

# Feature Kernel Functions: Improving SVMs Using High-level Knowledge

Qiang Sun, Gerald DeJong

*Department of Computer Science, University of Illinois at Urbana-Champaign*  
*qiangsun@uiuc.edu, dejong@cs.uiuc.edu*

## Abstract

*Kernel functions are often cited as a mechanism to encode prior knowledge of a learning task. But it can be difficult to capture prior knowledge effectively. For example, we know that image pixels of a handwritten character result from a few strokes from a single writing implement; it is not clear how to express this in a kernel function. We investigate an Explanation Based Learning (EBL) paradigm to generate specialized kernel functions. These embody novel high-level features that are automatically constructed from the interaction of prior knowledge and training examples. Our empirical results showed that the performance of the resulting SVM surpasses that of a conventional SVM on the challenging task of classifying handwritten Chinese characters.*

## 1. Introduction

The ability to learn from experience can benefit most computer systems. A system that is programmed once and for all cannot exploit unanticipated systematic behavior of the world. On the other hand, most approaches to machine learning have difficulty incorporating prior knowledge in a flexible way, and any task information that we possess but do not tell a learner will cost in terms of increased training requirements. It is well established that learning is mathematically impossible without prior bias. This inductive bias is often cited as the vehicle for encoding prior task knowledge. However, the bias vocabularies of learners seldom fit task experts' conceptualizations of their prior knowledge. We examine this issue in the context of automatically constructing specialized kernel functions for support vector machines.

Kernel functions are often mentioned as the mechanism by which prior knowledge may be incorporated into an SVM classifier [1]. Intuitively, a good kernel function should define a space in which inter-example distances reflect the intrinsic differences

between classes. Such kernel functions highlight features of the examples which are most informative of class membership while de-emphasizing less informative ones. Informative features are highly discriminative, reliably detected in examples from one class, and reliably missing from examples of the other.

To illustrate, consider distinguishing between postal code images of handwritten "3" and "8" digits. We know that the left part of the figure is likely to be more informative than the right. We might call such knowledge a "figure bias" since it applies to the actual "3" and "8" figures rather than, say, their pixel representations. Clearly, to be used in a kernel function, this figure bias must be expressed in the image vocabulary. But the particular combinations of pixels or image regions that best reflect this figure bias depends on the underlying population of examples: how they are represented, how effectively they are normalized and registered, the noise level, and so on. Thus, an effective specialized kernel function results from the interaction of three sources of knowledge: intra-class knowledge (e.g., the features that compose an "8"), inter-class knowledge (e.g., the subset reliable features that distinguish "8" from "3"), and example characteristics (e.g., which pixels can reliably indicate the presence of informative features).

Explanation-Based Learning (EBL), of the sort described in [2] (rather than the more conventional one [3]) supports just this sort of knowledge interaction. Central to the EBL approach is the construction of an *explanation* for why a particular training example merits its teacher-assigned class label. Training examples are viewed as illustrations of some underlying robust pattern. In this paper, we illustrate how to apply EBL approach to adapt a kernel function for the given classification task. The main contribution of our research is to use high-level knowledge in an EBL fashion to engineer a specialized kernel function that operates on low-level attributes.

Here, an explanation is not a logical proof but only general conjecture to justify the training examples' labeling using high-level features. It then forms what we will call a *component kernel*, which is a special

kernel function designed to detect a newly invented high-level feature. A collection of weighted component kernels defines a *feature kernel function*. It specifies a metric space which embodies the optimal linear blending of the informative features as detected by the component kernel functions. Thus, a feature kernel function is tailored to a specific classification task with awareness of high-level features.

We exercise our approach distinguishing images of handwritten Chinese characters. As there are thousands of common Chinese characters, no existing database of training images includes more than a few hundred of each. And yet individual Chinese characters are generally much more complex than individual digits or Western characters. The dual characteristics of complex examples and limited training data fit our approach well and apply to many real-world classification and discrimination tasks.

Compared to a conventional SVM of similar design, our results show that 1) a more accurate classifier is acquired, 2) an effective classifier can be acquired with significantly fewer training examples, and 3) the resulting classifier is significantly less complex, employing fewer support vectors.

## 2. Overview of Our Approach

EBL requires a domain theory of prior knowledge to drive the explanation process. While images of characters are composed of pixels, it is natural to break the span from pixels to characters into two sub-domains. The first relates handwritten characters to the *strokes* that are used to form them. The notion of a stroke is not intrinsic to this classification problem. A conventional SVM has no place for such a notion. Rather, strokes are introduced as “hidden” features to organize prior knowledge. In addition, combinations of strokes form yet another level of derivable hidden features, called *stroke-level features*. For example, the two prototype Chinese characters shown in the Figure 2 are quite similar. However, some stroke-level features are quite informative for this discrimination task. We have circled these in Figure 1. Described with these derived stroke-level features the characters are quite different.



**Figure 1.** Two similar Chinese characters with the difference between them high-lighted.

The second sub-domain explains the correspondence between pixels and strokes. Strokes are

modeled as straight lines of a particular width with a particular starting and ending location. The corresponding pixels are those that fall within the boundaries of a long thin rectangle which is the stroke. The correspondence is determined by applying a Hough transformation [4] to detect lines in the training images. Note that using a Hough transform on training images is far more reliable than to find lines in an unknown image. The label of the training image provides access to its prototype stroke representation. Thus, the procedure is reduced to finding image lines that best match the known stroke lines.

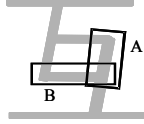
Given a pair of Chinese character labels and a set of training examples for each, the following procedure is performed to produce a feature kernel function:

- 1) Conjecture stroke-level features: The prototypes of the characters to be distinguished are examined, and distinctive stroke interactions are identified. These are candidates for the construction of component kernel functions.
- 2) Determine the best pixel representation for each stroke: This involves
  - i) Explaining labeled character images by using Hough transform to determine how each required stroke is realized by its pixels.
  - ii) Estimating the correlation between pixels and strokes across the data set. The detail is discussed in section 5.2.
  - iii) Constructing parameterized evidence pixel sets for each stroke. The parameter is a threshold specifying the minimal acceptable correlation for a pixel to be included.
- 3) Given evidence pixel sets for strokes, construct the set of component kernel functions. This is the topic of section 3. It also involves choosing values for the parameters mentioned above. They are evaluated so as to minimize stroke-level feature detection errors over whole training set.
- 4) Build the feature kernel function: Using the algorithms described in section 4, component kernel functions of the previous step are weighted to produce the final feature kernel function.

## 3. Component Kernel Functions

As described, stroke-level features represent interactions between strokes. A component kernel is a specialized kernel to serve as a detector for a stroke-level feature, and is used to assemble the final feature kernel function. Conceptually, component kernels operate over *monomials*. A monomial represents the product of pixels, as evidence for a stroke-level feature.

Given the connections between pixels and strokes, it is straightforward to determine evidence monomials for different stroke-level features. This is illustrated in Figure 2. When the pixels in the region A serve as evidence for a horizontal stroke, and the pixels in the region B serve as evidence for a vertical stroke, then those second-degree monomials with one pixel from region A and another from region B can be evidence for the “corner” feature composed with the horizontal and vertical strokes.



**Figure 2.** Two stroke-level features to detect a corner.

Specifying a function to compute the dot product between two examples using those evidence monomials gives us a component kernel function for SVMs to detect the corresponding stroke-level features. The corner feature shown in Figure 4, employs a component kernel function like the following:

$$k_{A,B}(\bar{x}_1, \bar{x}_2) = \left( \sum_{i \in A} x_{1i} x_{2i} \right) \left( \sum_{i \in B} x_{1i} x_{2i} \right)$$

It is easy to see that this kernel function computes the dot product between example  $x_1$  and  $x_2$  using the monomials of pixels from region A and region B:

$$k_{A,B}(\bar{x}_1, \bar{x}_2) = \sum_{i \in A, j \in B} x_{1i} x_{2i} x_{1j} x_{2j} = \sum_{i \in A, j \in B} (x_{1i} x_{1j}) (x_{2i} x_{2j})$$

Therefore, a component kernel function is similar to a conventional kernel function in that it is easy to compute, and it gives the value of dot product between two examples in high-dimensional space. The only difference is that a component kernel function is designed for discovering the presence of certain high-level feature in the example, not for determining the label. We can also build higher order component kernel functions for the same corner feature, such as:

$$K_{A,A,B}(\bar{x}_1, \bar{x}_2) = \sum_{i \in A, j \in A, k \in B} (x_{1i} x_{1j} x_{1k}) (x_{2i} x_{2j} x_{2k})$$

This kernel uses monomials with three pixels, two from region A, and the other one from region B. We can also define a similar function that uses monomials with one pixel from region A, and two from region B.

## 4. Combining Component Kernel Functions

The component kernel functions are designed to detect stroke-level features, but not all features are equally useful to distinguish characters. In this section, we examine how linearly combine the component

kernel functions to obtain a feature kernel function to classify future character examples. Using different weights, our feature kernel function is able to emphasize diagnostic stroke-level features, while de-emphasizing less informative or redundant ones.

Weighting kernel functions is a question addressed by many other researchers [5-7]. In our study, we employ the approach proposed in [7]. Here we briefly summarize the approach. It uses a simple computable notion, called *alignment* [5], to estimate the goodness of a kernel function. The (empirical) alignment of a kernel  $k_1$  with another kernel  $k_2$  with respect to the sample  $S$  is defined as:

$$A_S(k_1, k_2) = \frac{\langle K_1, K_2 \rangle_F}{\sqrt{\langle K_1, K_1 \rangle_F} \sqrt{\langle K_2, K_2 \rangle_F}},$$

where  $K_i$  is the kernel matrix for the sample  $S$  using kernel  $k_i$ , and  $\langle \cdot, \cdot \rangle_F$  is the Frobenius product<sup>1</sup>. Let  $y$  be the target function that gives the label, then the target kernel  $k^* \equiv yy'$  can be defined as  $k^*(\bar{x}, \bar{z}) = y(\bar{x})y(\bar{z})$ . The (empirical) *kernel target alignment* can be defined as  $A_S(k) = A_S(k, k^*)$ . Higher kernel target alignment implies better performance of the resulting SVM classifier [5]. Given a set of kernels  $k_i$ , the problem of combining them can be formulated as to choose  $\alpha$  in  $k(\alpha) = \sum_i \alpha_i k_i$  so that the alignment of  $k(\alpha)$  to the given target vector  $y$  is optimized. Kandola etc [7] showed that this optimization problem can be solved by the standard quadratic programming.

## 5. Analysis of Our Approach

Our algorithm uses the notion of stroke-level features to organize pixel level features, and enables SVMs to emphasize the distinctive stroke-level features by weighting component kernel functions. Intuitively, emphasizing small amount of useful features can greatly reduce learning complexity, therefore allowing SVM learning algorithm to perform well even when small number of training examples are available. In this section, we use the notion of kernel alignment to provide insights about how our algorithm can produce an improved kernel function.

### 5.1. Decomposing a Kernel Function

First, we observe that weighting multiple kernel functions can usually produce a better kernel function than simply combining them with equal weights. Specifically, given two kernel functions  $k_1$  and  $k_2$ , the

<sup>1</sup> The Frobenius product between two matrices  $M=[m_{ij}]$  and  $N=[n_{ij}]$  is defined by  $\langle M, N \rangle = \sum_{ij} m_{ij} n_{ij}$

target alignment of the combined kernel function with coefficient  $l$  and  $w$  is:

$$A(w) = \frac{\langle K_1, yy' \rangle + w \langle K_2, yy' \rangle}{\sqrt{\langle K_1, K_1 \rangle + 2w \langle K_1, K_2 \rangle + w^2 \langle K_2, K_2 \rangle} \cdot \sqrt{\langle yy', yy' \rangle}}$$

With some algebra, we see that the optimal  $w$  to maximize  $A(w)$  is  $w_{opt} = \frac{(A_2 - A_{12} \cdot A_1) \cdot |K_1|}{(A_1 - A_{12} \cdot A_2) \cdot |K_2|}$ , where  $A_1$ ,  $A_2$  denote target alignment for  $k_1$  and  $k_2$ ,  $A_{12}$  denotes alignment between  $k_1$  and  $k_2$ , and  $|K_i| = \sqrt{\langle K_i, K_i \rangle}$ . The target alignment of the optimally weighted kernel is

$$A(w_{opt}) = \frac{\sqrt{A_1^2 + A_2^2 - 2A_{12}A_1A_2}}{\sqrt{1 - A_{12}^2}},$$
 while when the scales

of the two kernels are normalized, meaning  $|K_1| = |K_2|$ , the target alignment of equally weighted kernel

$$A(1) = \frac{A_1 + A_2}{\sqrt{2 + 2A_{12}}}, \text{ and } A(w_{opt})^2 - A(1)^2 = \frac{(A_1 - A_2)^2}{2(1 - A_{12})}.$$
 This

indicates that the difference between optimally weighted kernel and equally weighted kernel becomes large when the ratio of their target alignment increases.

The above arguments suggest a better kernel function could result from a re-composition of the original kernel function as an appropriately weighted set of kernels. In our approach, specifying component kernels according to stroke-level features serves as a mechanism to decompose the conventional polynomial kernel function.

## 5.2. High-level Feature Alignment

Without prior knowledge to guide the processes, a decomposition is unlikely to produce kernel functions with significantly different alignments. On the other hand, knowledge of high-level features can help to group input features according to their information content. In particular, if we assume the high-level features have binary values, then we can define the alignment between a high-level feature label and a given kernel matrix. We call this its *high-level feature alignment*. We use  $A_F(\cdot)$  to denote alignment of a kernel to high-level feature, and  $A_t(\cdot)$  to denote alignment of a kernel to the target label. For a high-level feature that is unique for its class, its label has perfect alignment with the class label. Therefore, the corresponding high-level feature alignment equals the target alignment, which means  $A_F(K) = A_t(K)$ . On the other hand, high alignment to a non-informative high-level feature results, in a low alignment to the class label.

Now we examine those input features  $x$  that exhibit high alignment with some high-level feature  $f$ . Assume

we have binary input features  $x = \{0, 1\}$ , then:

$$\begin{aligned} A_F(x) &= \frac{\sum_{i,j} x_i x_j \cdot f_i f_j}{\sqrt{\sum_i x_i^2 \cdot f_i^2} \cdot \sqrt{\sum_j x_j^2 \cdot f_j^2}} = \frac{\sum_i x_i f_i \cdot \sum_i x_i f_i}{\sqrt{\sum_i x_i^2} \cdot \sqrt{\sum_j x_j^2} \cdot \sqrt{\sum_j f_j^2}} \\ &= \frac{(\sum_i x_i f_i)^2}{\sum_i x_i^2 \cdot \sum_j f_j^2} = \frac{(\sum_{f_i=1} x_i - \sum_{f_i=-1} x_i)^2}{N(x_i=1) \cdot \sum_j f_j^2} = \frac{\left( \frac{\sum_{f_i=1} x_i}{n} - \frac{\sum_{f_i=-1} x_i}{n} \right)^2}{\left( \frac{N(x_i=1)}{n} \right) \cdot \sum_j f_j^2} \\ &= \frac{(\Pr(x=1, f=1) - \Pr(x=1, f=-1))^2}{\Pr(x=1)} \end{aligned}$$

In the above equation,  $\Pr(x=1, f=1) - \Pr(x=1, f=-1)$  represents the correlation of  $x$  with high-level feature  $f$ . Those input features that exhibit high correlation with  $f$  can be used to reliably detect high-level feature  $f$ . In our experimental section, we use such correlation to determine evidence pixel set for each stroke, and specify component kernel functions.

## 6. Empirical Results

### 6.1. Experiment Setup

In this section, we present our empirical results to demonstrate the accuracy improvement made by incorporating stroke-level knowledge into kernel functions for SVM learning algorithms. In our experiments, we used handwritten Chinese image examples from the ETL9B database [8] created by Electrotechnical Laboratory of Japan. It contains 200 samples for each character. All examples are binary images of size 64 by 64.

We used *libsvm* package as our SVM learning algorithm. To simplify the comparison of kernel functions, we used the default value for all the SVM parameters, except those for kernel functions, during the learning.

We want to compare our specialized kernel function with a conventional polynomial kernel function. We have found that the optimal degree of a conventional polynomial kernel for the ETL9B dataset is 3. To make a fair comparison, we also used third degree polynomials to define component kernel functions and, therefore, the derived feature kernel functions.

### 6.2. Experiment 1: Does a feature kernel outperform a similar conventional kernel?

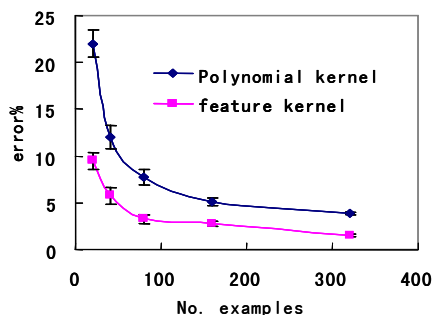
The first question we ask is following: Do feature kernel functions offer a significant improvement over similar conventional polynomial kernel functions when the same order polynomial is applied?

**6.2.1. Experiment 1a: Behavior on a single challenging discrimination task.** To compare two kernel functions, we first selected two very similar Chinese characters (shown in Figure 1) that represent a challenging discrimination task. We believe this sort of challenging task best illustrates the benefit of domain knowledge. These two characters are quite similar, which means it should be possible to decompose a conventional kernel function into informative ones and non-informative ones.

We constructed the component kernel functions for all the stroke-level features used in the explaining two characters. A feature kernel function was constructed by weighting those component kernels. It and a conventional third-order polynomial kernel function were evaluated using a leave-one-out test to compare their performance. The number of errors, average number of support vectors and the values of the empirical target kernel alignment are given in Table 1. It is clear that, as predicted, the SVM using specialized feature kernel exhibits significantly better performance than the conventional kernel. It also results in fewer support vectors, which demonstrates that the feature kernel emphasizes few, but reliable features. This is also supported by the comparison of target kernel alignments for two kernels.

**Table 1.** Results on leave-one-out test for classifying two characters using different kernels.

KERNEL	ERROR	NO. SV	ALIGNMENT
FEATURE	4	145	0.1956
POLYNOMIAL	15	197	0.1279

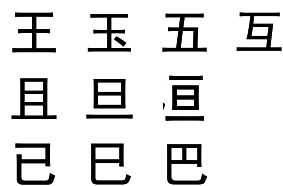


**Figure 3.** Feature kernel improves SVM accuracy.

Figure 3 shows the efficiency of learning. It plots the error rate with at different stages of training for the two kernels. The feature kernel is significantly better than the polynomial kernel in all cases. Notice that, with 80 training examples, the accuracy of the feature kernel achieves is similar to that of the polynomial kernel with 320 training examples. This can be a

crucial improvement when training examples are limited or expensive to produce.

**6.1.2. Experiment 1b: Behavior on a representative set of Chinese characters.** Experiment 1a shows that the adapted feature kernel function offers significant improvement for classifying two target characters chosen to illustrate its benefits. Is there something specific about these two characters that makes them somehow uniquely liable to feature kernel functions, or the results can generalize to other characters? In particular, does the presence of the domain theory degrade performance on antithetical classification problems which are less challenging?

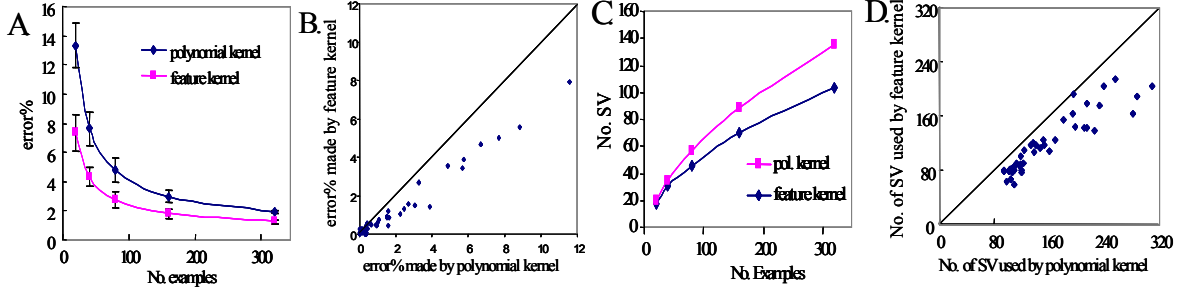


**Figure 4.** Ten characters used in the experiments.

To answer this question, we selected a set of ten Chinese characters, shown in figure 4. They are drawn from three different radical classes, and form 45 pairwise discrimination problems. 16 of them are challenging discrimination problems representing characters within the same radical group. Others are easier classification decisions from between radical groups.

Again we compared feature kernel function against polynomial kernel functions for each classifier. Figure 5 shows the results in two different ways: 1). Figure 5A and 5C plot average error rate and average number of the training examples for the two kernels; 2). Figure 5B and 5D show the comparison in all tasks using a leave-one-out test with 400 examples. We see in almost all cases, across different difficulty-levels and for different number of training examples, that the SVM with feature kernels outperform the one with polynomial kernels. The SVMs built by feature kernels achieve varying degree of improvements on accuracy, but consistently use fewer support vectors.

**6.1.3. Experiment 1b re-analysis.** What property of the task affects the improvement for specializing feature kernel functions? We believe that additional information in the form of prior domain knowledge may differentially help difficult classification tasks. To investigate, we reanalyzed the data of Figure 5A: We evenly divided the 45 classification tasks into three groups according to whether a task easy, medium, or



**Figure 5.** Comparison of the accuracy and number of SV used for two kernel functions.

difficult for the conventional SVM learner as judged by the number of errors made by the conventional SVM. Figure 6 shows the results: not only does the feature kernel function help, but it indeed appears to help most in the most difficult classification problems.

Finally, we compared multi-class recognition performance in our 10-character problem. We used 160 examples for each character to form the training set, while the remaining 40 examples of each character are held back to form a test set. The result is that the classifiers built with conventional polynomial kernel functions, in this case, have 88.7% recognition accuracy, while the feature kernel functions have 92.6% recognition accuracy.

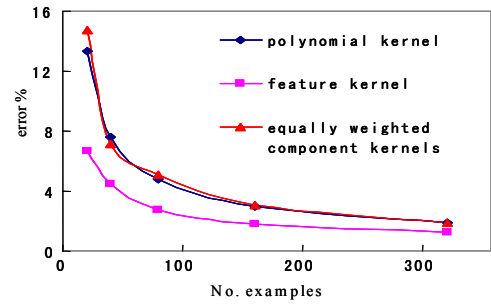
## 6.2. Experiment 2: How important is the EBL interaction between knowledge and examples?

The experiments and analyses of our first investigation provide compelling evidence that while the feature kernel function is significantly less expressive than the conventional kernel function, its performance is better. We attribute this improvement to the EBL approach. But is this interpretation fair? Perhaps it is the presence of the additional knowledge that is significant and not the EBL interaction between domain knowledge and training examples.

To test whether the EBL paradigm underlies the improvement (as we believe) or it is merely the presence of the additional high-level information, we designed a control experiment where the component kernel functions are constructed as before, but the feature kernel function is constructed without access to the training examples. The difference between such a kernel function and the weighted kernel function is that it misses the evaluation and adjustment of the conclusions drawn from the prior knowledge using training data. When this interaction is denied to the system, the prior knowledge is simply treated as correct

(as in conventional EBL). Figure 7 compares this new kernel function with both the conventional polynomial

kernel and previous feature kernel function. The results clearly show the importance of using data to guide the EBL system towards significant and useful conclusions.



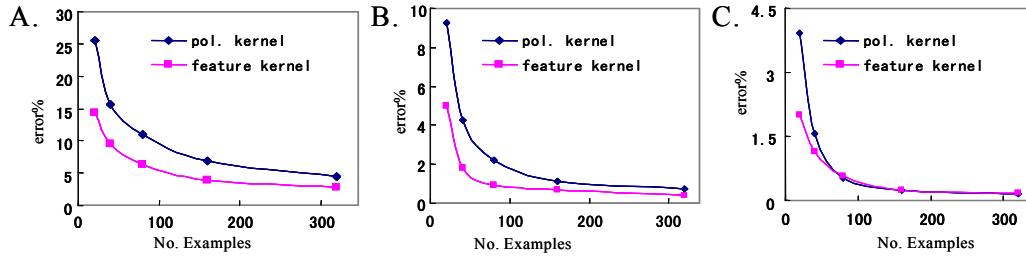
**Figure 7.** Interaction between knowledge and examples provides improvement.

## 6.3. Experiment 3: Can other representations also benefit from our approach?

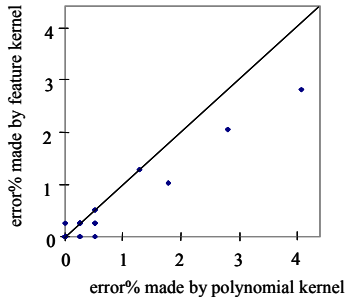
The final question we asked is: Can our approach add anything to alternative, maybe more specialized representations? One of such representation is based on gradient, instead of darkness, of each pixel, as studied by Dong et al [9]. We believe the success of our approach comes from the high-level knowledge of strokes. This knowledge should hold whether we use pixel features or gradient features to detect the stroke features. Therefore, our principle should also apply for their gradient representation.

To test this, we followed [9] to build gradient-based representations for our 10-character database. We used the same set of stroke-level features to design gradient-based component kernel functions. In particular, the gradient-based representations are calculated for each stroke by only using gradient information in the region for that stroke. If we use  $g_A$  and  $g_B$  to represent gradient-based evidence set for stroke  $A$  and  $B$  respectively, then, the component kernel function for the corner shown in Figure 2 becomes:

$$k_{A,B}(x_1, x_2) = \left( \sum_{i \in g_A} x_{1i} x_{2i} \right) \left( \sum_{j \in g_B} x_{1j} x_{2j} \right)$$



**Figure 6.** Comparison the accuracy used for two kernel functions by three groups of tasks. A. Group of the most difficult tasks. B. Group of the medium difficult tasks. C. Group of the easiest tasks.



**Figure 8.** A scatter plot comparing the leave-one-out errors using gradient-based representation.

With gradient-based component kernel functions, weighing and assembling them to form a gradient-based feature kernel function is the same as pixel-based. Figure 8 shows the comparison of our specialized gradient-based feature kernel function with a conventional gradient-based kernel for all 45 binary classification problems. In most of cases, our feature kernel functions achieved improvements.

## 7. Conclusion

In this paper, we propose an explanation-based approach to incorporating approximate and imprecise high-level domain knowledge into the SVM learning process. A specialized kernel function is formed that employs novel high-level features. These synthesized features result from the interaction of the information contained in the prior domain knowledge and information embodied in the training set. Prior knowledge helps to interpret the training examples so as to discover their robust features and the training data guide the use of the prior domain knowledge, focusing attention on those features that prove most informative as judged with the training data. The approach points a new direction to distinguish other complex stroke images such as diagrams and drawings. We believe that with suitable domain knowledge the approach may also apply to complex classification tasks beyond characters and line drawings.

## Acknowledgment

This material is based upon work supported by the National Science Foundation under Award NSF IIS 04-13161. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

## References

- [1]. Schölkopf, B., & Smola, A. J. (2002). *Learning With Kernels*, Cambridge, MA: MIT Press.
- [2]. DeJong, G. (2004) *Explanation-Based Learning*. In A. Tucker (Editor-in-Chief), *Computer Science Handbook*, second ed., Chapman and Hall/CRC and ACM. p.68.1- 68.18
- [3]. Russell, S. J. & Norvig, P. (1995). *Artificial Intelligence. A Modern Approach*, Prentice-Hall.
- [4]. Ponce, J., & Forsyth, D. A. (2002). *Computer Vision: A Modern Approach*. Prentice Hall.
- [5]. Cristianini, N., Shawe-Taylor, J., Elisseeff, A., & Kandola, J. (2002). *On Kernel-Target Alignment*. in *Advances in Neural Information Processing Systems (NIPS)* 14, 367-373.
- [6]. Ong, C. S., Smola, & A. J. (2003). *Machine Learning with Hyperkernels*. *Machine Learning, Proceedings of the Twentieth International Conference(ICML)*. 568-575.
- [7]. Kandola, J., Shawe-Taylor, J., & Cristianini, N. (2002). *Optimizing kernel alignment over combinations of kernels*. *NeuroCOLT Technical Report NC-TR-02-121*
- [8]. Saito, T., Yamada, H. & Yamamoto K. (1985). *On the data base ETL 9 of handprinted characters in JIS Chinese characters and its analysis*. *IEICE Transactions*, J68-D(4):757--764.
- [9]. Dong, J. X., Krzyzak, A., & Suen, C.Y. (2003) *High accuracy handwritten Chinese character recognition using support vector machine*. *Proceedings of International Workshop on Artificial Neural Networks on Pattern Recognition*.