its relationship to the baseline on the page when it was scanned. The original position on the page is part of the database, but this information was discarded prior to our experiment.

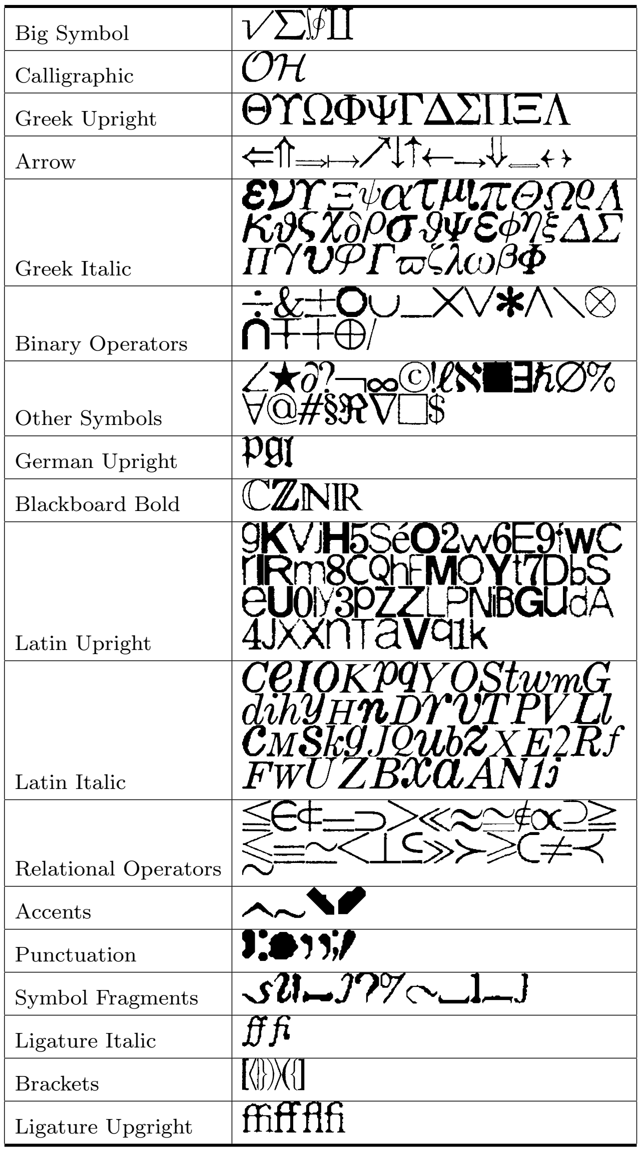
Big Symbol
Calligraphic
Greek Upright
Arrow
Greek Italic
Binary Operators
Other Symbols
German Upright
Blackboard Bold
Latin Upright
Latin Italic
Relational Operators
Accents
Punctuation
Symbol Fragments
Ligature Italic
Brackets
Ligature Upgright

Figure 1 Symbols with 10 training, selection, and testing samples

## 3. Directional Features

Given an instance of a symbol, let $w$ be its width and $h$ be its height. Our feature vectors consist of the aspect ratio $(\frac{h}{w})$, followed by 160 floating–point coordinates of mesh directional feature data.

This mesh data is divided into "tall", "square", and "short" blocks of 48, 64, and 48 coordinates respectively. When the aspect ratio of a character exceeds 1.3, the tall block contains directional feature data computed from a $3 \times 4$ mesh; otherwise it contains zero–valued entries. When the aspect ratio of a character is between $\frac{1}{1.7}$ and 1.7, the square block contains directional feature data from a $4 \times 4$ mesh; otherwise it contains zero–valued entries. When the aspect ratio of a character is less than $\frac{1}{1.3}$, the short block contains directional features computed from a $4 \times 3$ mesh; otherwise it contains zero–valued entries. Thus, for any symbol, one or two (but never three) of the blocks are assigned nonzero entries.

We describe roughly the algorithm for associating directional feature data to an $m \times n$ mesh block. Divide the original bitmap horizontally into $m$ equally sized lengths, and vertically into $n$ equally sized lengths. Assign a "chunk" of four coordinates of the block to each of the $m \times n$ grid positions; initially, their values are zero. These four coordinates represent the horizontal and vertical directions, and two diagonals.

The contribution of part of the outline's direction to the mesh features is determined from its position in the bitmap, using a partition of unity. Given a positive integer $r$, consider the $r$–fold partition of unity given by the functions $p_i^r : [0,1] \to [0,1]$ defined by

$$p_i^r(x) = \begin{cases} 0 & x < \frac{i}{r} - \frac{1}{2r} \\ r(x - (i/r - 1/2r)) & \frac{i}{r} - \frac{1}{2r} \leqq x < \frac{i}{r} + \frac{1}{2r} \\ -r(x - (i/r + 3/2r)) & \frac{i}{r} + \frac{1}{2r} \leqq x < \frac{i}{r} + \frac{3}{2r} \\ 0 & x > \frac{i}{r} + \frac{3}{2r} \end{cases} \tag{1}$$

for $1 \leqq i < r - 1$,

$$p_0^r(x) = \begin{cases} 1 & x < \frac{1}{2r} \\ -r(x - 3/2r) & \frac{1}{2r} \leqq x < \frac{3}{2r} \\ 0 & x > \frac{3}{2r} \end{cases} \tag{2}$$

and

$$p_{r-1}^r(x) = \begin{cases} 0 & x < 1 - \frac{3}{2r} \\ r(x - (1 - \frac{3}{2r})) & 1 - \frac{3}{2r} \leqq x < 1 - \frac{1}{2r} \\ 1 & 1 - \frac{1}{2r} \leqq x \end{cases} . \tag{3}$$

Discard every isolated black pixel from the original bitmap. In the remaining bitmap, trace every outline between white and black pixels, following its chain code description. When visiting the pixel in location $(x, y)$ during this trace, identify the direction (horizontal, vertical, diagonal one, or diagonal two) where the next pixel in the outline will be. For every $i$, $0 \leqq i < m$, and every $j$, $0 \leqq j < n$, add $p_i^m(\frac{x}{w}) \cdot p_j^n(\frac{y}{h})$ to the coordinate of the $(i, j)$ chunk representing that direction.

After completing the trace of each outline component, divide all the values by the perimeter of the bounding box. This result gives the values to be entered in the corresponding block of the feature vector.

## 4. Naive classifier

Typically, a support vector machine learns a binary classification. There are various techniques for putting SVM's together to distinguish multiple classes; a comparison of some popular methods (1–vs–1, 1–vs–all, and the Directed Acyclic Graph) may be found in [4]. Except for the 1–vs–all method, these methods require the construction of $O(n^2)$ classifiers to solve an $n$–class classification problem. Because the Infty character set includes more than 1,500 entities, this seemed unnecessarily burdensome. Therefore, we try to extract an easier part of the classification problem that can be solved without SVM.

Taking the data assigned to the "training" portion of the database, we compute the mean feature vectors for the instances of each symbol. We create a naive classifier that assigns an input to the class whose mean feature vector is nearest, by Euclidean distance.

We run this naive classifier on the "selection" portion of the database, to produce a confusion matrix. The $(i, j)$ entry of this matrix counts the number of samples in which a character truly belonging to class $i$ was assigned to class $j$ by this rule. The 297 by 297 confusion matrix we produced had 947 nonzero off-diagonal entries, an average of 3.2 misrecognitions per character.

We consider some of the misrecognitions to be too difficult for any classifier to resolve on the basis of our mesh of directional features. Particularly, we do not expect bold and non–bold variants of the same character to be distinguishable. Also, we do not expect upper and lower case variants of the letters C, O, P, S, V, W, X, and Z to be distinguishable in the same style, or in styles that are identical except for boldness. Disregarding misrecognitions of these two kinds, 896 other nonzero off–diagonal entries remain in the confusion matrix.

For 62 of the 297 characters with ten training, selection, and testing samples, the naive classifier recognized less than half of the selection samples correctly. These characters are displayed in Figure 2. In comparison, ninety percent accuracy is achieved for 197 of the 297 symbols, 95 percent accuracy for 163 symbols, and 99 percent accuracy for 123 symbols.

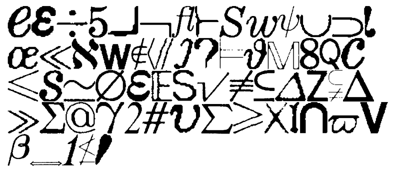
Figure 2 Characters the naive classifier usually fails to recognize

Although the confusion matrix is relatively sparse, certain troublesome characters have many misrecognition results, as can be seen in Figure 3. For 95 of the 297 characters, at least four distinct characters occur as misrecognition results.

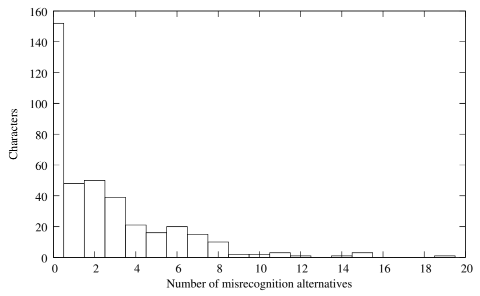

Figure 3 Histogram of distinct misrecognitions of an input character by the naive classifier

Eleven letters (plain '1', '4', 'E', 'I', 'l', 'r', 's', 't', ' “ ', and italic '$\gamma$' and '$\psi$') had ten or more distinct characters appear as misrecognition results.

At runtime, the naive classifier will be used to assign each letter to a cluster of possible candidates, consisting of the recognition result and the other candidates most likely to have produced that recognition result (as determined by our confusion matrix). The harder problem of distinguishing between the letters in each of these clusters will be assigned to support vector machines.

## 5. Linear SVM

Within each cluster, we will use the 1–to–1 approach to multiclass classification. This requires first creating a binary SVM for each pair of classes in the cluster.

Because they are simple and can be computed quickly, we begin our experiment with SVM's that use the linear kernel:

$$K(\vec{x}, \vec{y}) = \vec{x} \cdot \vec{y}. \tag{4}$$

The naive classifier, when restricted to two classes, can be thought of as the linear classifier determined by the hyperplane equidistant from the two cluster centers. The support vector method enables us to search for hyperplanes in the original feature space that perform better on the training data.

There are no kernel parameter choices needed to create a linear SVM, but it is necessary to choose a value for the soft margin ($C$) in advance. Then, given training data with feature vectors $\vec{x_i}$ assigned to class $y_i \in \{-1, 1\}$ for $i = 1, \ldots, l$, the support vector machines solve

$$\min_{\vec{w}, b, \vec{\xi}} \ \tfrac{1}{2}K(\vec{w}, \vec{w}) + C\sum_{i=1}^{l} \xi_i \tag{5}$$

$$\text{subject to } \ y_i(K(\vec{w}, \vec{x_i}) + b) \geqq 1 - \xi_i$$

$$\xi_i \geqq 0$$

where $\vec{\xi}$ is an $l$–dimensional vector, and $\vec{w}$ is a vector in the same feature space as the $\vec{x_i}$ (see, *e.g.*, [3]). The values $\vec{w}$ and $b$ determine a hyperplane in the original feature space, giving a linear classifier. *A priori*, one does not know which value of soft margin will yield the classifier with the best generalization ability. We optimize this choice for best performance on the selection portion of our data, as follows.

Our basic parameter search method, here and in the following sections, is a grid search method that generalizes to any number of dimensions. For each candidate parameter assignment, we train an SVM with those parameters on the training portion of our data. Then we measure its performance on the instances of two classes that appear in the selection data. The score of the parameter is the minimum of the accuracy on the first class's input and the accuracy on the second class's input. *Hereafter, “accuracy” by itself, in the context of a binary classification problem, will refer to this score.*

Often, grid searches require a search interval to be specified for each dimension. Our approach requires only an initial parameter choice, and then grows the search region outward, until performance stops improving. More formally, let $x_1, \ldots, x_n$ be the variable parameters, and $x_1^{init}, \ldots, x_n^{init}$ be their initial values. Let $L$ be the discrete set of points of the form $(2^{j_1}x_1^{init}, \ldots, 2^{j_n}x_n^{init})$ for some $(j_1, \ldots, j_n) \in \mathbb{Z}^n$.

(1) For each $i$, set $x_i^{min}$ and $x_i^{max}$ to $x_i^{init}$.

(2) Set the iteration count $t$ to zero.

(3) Let $B$ be the box $\left[x_1^{min}, x_1^{max}\right] \times \cdots \times \left[x_n^{min}, x_n^{max}\right]$, and $\partial C$ be its boundary. Record the best value on each surface of the box.

(4) For each untested parameter value in the finite set $L \cap \partial C$, train a new SVM on those parameters, and assign it a score based on selection data performance, as explained above.

(5) If the performance is superior to 99.5% accuracy, skip to the last step.

(6) From these results, for every $i$, set $b_i^{t,+}$ and $b_i^{t,-}$ to the highest scores achieved on the hyperplanes $x_i = x_i^{max}$ and $x_i = x_i^{min}$, respectively.

(7) If $t > 0$, double $x_i^{max}$ for every $i$ such that $b_i^{t,+} \geqq b_i^{t-1,+}$, and halve $x_i^{min}$ for every $i$ such that $b_i^{t,-} \geqq b_i^{t-1,-}$. If $t = 0$, double $x_i^{max}$ and halve $x_i^{min}$ for all $i$, regardless.

(8) Increment the iteration count $t$.

(9) If $t < 3$ or the best performance has increased in the last iterations, return to step 3.

(10) Report the best performance, and the most recently tested parameter selection that reached that performance.

Initial choices may matter under this grid search algorithm, if the algorithm terminates before reaching a selection of parameters that produces globally optimal results. This possibility seems unlikely as long as the resulting SVM performs better than random guessing in each case. The linear SVM problem has only the soft margin $C = x_1$ as a parameter, and we set it initially to be $x_1^{init} = 1024$.

Table 1 displays the accuracy achieved by the linear SVM selected, on the testing data for pairs of symbols that the naive classifier sometimes confused.

| Accuracy > | Number of pairs |
|---|---|
| Total | 795 |
| 0 | 783 |
| .5 | 774 |
| .6 | 770 |
| .7 | 767 |
| .8 | 759 |
| .9 | 750 |
| .95 | 742 |
| .97 | 720 |
| .99 | 684 |
| .995 | 650 |
| .999 | 609 |

Table 1 Linear SVM performance

We compared the chosen linear SVM classifier's performance on the letters where the naive classifier did not reach 100% accuracy, to the performance of the naive classifier. The 896 misrecognitions of the naive classifier comprise 795 unordered pairs of symbols. For nine of these pairs, both the naive classifier and the linear SVM always misassigned one of the two characters. Figure 4 compares the performance of the two methods on the remaining 786 pairs. Of the 786 pairs, 34 did not perform as well under the linear SVM as with the naive classifier. The exact same performance was achieved on 95 pairs, and improvement occurred on 657 pairs. The histogram does not report the 24 symbols with more than a three–fold improvement in accuracy. Thirteen of these symbols received zero accuracy from the naive classifier, for an infinite improvement in performance.

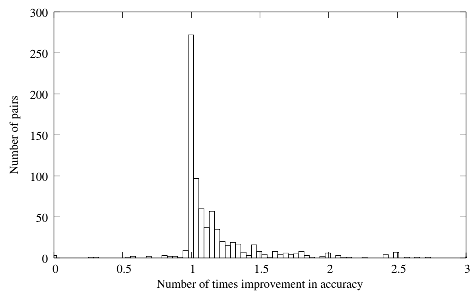

Figure 4 Histogram of improvement of linear SVM compared to naive classifier

Figure 5 illustrates the cases where the linear SVM achieved double the accuracy of the naive classifier.

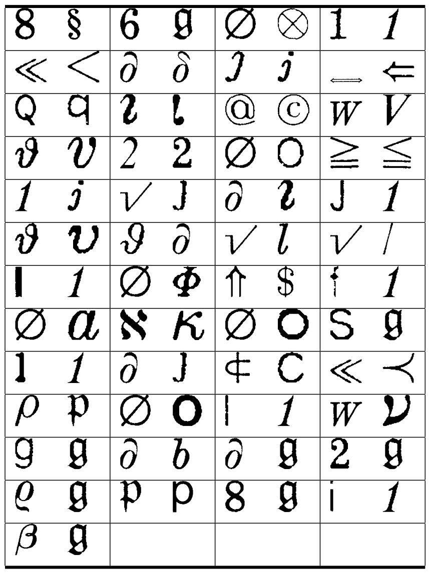

Figure 5 Pairs with two–fold improvement over naive classifier using linear SVM

## 6. Gaussian SVM

Just by using linear kernel support vector machines, our symbol recognition rates dramatically improved, but the use of a linear kernel severely limits the potential benefit of a support vector machine. The use of a Gaussian (radial) kernel

$$K(\vec{x}, \vec{y}) = e^{-\gamma \|\vec{x}-\vec{y}\|^2} \qquad (6)$$

in the SVM problem (5) effectively transforms the input feature space into an infinite–dimensional one, where the search for an optimal separating hyperplane is carried out. Classifiers of this form may perform better on classes whose feature data is not linearly separable in the original feature space. However, the addition of the parameter $\gamma$ in the kernel definition makes the parameter search two–dimensional, adding computational expense to the selection of a classifier.

According to a result of Keerthi and Lin [6], given a soft margin $C$, the sequence of Gaussian SVM classifiers with kernel parameter $\gamma$ and soft margin $\frac{C}{2\gamma}$ converges pointwise, as $\gamma \to 0$, to the linear SVM classifier with soft margin $C$. Thus, if our parameter search is wide enough, we should achieve higher accuracy with the Gaussian kernel than with the linear one.

We constructed Gaussian–kernel SVM classifiers for the 75 pairs of letters that the linear kernel failed to distinguish with 97% accuracy. A comparison of the performance of the chosen classifiers for each kernel type is given in Figure 6. In Figure 7, we display the eight pairs on which the Gaussian SVM performed with at least 10% higher accuracy than the linear SVM. The 31 pairs where Gaussian SVM accuracy falls below 80% are shown in Figure 8.

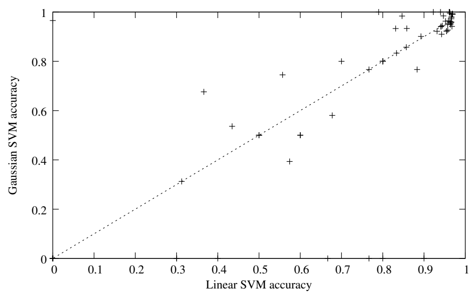

Figure 6 Comparison of linear and Gaussian SVM performance

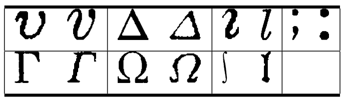

Figure 7 Pairs with 10% improvement from linear to Gaussian kernel

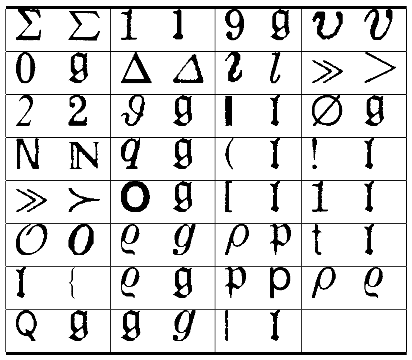

Figure 8 Pairs with under 80% accuracy by Gaussian SVM

## 7. Conclusion

Even with the simplest kernel, the support vector method is strong enough to achieve good generalization accuracy on an optical character recognition problem that causes difficulty for simpler classification methods. We believe that our SVM results may be the best classification possible on the basis of the mesh of directional features we are using.

To distinguish the characters that confuse our SVM classifier, we plan to add new features. For example, by counting the number of connected components in a symbol, we could distinguish many variants of the greater–than sign (>). We also plan to record the convexity or concavity of a symbol as traced along its outline, to distinguish various nearly vertical characters. These features will be the topic for a future paper.

To our surprise, the SVM's we constructed with the Gaussian kernel did not show significantly stronger performance on the testing data. We attribute this phenomenon to the simple nature of our mesh of directional features. We plan to repeat this comparison after attaching a greater variety of features to our data.

## Acknowledgments

The first author thanks the MIT Japan Program for its support of the visit that made this collaboration possible, and Kyushu University for its generous hospitality during his stay.

### References

[1] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, 1995.

[2] Yuko Eto and Masakazu Suzuki. Mathematical formula recognition using virtual link network. In *Sixth International Conference on Document Analysis and Recognition (ICDAR)*, pages 430–437. IEEE Computer Society Press, 2001.

[3] C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification. `http://www.csie.ntu.edu.tw/%7Ecjlin/papers/guide/guide.pdf`, July 2003.

[4] C.-W. Hsu and C.-J. Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13:415–425, 2002.

[5] The infty project. `http://infty.math.kyushu-u.ac.jp`.

[6] S. Sathiya Keerthi and Chih-Jen Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural Comput.*, 15(7):1667–1689, 2003.

[7] Masakazu Suzuki, Fumikazu Tamari, Ryoji Fukuda, Seiichi Uchida, and Toshihiro Kanahori. Infty: an integrated ocr system for mathematical documents. In *DocEng '03: Proceedings of the 2003 ACM symposium on Document engineering*, pages 95–104, New York, NY, USA, 2003. ACM Press.

[8] Masakazu Suzuki, Seiichi Uchida, and Akihiro Nomura. A ground-truthed mathematical character and symbol image database. In *ICDAR '05: Proceedings of the Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 675–679, Washington, DC, USA, 2005. IEEE Computer Society.