USE OF PRIOR KNOWLEDGE IN CLASSIFICATION OF SIMILAR AND STRUCTURED OBJECTS

BY

LI-LUN WANG

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Doctoral Committee:

Professor Gerald DeJong, Chair
Professor Dan Roth
Assistant Professor Derek Hoiem
Assistant Professor Paris Smaragdis
Doctor Qiang Sun, Alexa Internet, Inc.

Abstract

Statistical machine learning has achieved great success in many fields in the last few decades. However, there remain classification problems that computers still struggle to match human performance. Many such problems share the same properties—large within class variability and complex structure in the examples, which is often true for real world objects. This does not mean lack of information for classification in the examples. On the contrary, there is still a clear pattern in the examples, but hidden behind a many-way covariance structure such that useful information is too dilute for conventional statistical machine learners to pick up. However, if we can exploit the structural nature of the objects and concentrate information about the classification, the problem can become much easier. In this dissertation we propose a framework using prior knowledge about modeling the structures in the examples to concentrate information for classification. The framework is instantiated to the task of classifying pairs of similar offline handwritten Chinese characters. We empirically demonstrate that our proposed framework indeed concentrates useful information for classification and makes the classification problem easier for statistical learning. Our ap-

proach advances the state of the art both in offline handwritten character recognition and in machine learning.

Table of Contents

Chapter 1

Introduction

1.1 Challenging Classification Problems

Classification using statistical machine learning has achieved great success in many fields in the last few decades. However, there remain problems that the computer still struggles to match the human's performance. One example of such problems is pedestrian detection. One of the best approaches is only able to achieve a 60% recall at 1 false positive per image for unoccluded pedestrians over 80 pixels high [7], or 80% recall at 1 false positive per image on the dataset it is trained on [5]. Another example of classification problem that is difficult for computers is the classification between similar objects, such as distinguishing images of ketches vs. schooners [16].

Consider, as a more detailed example, the problem of classifying cars of different brands in a clear and unoccluded image. This is different from the extensively studied problem of recognizing cars in images. In the problem of recognizing cars in images, we collect evidence of whether a car is present in

the image, e.g. by classifying patches of the image, and decide whether the image contains a car. In the problem of classifying cars of different brands, however, we know there is a car in the image. We want to, through looking at the various details of the car, decide the brand of the car.

Because cars share very similar structures, this is a much harder problem. Different images of the cars can show a lot of variability due to the different backgrounds of the images, the slight change in camera angles, the lighting, different dents and dings through out the years, dirtiness of the cars, etc. The difference in appearance between different brands of cars is very much diluted by this variability. However, cars are highly structured objects. If we are able to make use of their structures and locate the right parts of the cars, say the hood ornament or the logo, the relatively subtle difference at this specific part of the image makes the classification almost trivial.

These difficult classification problems share the same property—large within-class variability and complex structure, which is often true for real world objects. The discriminating information in the examples for class labels is too dilute for the statistical machine learner to pick up. However, if we can exploit the structure of the object and concentrate the information by focusing on the right part of the object, the classification can be made much easier.

1.2 Exploiting the Structure in Objects

We propose a way of exploiting structures in objects to benefit classification through the use of prior domain knowledge on modeling the structure,

along with a few *structurally labeled* training examples. These structurally labeled examples construct and calibrate the structural models, and help in concentrating information for classification.

The structural models are a sequence of generative models, each of which models a sub-structure of the object. Structures in the objects are represented as variables in the structural models, which we call structural features. We evaluate the structural features by fitting the structural models to the objects.

After we find the structures in the object by evaluating the structural features, the target feature model designates part of the object according to the structures that bears the most information for classification, and uses it to generate discriminative features. These discriminative features contain concentrated information for classification, resulting in a much easier classification problem for the classifier.

Prior domain knowledge specifies how the structural models and the target feature model are constructed in terms of basic constituents of the objects, as well as how the basic constituents are modeled. It also specifies how information of an estimated sub-structure in the object can be subtracted from its representation. The structurally labeled examples are examples with their basic constituents labeled.

The structural model can be regarded as a learning bias automatically constructed for the given task. Figure 1.1 outlines the learning of this approach. Compare to the conventional machine learning in Figure 1.2. The conventional machine learning relies on having a static learning bias that is just right for the task, a flexible enough statistical learner, and a very

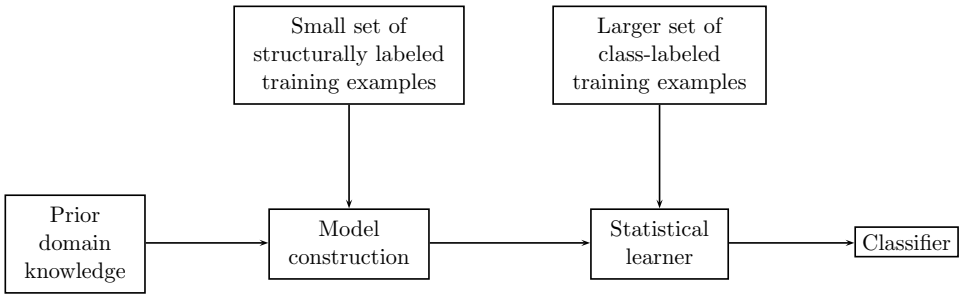


Figure 1.1: Simplified outline of the learning by exploiting the structures in objects. The output of model construction serves as a very strong learning bias to the statistical learner. Compare to the conventional machine learning in Figure 1.2.

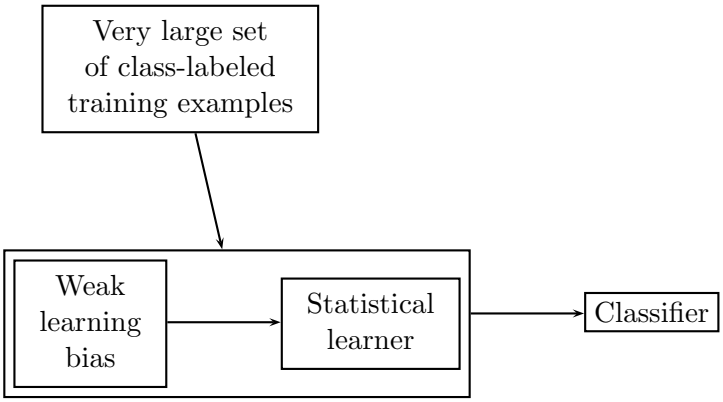


Figure 1.2: Outline of the conventional machine learning.

large set of class-labeled training examples. On the other hand, the output of the model construction in our approach serves as a very strong learning bias, which may not be as high quality as human-supplied bias, but is automatically generated to fit all problems in the same domain.

Figure 1.3a shows the detailed block diagram for doing classification using our proposed approach. Given an unknown test example, we apply the structural models and evaluate the structural features in the test example.

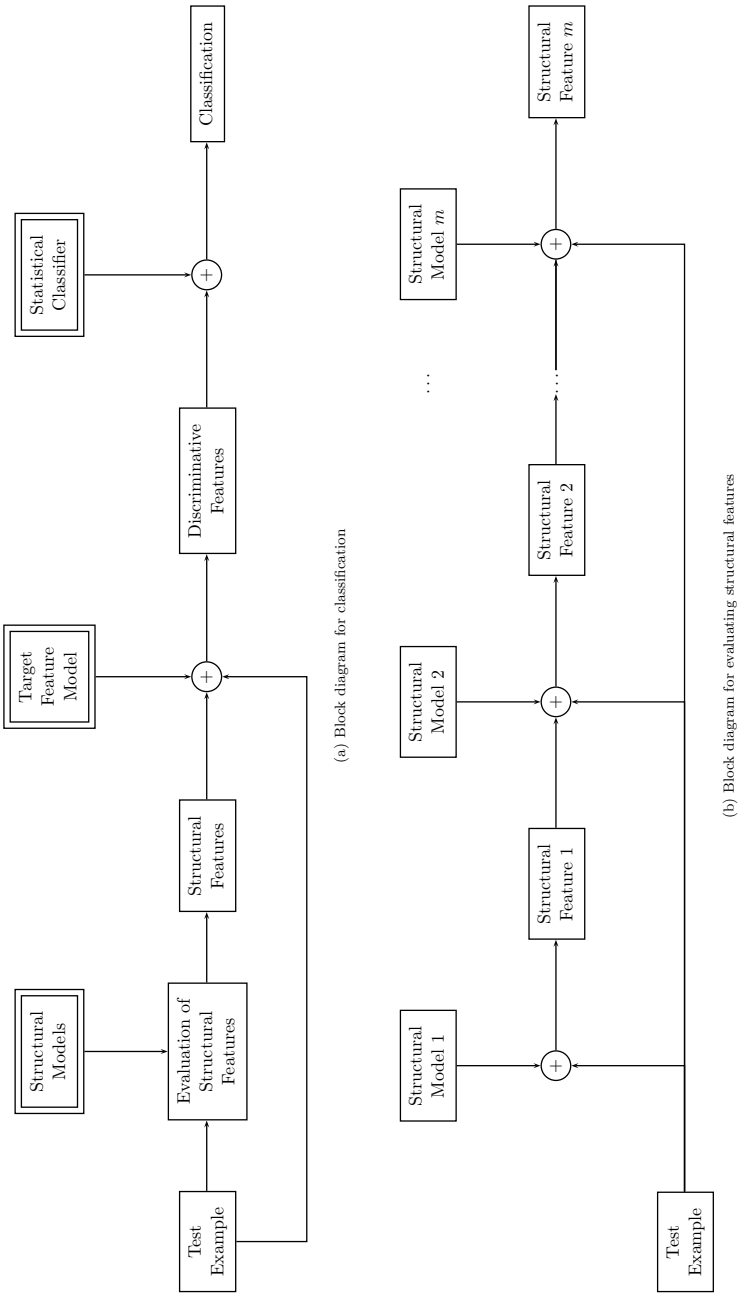


Figure 1.3: Block diagram for exploiting structures in examples to benefit the classification of similar objects. Double-boxed items in Figure 1.3a are components that need to be learned. Figure 1.3b details the evaluation of structural features in Figure 1.3a.

With the evaluated structural features, we then generate discriminative features with high concentration of information about the classification. These discriminative features are then fed into a statistical classifier for to decide the class label.

Evaluation of structural features in this process is in the form of a series of model fittings, and is shown in Figure 1.3b. With a generative model of part of the structure, we find the set of parameters for the model that best fits the test example and parameters of previous model fittings.

In Figure 1.3, the structural model, the target feature model, and the statistical classifier are components that need to be learned in our approach. Figure 1.4 shows the detailed block diagram for learning these models. Given the prior domain knowledge about modeling the structures, along with the structurally labeled training examples, we infer the structural correspondences between the two classes of objects. Structures that do not have correspondences in the other class are used in constructing target features. We then hierarchically decompose the structures, and construct structural models for evaluating structural features. Given the set of class-labeled training examples, we use the learned structural models to evaluate their structural features, and use the target feature model to generate discriminative features. The statistical machine learner is then applied to the discriminative features to learn the statistical classifier.

We choose the task of classifying similar offline handwritten Chinese characters as our illustrative domain. Note that when applying our approach to offline handwritten Chinese characters, the same domain knowledge about modeling the Chinese characters is used across classification problems of

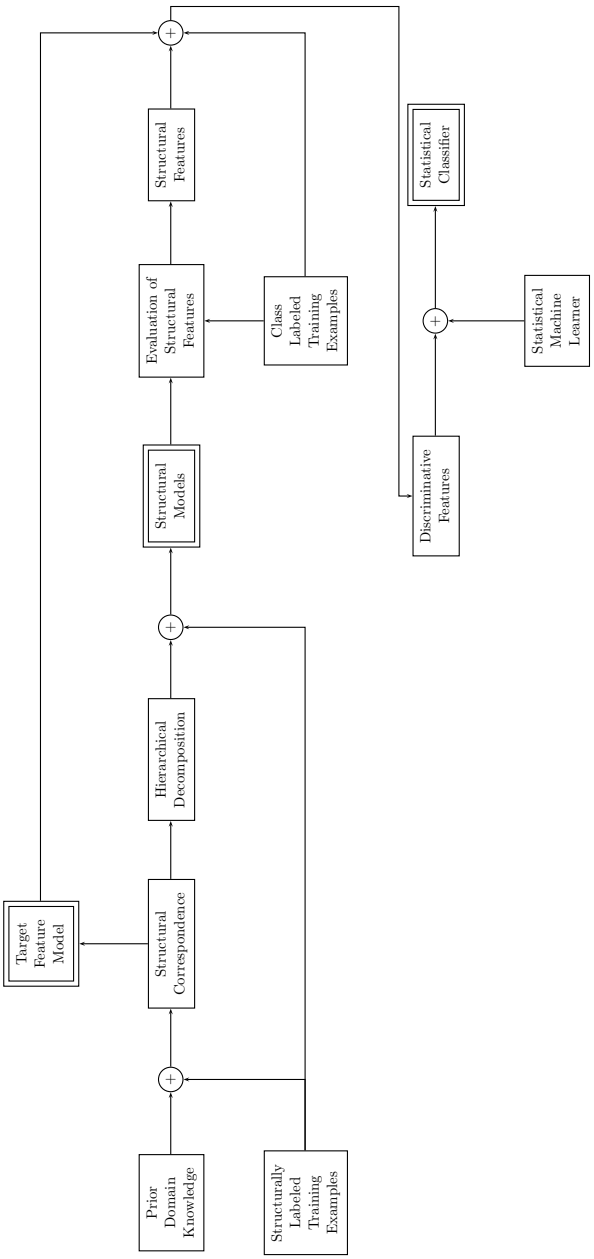


Figure 1.4: Block diagram for learning the models in Figure 1.3. Double-boxed items are the learned components.

different pairs of characters of varying difficulty. The resulting structural models are constructed using structurally labeled training examples for the specific problem and the same domain knowledge.

1.3 Classifying Similar Offline Handwritten Chinese Characters

In the following chapters, we instantiate the framework described in Section 1.2 to the problem of classifying pairs of similar offline handwritten Chinese characters.

Offline handwritten Chinese character recognition is an extensively studied problem. Over the years the most mature and popular set of features for use in Chinese character recognition remains the gradient-based features [14] [13] [20] [6]. Although overall recognition accuracies of over 99% have been reported on certain data sets [13] [32], this is largely due to the fact that most Chinese characters look very different from each other. There are still pairs of similar Chinese characters that the best recognition systems perform significantly worse than humans. We are interested in classifying such pairs of characters using our approach. Figure 1.5 lists some pairs of similar handwritten Chinese characters that existing recognition systems have trouble on.

Figure 1.6 illustrates how we concentrate information for classification in a pair of similar Chinese characters. Crucial information for classification in this pair is in the middle box radical. In order to find structures in the character image, we first construct templates of parts of the character

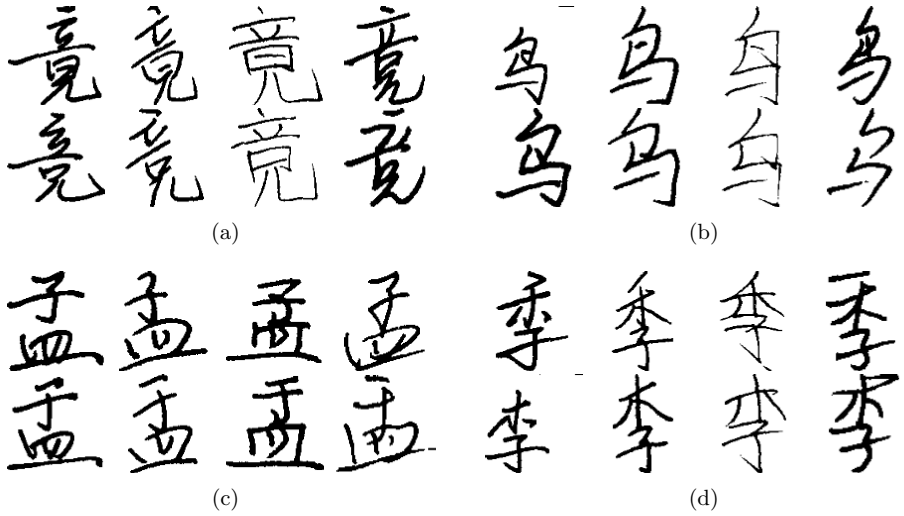


Figure 1.5: Examples of similar pairs of handwritten Chinese character images that existing recognition systems have trouble on.

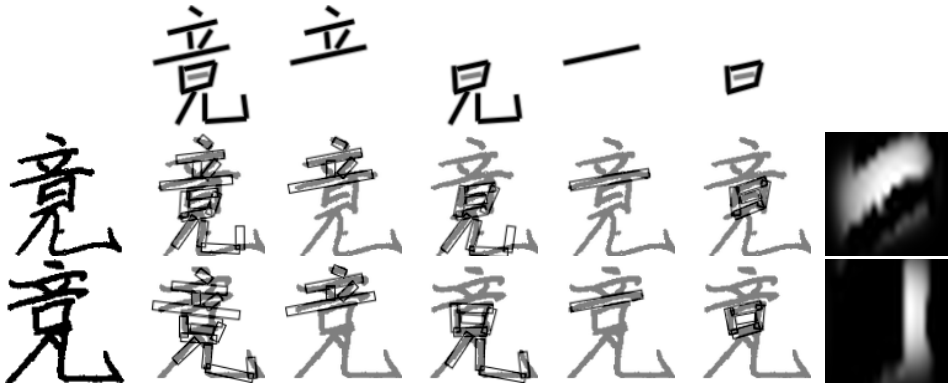


Figure 1.6: Concentrating information for classification through the use of a sequence of template matching operations. Structures in the character images are found by matching a sequence of affine-transformed templates to the image. The sequence of match operations allows for both convexity and accuracy in estimating the structures.

consisting of strokes. We then apply affine transformations and match the templates to the character image. The affine transformation parameters are our structural features. After we find the structures, or stroke locations, in the character image, we extract a region around the middle box from the image, and generate discriminative features from the extracted region. This target region contains concentrated information for classification. Note that the whole character template at the beginning of the sequence results in the least accurate overall match of the strokes, and each subsequent smaller template refines the match of a sub-structure. While larger templates results in less accurate matches, the objective function for matching the template is more convex with larger templates. On the other hand, while smaller templates can produce more accurate template matches, there are more local optima in the objective function. Using a sequence of match operations as in Figure 1.6 we achieve both convexity and accuracy of the match. This sequence of match operations are our structural models.

Figure 1.7 shows a target learning curve that we want to achieve using our approach. This learning curve is for the pair 季 vs. 李, which is one of the pairs that our approach does relatively well on. The LDA baseline is the accuracy of classifying examples from this pair of classes using the multiclass LDA classifier training on all classes of characters. The learning curve for our approach is steeper than that of the SVM for this pair, and achieves a better accuracy than all baseline accuracies except the human baseline. However, we do not think this trend will always continue. Because of the strong bias in our approach, with enough training examples, the conventional purely statistical approach will eventually surpass the accuracy of our approach.

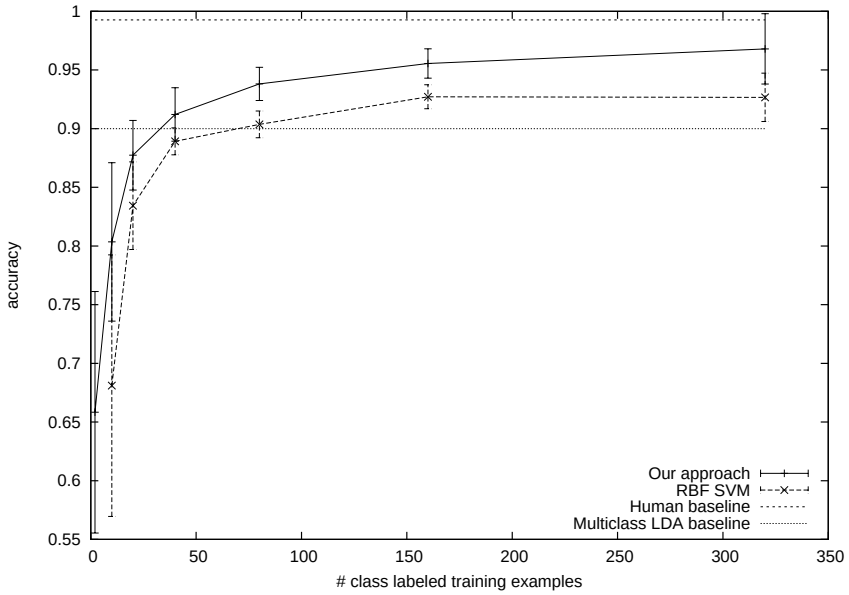


Figure 1.7: The goal learning curve that we want to achieve.

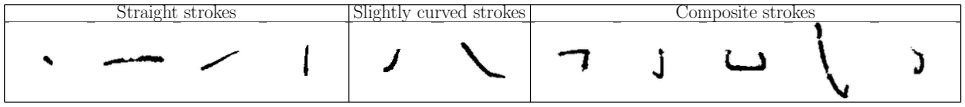
Several components need to be implemented to instantiate our framework to the problem of classifying similar offline handwritten Chinese characters:

- **Prior knowledge for modeling**

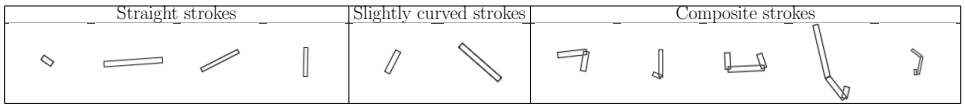
  We model a Chinese character as a set of simple long and thin rectangular strokes. Figure 1.8 lists some common stroke types and the use of long rectangles to model them. Each stroke is parameterized using the coordinates of its two end points and its width. Modeling of Chinese characters is detailed in Section 2.1.1.

- **Structural models**

  Structures in the character are specified using joint configurations of



(a) Actual strokes in images of Chinese characters



(b) Modeled strokes using long and thin rectangles

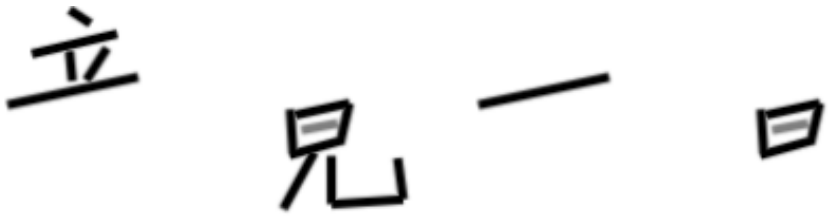
Figure 1.8: Use of long and thin rectangles to model strokes in Chinese characters.

(a) (b) (c) (d)

Figure 1.9: Character templates for pairs of characters in Figure 1.5. Strokes specific to each character class are rendered with half intensity.

sets of strokes. Sets of highly correlated strokes form *templates* of portions of the character. These templates are our structural models. Figure 1.9 shows the whole character templates for the pairs of characters in Figure 1.5, and Figure 1.10 shows examples of templates for some sub-structures of the pair of characters in Figure 1.5a. Note that our approach is not limited to using radicals of characters as the templates.

We use the affine transformation parameters of a template as a reduced representation of the structure. Figure 1.11 shows examples of affine transformed templates. This is detailed in Section 2.2.



(a) (b) (c) (d)

Figure 1.10: Examples of templates of sub-structures for the character in Figure 1.9a.

Construction of the series of structural models to use in our approach is described in Section 2.6.2 and Section 2.6.3.

- **Evaluation of structural features**

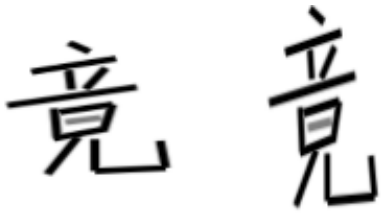
  Evaluation of the structural features is done by matching the templates to the character image, and is described in Section 2.3 and Section 2.4. Figure 1.12 shows the matching of the whole character template to the character image and the matching of a radical template to the character image. We use a gradient-based optimization to match the templates. Section 2.5 discusses in detail the importance of initializing the optimization to the right place.

- **Target feature model**

  Strokes that do not have correspondences in the other class are treated as target strokes. Section 2.6.1 talks about how stroke correspondences are found.

- **Discriminative features**

  We use gradient-based features around the target region as our dis-



$$M = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}$$

(a) Scaling

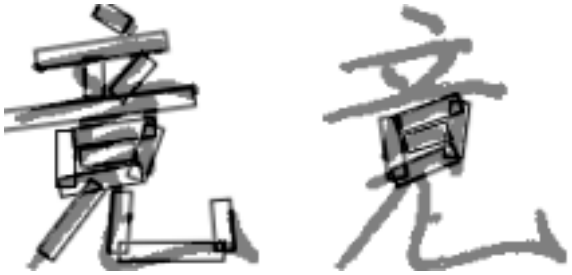
$$M = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

(b) Rotation

$$M = \begin{pmatrix} 1 & k_1 \\ k_2 & 1 \end{pmatrix}$$

(c) Shearing

Figure 1.11: Affine transformed templates and their transformation matrices.



(a) Matching the whole character

(b) Matching the middle radical

Figure 1.12: Matching of templates to a character image.

(a) Original character image

(b) Target region

Figure 1.13: The target region for a character image.

criminative features for the statistical classifier. Section 2.7 details how the target region is prepared and how the gradient features are generated. Figure 1.13 shows the target region for a character image.

- **Statistical machine learner**

  We use the support vector machine with the radial basis function kernel as our statistical machine learner.

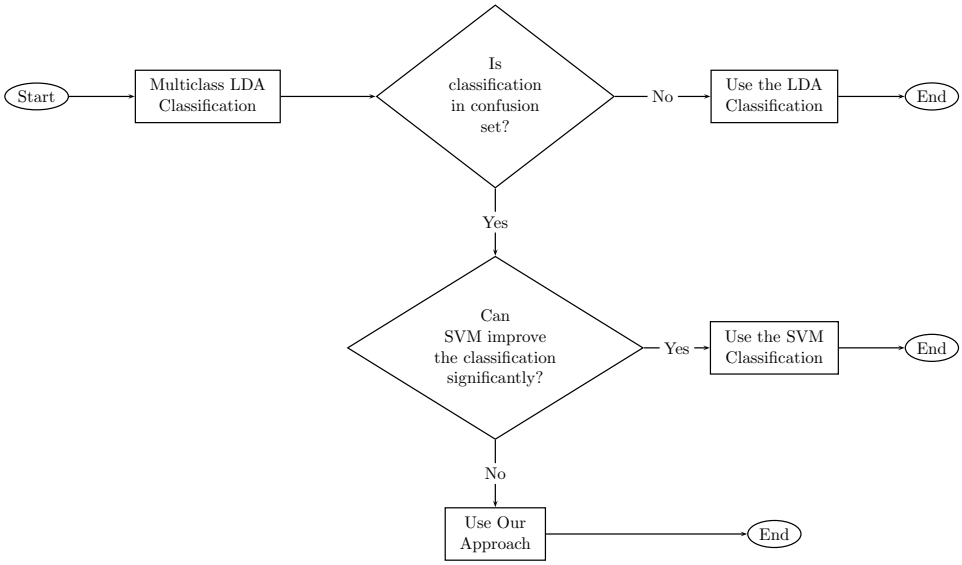


Figure 1.14: A full Chinese character recognition system that incorporates our approach and a multiclass Chinese character recognition system.

1.4 A Full Offline Handwritten Chinese Character Recognition System

Although we focus on the task of pairwise classification of difficult Chinese characters, it is easy to integrate our approach into a full Chinese character recognition system.

Figure 1.14 outlines this full Chinese character recognition system. We describe this system in detail in Chapter 5.

Given any multiclass Chinese character recognition system such as the multiclass linear discriminant-based system in [14] and a training data set, we use cross-validation to find the confusion set that the system performs the worst on. The confusion set will be small, as most Chinese characters look very different from each other. Among these most confusing characters,

the recognition accuracies of some characters can be significantly improved by using a support vector machine. We identify these characters as well.

During testing, we first apply the multiclass Chinese character recognition system on the test example. If the classification is not among one of the most confusing characters, the output of the multiclass recognition system is used as the classification. If the classification is among those that can be significantly improved by using the SVM, we use the SVM classification. Otherwise, we apply our approach on the test example.

1.5 Related Work

Chinese characters are made of radicals, and it is natural to recognize the character by looking for its radicals. Ni et al. [21] proposed a way of extracting radicals from Chinese characters. However, they fall short of recognizing the character after extracting the radicals. Our approach also differs from theirs in that we are not restricted to using pre-defined radicals of Chinese characters, but we are free to use any set of highly correlated strokes in our system.

There have been other approaches in classifying structured objects by finding its parts. Endres et al. [9] use shared body plans to learn to recognize several related classes of objects. Felzenszwalb et al. [10] use mixtures of multiscale deformable part models to recognize different objects in images. Karlinsky et al. [12] propose the chains model that detects specific target parts of the object by creating and evaluating feature chains between known reference points on the target and the target parts. Our approach differs

from these approaches in that we focus on the important part of the object based on the estimated structures in the object to distinguish extremely similar objects.

Duan et al. [8] also deal with the classification problem of closely related, or structurally similar, categories of objects by finding discriminative local features. However, they do not build models to estimate the structures. Instead of making use of the structure of the object to find the discriminative region, they rely on hierarchical segmentation to produce regions at different scales, and learn a latent CRF to choose a region for each attribute in each image. Classification and feature detection become inference problems on the same CRF.

Besides computer vision applications, structure prediction is also useful in many other fields of machine learning. Roth et al. [23, 3] introduced a general framework for constrained optimization, the *constrained conditional models*, that supports incorporating declarative knowledge into statistical models. While there is much similarity in incorporating prior knowledge into statistical models, instead of predicting the structured output, we focus on using the predicted structure to concentrate information that ultimately helps classification.

Chapter 2

Knowledge-guided Classification

2.1 Modeling Offline Chinese Characters

2.1.1 Modeling the strokes

A Chinese character is composed of one or more radicals arranged in a particular way. These radicals are portions of characters shared among different characters. Figure 2.1 shows some examples of offline Chinese character images, as well as some cases of shared radicals. The composition of radicals in Chinese characters forms a hierarchical structure, providing us a natural way to divide the image of a character into sub-parts. Figure 2.2 shows an example of a more complex character and one way of decomposing it into sub-parts. Note that this is merely one of the many possible ways of decomposing the character. There are an exponential number of possible ways of

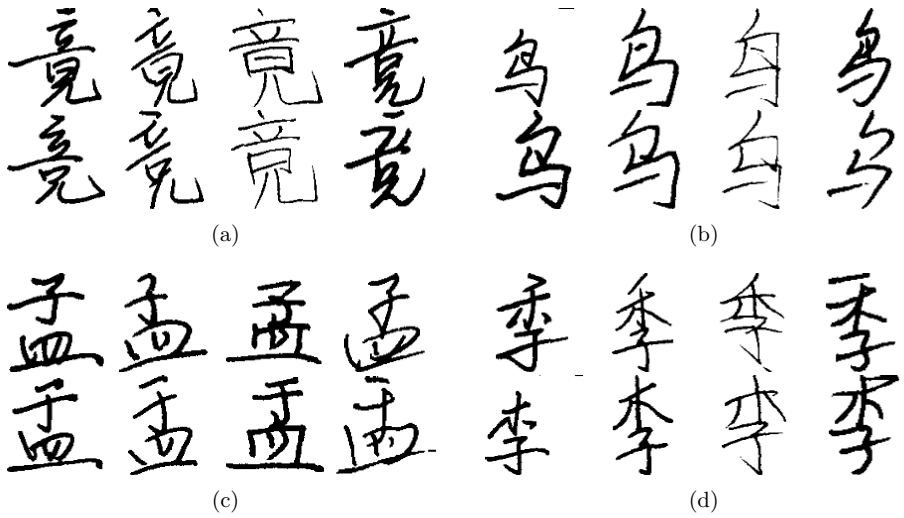


Figure 2.1: Examples of offline Chinese character images. In Figure 2.1a, the top radical is shared between the two classes. In Figure 2.1c, the bottom radical is shared between the two classes. The top radical of the top class in Figure 2.1c and the bottom radical in Figure 2.1d are also shared.

decomposing a character, the same as the number of possible tree structures given the strokes as leaves.

The radicals in Chinese characters are made of strokes. As can be seen in Figure 2.1, the actual appearance of the strokes in a character image is almost never perfect lines, and varies according to the writer, the writing instrument, and the discretization of the image. The same character made by different writers can look very different, while similar characters made by the same writer can look very similar.

There are a few common types of strokes in Chinese characters. Most strokes are straight lines of various lengths and orientations. Other stroke types include slightly curved lines and bent lines. Figure 2.3 lists some common types of strokes.

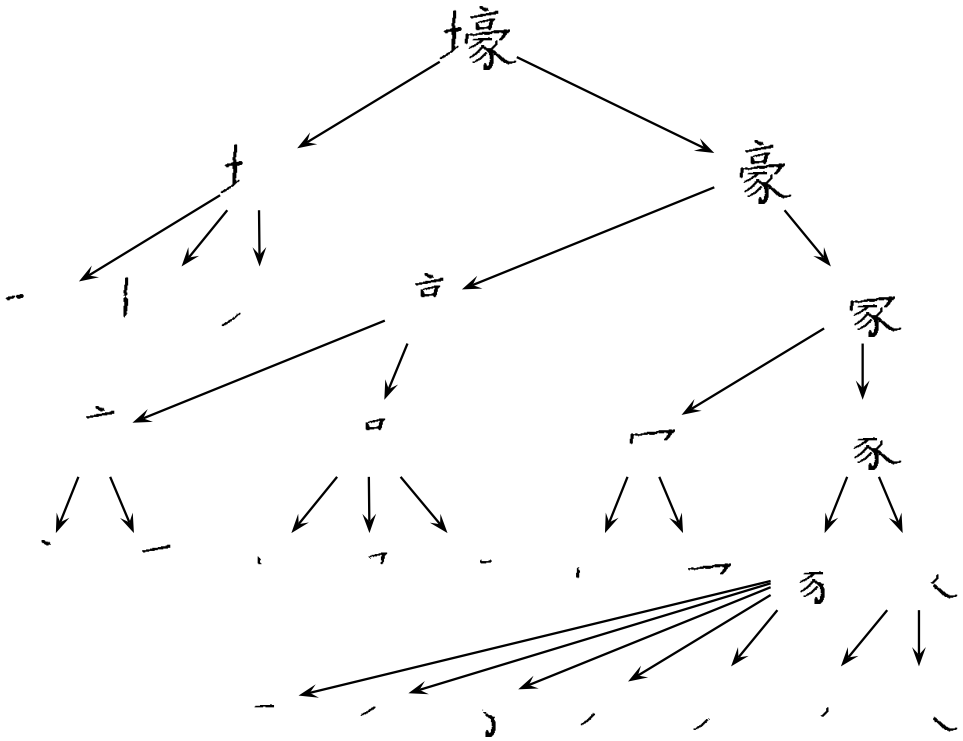
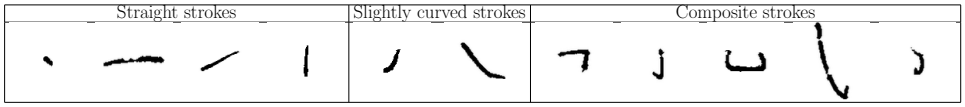
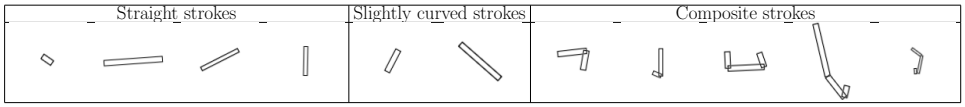


Figure 2.2: One way of hierarchically decomposing a character.



(a) Actual strokes in images of Chinese characters



(b) Modeled strokes using long and thin rectangles

Figure 2.3: Examples of strokes of different types in Chinese characters.

Despite the noisy appearance of the strokes in Chinese characters, we model them as simple long and thin rectangles, as shown in Figure 2.3b. These long rectangles are parameterized using the coordinates of their two end points, along with the width. The slightly curved strokes are treated as straight strokes, and modeled as long rectangles as well. The bent stroke is treated as a composite stroke with straight line segments joined at the end points, and modeled as several long rectangles separately for each segment. Each long rectangle is represented as a vector of 5 parameters:

$$\Psi = \langle x_1, y_1, x_2, y_2, w \rangle, \tag{2.1}$$

where (x_1, y_1) and (x_2, y_2) are the coordinates of the two end points, and w is the stroke width.

2.1.2 Labeling the strokes in training examples

Figure 2.4 shows some images of actual Chinese characters, along with their rectangular approximations. Each example is normalized and re-registered to a 100×100 image without changing the aspect ratio of the character. We generate the structurally labeled training examples by marking the two end points of the strokes. The stroke width is approximated by repeatedly thinning the strokes until 80% of the stroke pixels in the image are accounted for, and is fixed across all strokes within the same instance of the character. Therefore, a character with l stroke segments is represented as a vector of $4l + 1$ numbers, with 4 parameters representing the two end points of each

stroke, and one parameter shared across every stroke for their width:

$$\begin{aligned}
\Gamma &= \langle \Psi_1, \Psi_2, \ldots, \Psi_l \rangle && (2.2)\\
&= \langle \langle x_{1,1}, y_{1,1}, x_{2,1}, y_{2,1}, w \rangle, \langle x_{1,2}, y_{1,2}, x_{2,2}, y_{2,2}, w \rangle, \ldots, \\
&\quad \langle x_{1,l}, y_{1,l}, x_{2,l}, y_{2,l}, w \rangle \rangle && (2.3)\\
&= \langle x_{1,1}, y_{1,1}, x_{2,1}, y_{2,1}, x_{1,2}, y_{1,2}, x_{2,2}, y_{2,2}, \ldots, \\
&\quad x_{1,l}, y_{1,l}, x_{2,l}, y_{2,l}, w \rangle. && (2.4)
\end{aligned}$$

2.2 Templates and Affine Transformations

2.2.1 Character templates

Given structurally labeled training examples of the two classes of characters that we want to classify, we make a template character based on the stroke parameters. For strokes common to both classes of characters, we take the mean values of the stroke parameters from both classes. For strokes specific to each class, we assume a multivariate normal distribution among the stroke parameters, condition on the mean common stroke parameters, and take the conditional mean of the stroke parameters specific to the class. The strokes specific to each class are rendered with half intensity. Figure 2.5 shows the character templates for characters classes in Figure 2.4. These templates are used to find structures in unknown test examples.

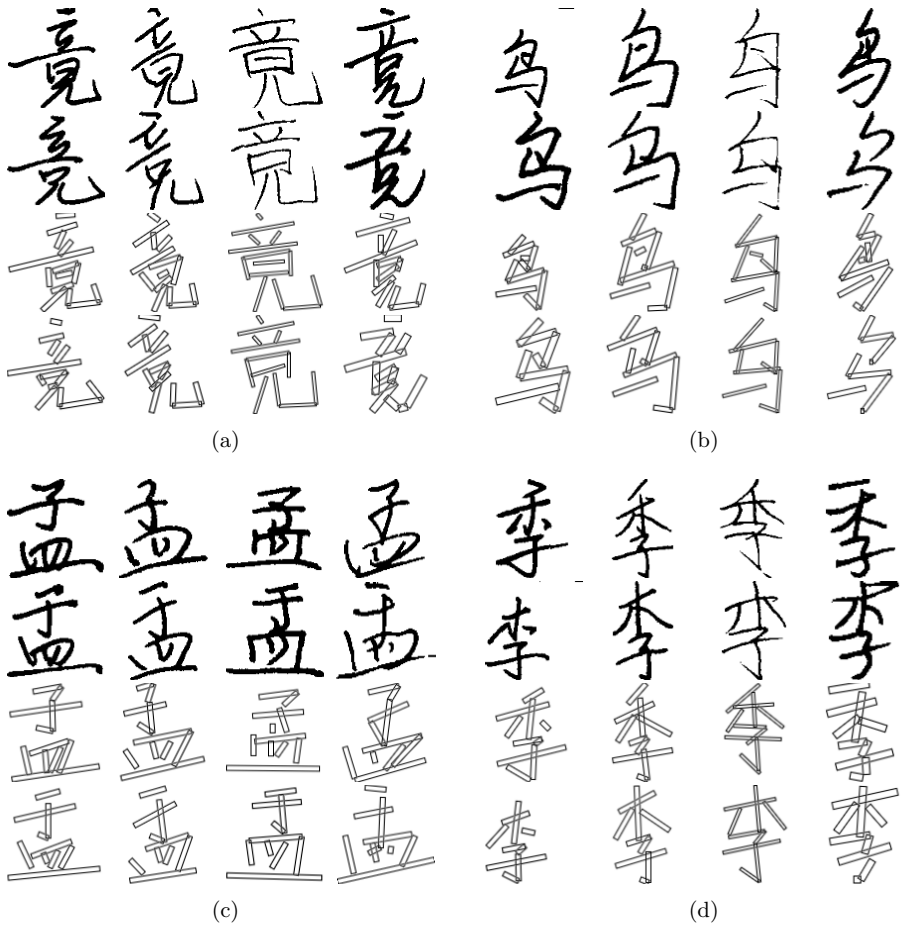


Figure 2.4: Examples of characters with their strokes modeled using long and thin rectangles.

(a) (b) (c) (d)

Figure 2.5: Character templates generated using stucturally labeled training examples. Strokes specific to each character class are rendered with half intensity.

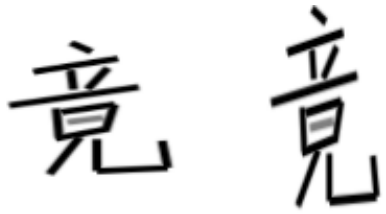
2.2.2 Affine transformations on the templates

We find structures in character images by affine transforming characters templates such as those in Figure 2.5 and matching them to unknown test character images. The family of transformations we consider include translation, scaling, rotation, shearing, as well as any composition of them. An affine transformation is represented as 6 parameters, with 4 parameters specifying the 2×2 transformation matrix and 2 parameters for the translation in the two directions:

$$A = \langle M, \langle t_x, t_y \rangle \rangle \tag{2.5}$$

$$= \left\langle \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \langle t_x, t_y \rangle \right\rangle . \tag{2.6}$$

Figure 2.6 shows some examples of transformed templates and their transformation matrices.



$$M = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}$$

(a) Scaling

$$M = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

(b) Rotation

$$M = \begin{pmatrix} 1 & k_1 \\ k_2 & 1 \end{pmatrix}$$

(c) Shearing

Figure 2.6: Affine transformed templates and their transformation matrices.

2.2.3 Origin used in affine transformations

The choice of the origin used in the affine transformation is important in our application. The origin of the transformation is typically fixed at the upper-left corner of the image. This makes the gradient of some components of the transformation less apparent when we use gradient-based optimizations to find the transformation that best matches the template to an image (Figure 2.7a). Instead, we place the origin of the transformation at the center of mass of the template (Figure 2.7b) so that the different components of the transformation are more linearized.

2.3 Gaussian-smoothed Matching of Character Images

The affine transformed templates are used to find structures in the character images. This is achieved by Gaussian-smoothing the template and the character image, and doing a gradient-based optimization to find the affine transformation on the Gaussian-smoothed template that best matches the

(a) Origin fixed at the upper-left corner

(b) Origin placed at the center of mass

Figure 2.7: Placement of the origin in affine transformations and its effect. In Figure 2.7a, when the origin is fixed at the upper-left corner, it requires a scaling and a translation to shrink a template in place. When the origin is placed at the center of mass as in Figure 2.7b, shrinking the template in place only requires a scaling operation.

smoothed character image as follows:

$$
\begin{aligned}
A^* & \\
&= \arg\max_A P(A|X) && (2.7)\\
&= \arg\max_A \frac{P(X|A)P(A)}{P(X)} && (2.8)\\
&= \arg\max_A P(X|A)P(A) && (2.9)\\
&= \arg\max_A P(X|A) && (2.10)\\
&= \arg\max_A P(X|\text{Transform}(T,A)) && (2.11)\\
&= \arg\min_A -\log P(X|\text{Transform}(T,A)) && (2.12)\\
&= \arg\min_A \left\|\text{Smooth}(\text{Transform}(T,A),\sigma) - \text{Smooth}(X,\sigma)\right\|^2, && (2.13)
\end{aligned}
$$

where X is the character image, T is the template image for the whole character, $\text{Transform}(T, A)$ transforms T using affine parameters A, and $\text{Smooth}(X, \sigma)$ applies a Gaussian filter with standard deviation σ to image X. Equation (2.10) assumes that all transformations are equally likely. $P(X|\text{Transform}(T, A))$ in Equation (2.11) is the conditional probability of seeing the character image X assuming that the stroke pixels in X are generated by the underlying stroke configuration specified by the affine transformed template $\text{Transform}(T, A)$ with a Gaussian noise model, and the objective function in Equation (2.13) is proportional to the negative log-likelihood. We use an implementation of limited memory bound-constrained BFGS algorithm [1, 33] for our gradient-based optimization.

After the optimization finishes, we can easily extract the approximate

configuration of each stroke in the transformed template:

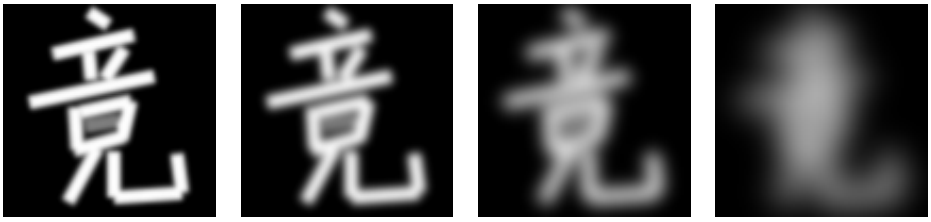
$$\Gamma^* = \text{ExtractStrokes}(\text{Transform}(T, A^*)). \tag{2.14}$$

The Gaussian-smoothing serves as the error model for the appearance matching. It assumes a spherical normal distribution for each stroke pixel in the template image and the character image. That is, for each stroke pixel appearing at (x_0, y_0) in the image, the actual location that generates that stroke pixel is distributed according to the probability density function:

$$f(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}\left((x-x_0)^2+(y-y_0)^2\right)}. \tag{2.15}$$

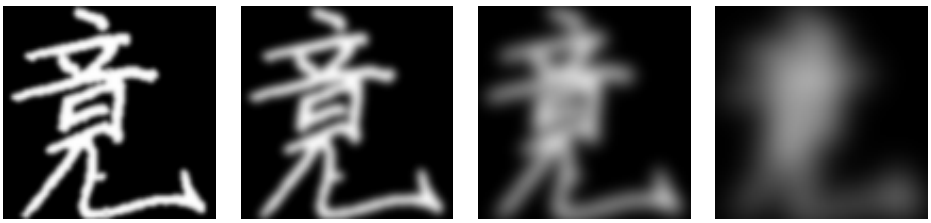
Figure 2.8 shows a character template smoothed using Gaussian filters of different standard deviations, and Figure 2.9 shows an actual character image smoothed using Gaussian filters of different standard deviations. The best match between the affine transformed template and the character image is shown in Figure 2.10. In general, the less heavily the images are smoothed, the more accurately the global minimum of Equation (2.13) will match each individual stroke.

Although the global minimum of Equation (2.13) corresponds to an affine transformation such that the transformed template more accurately matches the stroke pixels in the character image, unfortunately Equation (2.13) is not convex. A gradient-based optimization may not necessarily converge to the global optimum. Furthermore, the optimization is more likely trapped in an undesirable local optimum when insufficient amount of smoothing is



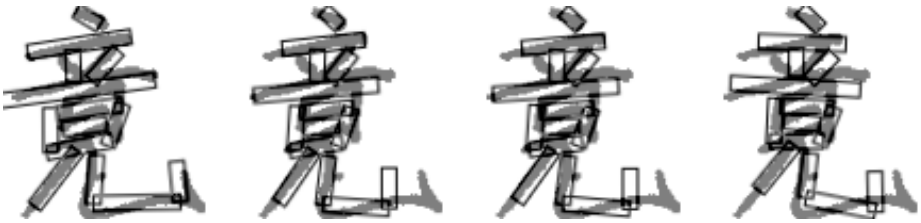
(a) $\sigma = 1$ (b) $\sigma = 2$ (c) $\sigma = 4$ (d) $\sigma = 8$

Figure 2.8: A character template smoothed with Gaussian filters of different standard deviations.



(a) $\sigma = 1$ (b) $\sigma = 2$ (c) $\sigma = 4$ (d) $\sigma = 8$

Figure 2.9: An actual character image smoothed with Gaussian filters of different standard deviations.



(a) $\sigma = 1$ (b) $\sigma = 2$ (c) $\sigma = 4$ (d) $\sigma = 8$

Figure 2.10: The best affine transformed character template that matches the character image with different standard deviations for the Gaussian smoothing used in the optimization in Equation (2.13). The best affine transformed template matches the individual strokes more accurately with less smoothing.

used. Figure 2.11 shows how the optimization converges to different local optima when initialized at different places and how smoothing helps the optimization to converge to the global optimum.

The amount of smoothing, or the standard deviation for the Gaussian filtering σ, is chosen to balance the accuracy of the match and convergence to the global optimum. Section 2.4.3 describes learning σ, as well as other parameters used in matching templates, in detail.

Section 2.3.1 describes registration of the character image, which helps to initialize the optimization to find the global optimum. Section 2.5 describes initialization of the optimization in detail.

2.3.1 Registration of the character image

In order to place the initial template so that the optimization is more likely to converge to the global optimum, as well as coming up with a better statistical model for the stroke parameters used to predict the sub-structures as will be discussed in Section 2.4, each example of the character is re-registered in the image frame. This is done through a two-step template matching on the character image as described in Algorithm 1. First, given the character template T_0, copies of T_0 of various scales and aspect ratios are generated as in Figure 2.6a. Then, for each scale and aspect ratio of the template, T_i, the best global match T_i^* to the character image X and its matching score v_i are found efficiently using the sum of squared difference criterion and Fast Fourier Transform. Notice that this first stage matching only considers a discretized set of scaling and translation. The best match among all T_i^* is then used as the initial template to do a more flexible matching with

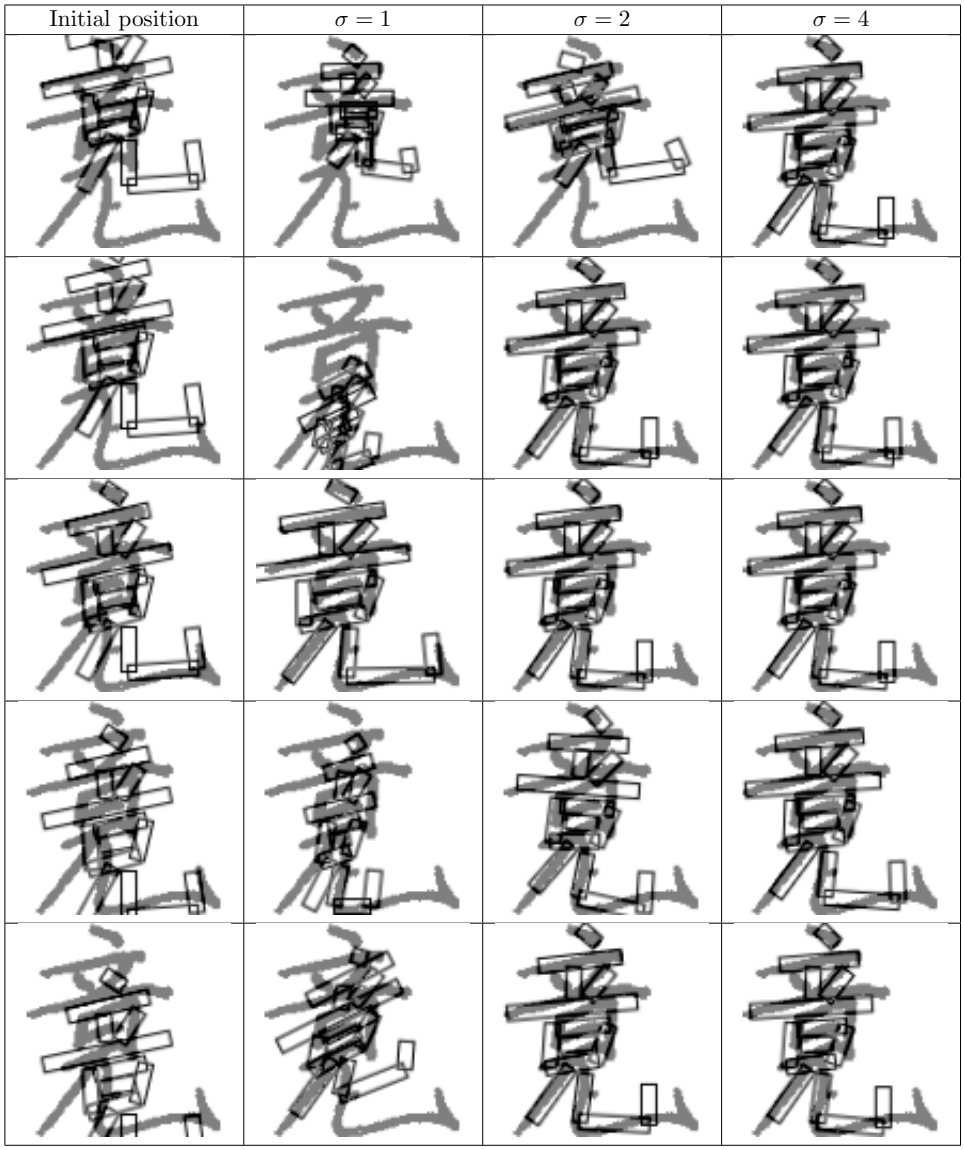


Figure 2.11: Where the gradient-based optimization converges with different amount of smoothing and different initial template position. With insufficient smoothing, the optimization has to be initialized very close to the desired optimum in order to converge to it. With more heavily smoothing, the optimization is more tolerant to the initial position, albeit less accurate match.

Algorithm 1 Re-registering a character image X given character template T_0.

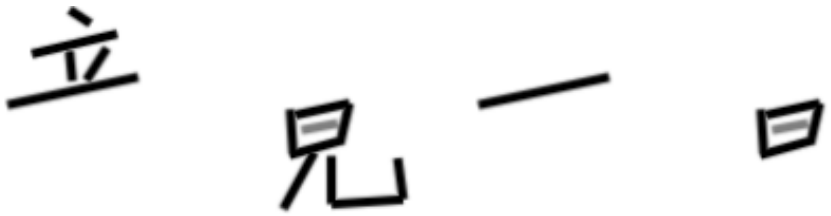
function REREGISTER(X, T_0)
  $S \leftarrow$ MAKESCALEDCOPIES(T_0)
  for all $T_i \in S$ **do**
    $T_i^*, v_i \leftarrow$ FFTSSDBESTMATCH(X, T_i)
  end for
  $i^* \leftarrow \arg\min_i v_i$
  $T^* \leftarrow$ AFFINEMATCH(X, T_i^*)
  $b \leftarrow$ BOUNDINGBOX(T^*)
  $X^* \leftarrow$ REGISTERTOBOUNDINGBOX(X, b)
  return X^*
end function

full affine transformations on the character image using the same gradient based algorithm as described in Section 2.3. Finally, the bounding box of the affine-transformed template T^* that best matches X is used to re-register the character image.

2.3.2 Estimating the stroke width

After the character image is properly registered, the width of the strokes is approximated by multiplying the stroke width of the character template with the ratio of the number of stroke pixels in the character image to the number of stroke pixels in the template:

$$width = width_{template} \times \frac{\#\text{stroke pixels in the character image}}{\#\text{stroke pixels in the template}}. \quad (2.16)$$



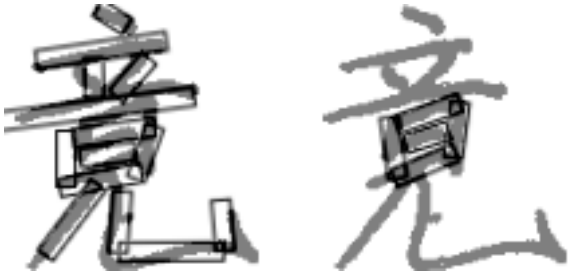
(a) (b) (c) (d)

Figure 2.12: Examples of templates of sub-structures for the character in Figure 2.5a.

2.4 Gaussian-smoothed Matching of Sub-structures

After a rough estimate of the location of each stroke in the character image is obtained using the whole character template matching in Section 2.3, we proceed to refine the match of its sub-structures such as radicals, sub-radicals, and other highly correlated sets of strokes, by generating templates of portions of the character and matching them using a similar gradient-based algorithm. Figure 2.12 shows a few examples of templates of sub-structures for the character in Figure 2.5a. Figure 2.13 shows how the strokes of the middle radical can be more accurately located over the locations found by doing the whole-character match. This sub-structure matching can be performed recursively to improve the accuracy of the match for each subpart of the character.

The optimization for matching the template of a sub-structure to the character image X given affine parameters found in previous matchings $A^*_{r-1}, A^*_{r-2}, \ldots, A^*_1$ is:



(a) Matching the whole character

(b) Matching the middle radical

Figure 2.13: Comparison of the match quality for the middle radical. Note how the match quality of each strokes in the middle radical in Figure 2.13b is much better than the match quality of each stroke in the middle radical in Figure 2.13a.

$$
\begin{aligned}
& A_r^* \\
&= \arg\max_{A_r} P(A_r|X, A_{r-1}^*, A_{r-2}^*, \ldots, A_1^*) && (2.17)\\
&= \arg\max_{A_r} \frac{P(X|A_r, A_{r-1}^*, \ldots, A_1^*)P(A_r|A_{r-1}^*, A_{r-2}^*, \ldots, A_1^*)}{P(X|A_{r-1}^*, A_{r-2}^*, \ldots, A_1^*)} && (2.18)\\
&= \arg\max_{A_r} P(X|A_r, A_{r-1}^*, \ldots, A_1^*)P(A_r|A_{r-1}^*, A_{r-2}^*, \ldots, A_1^*) && (2.19)\\
&= \arg\max_{A_r} P(X_r|A_r)P(A_r|A_{r-1}^*, A_{r-2}^*, \ldots, A_1^*) && (2.20)\\
&= \arg\max_{A_r} P(X_r|\text{Transform}(T_r, A_r))P(A_r|A_{r-1}^*, A_{r-2}^*, \ldots, A_1^*) && (2.21)\\
&= \arg\min_{A_r} -\log(P(X_r|\text{Transform}(T_r, A_r)) \\
&\quad \times P(A_r|A_{r-1}^*, A_{r-2}^*, \ldots, A_1^*)) && (2.22)\\
&= \arg\min_{A_r} \|\text{Smooth}(\text{Transform}(T_r, A_r), \sigma_{f,r}) - \text{Smooth}(X_r, \sigma_{f,r})\|^2 \\
&\quad - \hat{\rho}_r \log P(A_r|A_{r-1}^*, A_{r-2}^*, \ldots, A_1^*), && (2.23)
\end{aligned}
$$

where X is the character image, and X_r is the character image with pixels in irrelevant parts of the character discounted. T_r is the template image for sub-structure r, Transform(T_r, A_r) transforms T_r using affine parameters A_r, and Smooth$(X_r, \sigma_{f,r})$ applies a Gaussian filter with standard deviation $\sigma_{f,r}$ to image X_r. $A^*_{r-1}, A^*_{r-2}, \ldots, A^*_1$ are affine parameters for sub-structures $r-1, r-2, \ldots, 1$ found in previous matchings, among which A^*_1 is the set of best affine parameters for matching the whole character.

In Equation (2.23), compared to the objective function for the whole character match in Equation (2.13), there are two differences. First, instead of the whole character image X, we discount pixels in irrelevant parts of the character and use the discounted character image X_r specific to the sub-structure that we want to match. This will be detailed in Section 2.4.1. The other difference is the conditional prior $P(A_r|A^*_{r-1}, A^*_{r-2}, \ldots, A^*_1)$, which will be discussed in Section 2.4.2. Note that the objective function for the whole character match in Equation (2.13) is a special case of Equation (2.23).

2.4.1 Discounting distracting stroke pixels

Consider the alternative optimization for matching a sub-structure, that better resembles the optimization used for matching the whole character in Equation (2.13):

$$A^*_r = \arg\min_{A_r} \|\text{Smooth}(\text{Transform}(T_r, A_r), \sigma_{f,r}) - \text{Smooth}(X, \sigma_{f,r})\|^2 . \tag{2.24}$$

Equation (2.24) matches the transformed template Transform(T_r, A_r) to the whole character image X instead of the discounted character image X_r, and does not have the conditional prior $P(A_r|A^*_{r-1}, A^*_{r-2}, \ldots, A^*_1)$.

As is the same for the whole character matching, matching the template of a sub-structure to the character image using Equation (2.24) is not a convex optimization problem. In fact, the match that we want may not even be the global optimum based on the appearance measure, and the optimization is even more likely trapped in an undesirable local optimum. Figure 2.14 shows some examples of local optima with different amount of smoothing used in the matching.

Unfortunately, unlike the whole character matching, re-registering the character image does not help much in initializing the optimization to the right place. More heavily smoothing the image does not by itself help much, either. The problem is that, in addition to the sub-structure that we are trying to find, there are also many distracting stroke pixels that we do not intend to match to the template in the character image. Not matching these distracting stroke pixels that we do not intend to match should not result in a penalty in the objective function.

We find the structures in a character image by matching templates to the character image in a hierarchical way. That is, we match the whole character first, then we recursively match its sub-structures. When we match templates of a portion of the character to the character image, we can discount pixels in the character image that we do not intend to match using the rough stroke locations that we found in previous template matchings.

X_r, the character image with irrelevant parts discounted, is computed

Figure 2.14: Where the gradient-based optimization converges with different amount of smoothing and different initial template position. More heavily smoothing by itself does not help much in finding the desired optimum.

as follows:

$$\begin{aligned} X_r &= X \circ (\mathbf{1} - \text{Smooth}(\text{Render}(\hat{\Gamma}^*_{r-1}), \sigma_{b,r})) \\ &\quad \circ (\mathbf{1} - \text{Smooth}(\text{Render}(\hat{\Gamma}^*_{r-2}), \sigma_{b,r})) \circ \dots \\ &\quad \circ (\mathbf{1} - \text{Smooth}(\text{Render}(\hat{\Gamma}^*_{1}), \sigma_{b,r})), \end{aligned} \tag{2.25}$$

where $\circ$ is the Hadamard product defined as $(A \circ B)_{i,j} = (A)_{i,j}(B)_{i,j}$. $\mathbf{1}$ is the matrix of 1s, $\text{Smooth}(T, \sigma_{b,r})$ applies a Gaussian filter of standard deviation $\sigma_{b,r}$ to image T, and $\text{Render}(\Gamma)$ renders stroke parameters Γ into an image of strokes. $\Gamma^*_i = \text{ExtractStrokes}(\text{Transform}(T_i, A^*_i))$ is the stroke parameters extracted from the transformed template $\text{Transform}(T_i, A^*_i)$, and $\hat{\Gamma}^*_i$ is the largest subset of Γ^*_i that does not include any stroke refined by later matchings $\Gamma^*_{i+1}, \Gamma^*_{i+2}, \dots, \Gamma^*_{r-1}$ or strokes in the current sub-structure to be matched Γ_r.

The Gaussian smoothing in Equation (2.25) again serves as the error model. It assumes that for each stroke pixel located at (x_0, y_0) in the transformed template, the location of the actual stroke pixel in the character image corresponding to it is distributed according to the spherical normal distribution

$$f(x, y) = \frac{1}{2\pi\sigma^2_{b,r}} e^{-\frac{1}{2\sigma^2_{b,r}}\left((x-x_0)^2+(y-y_0)^2\right)}. \tag{2.26}$$

The value for the standard deviation of the Gaussian smoothing $\sigma_{b,r}$ will be optimized for the matching, and will be described in Section 2.4.3.

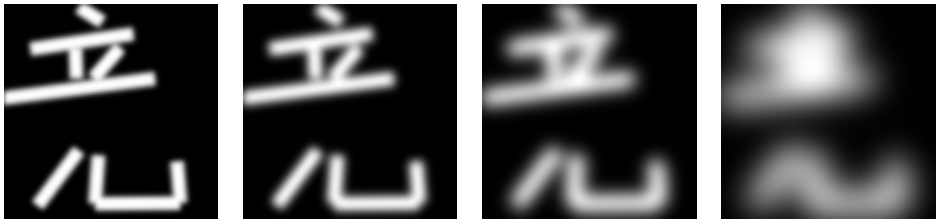
Suppose we want to match the middle radical after the whole character is matched as in Figure 2.13, Figure 2.15 shows the matching results to the

Figure 2.15: Previous matching results to the strokes that we do not intend to match using the current template.

strokes other than those in the middle radical. These are the rough locations of strokes that we do not intend to match using the current template of the radical, and the stroke pixels at these locations in the character image should be discounted.

To discount the irrelevant pixels in the character image, we first apply Gaussian smoothing to the strokes in the transformed templates matched to the irrelevant strokes to compute the probability of each pixel explained by an irrelevant stroke according to the error model in Equation (2.26), then the pixels in the character image is discounted according to Equation (2.25). Figure 2.16 shows the matched irrelevant strokes found in previous template matchings smoothed with Gaussian filters of various standard deviations, and Figure 2.17 shows the corresponding discounted character image X_r.

Figure 2.18 shows the results of matching the middle radical to the discounted image X_r with the irrelevant strokes found in previous template matchings smoothed using Gaussian filters of various standard deviations and discounted. Notice how the matchings are greatly improved over those in Figure 2.14.

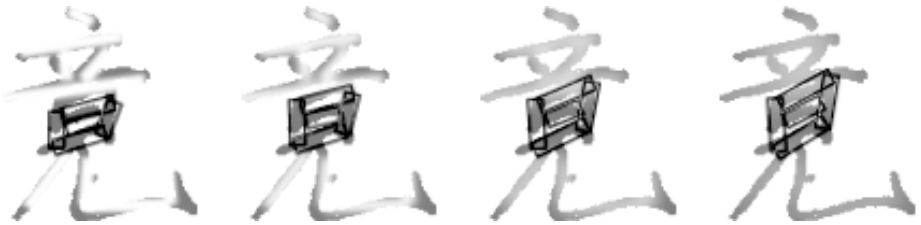


(a) $\sigma_{b,r} = 1$ (b) $\sigma_{b,r} = 2$ (c) $\sigma_{b,r} = 4$ (d) $\sigma_{b,r} = 8$

Figure 2.16: Irrelevant strokes in the matched template smoothed using Gaussian filters with various standard deviations.

(a) $\sigma_{b,r} = 1$ (b) $\sigma_{b,r} = 2$ (c) $\sigma_{b,r} = 4$ (d) $\sigma_{b,r} = 8$

Figure 2.17: Discounted character image X_r with irrelevant strokes smoothed using Gaussian filters of various standard deviations and discounted.



(a) $\sigma_{b,r} = 1$ (b) $\sigma_{b,r} = 2$ (c) $\sigma_{b,r} = 4$ (d) $\sigma_{b,r} = 8$

Figure 2.18: Results of matching the middle radical to the discounted image X_r with irrelevant strokes found in previous template matchings smoothed using Gaussian filters of various standard deviations and discounted. $\sigma_{f,r}$ is set to 1.

2.4.2 Conditional prior prediction for sub-structures

The optimization in Equation (2.23) includes a conditional prior $P(A_r|A^*_{r-1}, A^*_{r-2}, \ldots, A^*_1)$, or the conditional distribution for the affine parameters A_r given best affine parameters found in previous template matchings $A^*_{r-1}, A^*_{r-2}, \ldots, A^*_1$. This distribution is defined in terms of stroke parameters.

The affine parameters A_r along with its associated template T_r define a set of parameters for strokes included in T_r:

$$\begin{aligned} \Gamma_r &= \text{ExtractStrokes}(\text{Transform}(T_r, A_r)) && (2.27) \\ &= \langle x_{1,1,r}, y_{1,1,r}, x_{2,1,r}, y_{2,1,r}, x_{1,2,r}, y_{1,2,r}, x_{2,2,r}, y_{2,2,r}, \ldots, \\ &\quad x_{1,l_r,r}, y_{1,l_r,r}, x_{2,l_r,r}, y_{2,l_r,r}, w\rangle. && (2.28) \end{aligned}$$

Note that the stroke width w is estimated separately. We define Γ'_r to be the stroke parameters without the stroke width w:

$$\begin{aligned} \Gamma'_r &= \langle x_{1,1,r}, y_{1,1,r}, x_{2,1,r}, y_{2,1,r} x_{1,2,r}, y_{1,2,r}, x_{2,2,r}, y_{2,2,r}, \ldots, \\ &\quad x_{1,l_r,r}, y_{1,l_r,r}, x_{2,l_r,r}, y_{2,l_r,r}\rangle. && (2.29) \end{aligned}$$

Similarly, the affine parameters $A^*_{r-1}, A^*_{r-2}, \ldots, A^*_1$ along with their associated templates $T_{r-1}, T_{r-2}, \ldots, T_1$ also define sets of stroke parameters $\Gamma^*_{r-1}, \Gamma^*_{r-2}, \ldots, \Gamma^*_1$. We define $\bar{\Gamma}^*_i$ to be the largest subset of Γ^*_i that does not include any stroke refined by later matchings $\Gamma^*_{i+1}, \Gamma^*_{i+2}, \ldots, \Gamma^*_{r-1}$. The

union

$$\bar{\Gamma}^*_{\bigcup_{r-1}} = \bigcup_{i=1}^{r-1} \bar{\Gamma}^*_i \tag{2.30}$$

is the set of parameters of the most refined strokes up to Γ^*_{r-1}. Note that $\bar{\Gamma}^*_{\bigcup_{r-1}}$ includes every stroke in the character, because Γ^*_1 is the strokes obtained from the template match to the whole character and includes every stroke. Again we define $\bar{\Gamma}^{*\prime}_{\bigcup_{r-1}}$ to be $\bar{\Gamma}^*_{\bigcup_{r-1}}$ without the stroke width w:

$$\bar{\Gamma}^{*\prime}_{\bigcup_{r-1}} = \langle x^*_{1,1}, y^*_{1,1}, x^*_{2,1}, y^*_{2,1}, x^*_{1,2}, y^*_{1,2}, x^*_{2,2}, y^*_{2,2}, \ldots, x^*_{1,l}, y^*_{1,l}, x^*_{2,l}, y^*_{2,l} \rangle. \tag{2.31}$$

Given structurally labeled training examples, we define a joint multivariate normal distribution

$$\begin{aligned} P(\Gamma'_r, \bar{\Gamma}^{*\prime}_{\bigcup_{r-1}}) &= N(\mu_r, \Sigma_r + \frac{\lambda_1}{100}\mathbf{I}) && (2.32) \\ &= N\left(\begin{pmatrix} \mu_{r1} \\ \mu_{r2} \end{pmatrix}, \begin{pmatrix} \Sigma_{r11} & \Sigma_{r12} \\ \Sigma_{r21} & \Sigma_{r22} \end{pmatrix} \right), && (2.33) \end{aligned}$$

where $N(\mu, \Sigma)$ denotes a multivariate normal distribution with mean μ and covariance matrix Σ. In Equation (2.33), μ_{r1} is the mean corresponding to Γ'_r, and μ_{r2} is the mean corresponding to $\bar{\Gamma}^{*\prime}_{\bigcup_{r-1}}$. The covariance matrix is split in the same way, and smoothed by adding $\frac{\lambda_1}{100}\mathbf{I}$ to it, where $\mathbf{I}$ is the identity matrix and λ_1 is the largest eigenvalue of Σ_r.

The conditional distribution $P(\Gamma'_r | \bar{\Gamma}^{*\prime}_{\bigcup_{r-1}})$ is also a multivariate normal

distribution:

$$P(\Gamma'_r|\bar{\Gamma}^{*\prime}_{\bigcup_{r-1}}) = N\left(\mu_{r1} + \Sigma_{r12}\Sigma_{r22}^{-1}(\bar{\Gamma}^{*\prime}_{\bigcup_{r-1}} - \mu_{r2}), \Sigma_{r11} - \Sigma_{r12}\Sigma_{r22}^{-1}\Sigma_{r21}\right). \tag{2.34}$$

The most likely Γ'_r given $\bar{\Gamma}^{*\prime}_{\bigcup_{r-1}}$ is the conditional mean:

$$\begin{aligned} \Gamma'_{r,ML} &= \arg\max_{\Gamma'_r} P(\Gamma'_r|\bar{\Gamma}^{*\prime}_{\bigcup_{r-1}}) && (2.35)\\ &= \mu_{r1} + \Sigma_{r12}\Sigma_{r22}^{-1}(\bar{\Gamma}^{*\prime}_{\bigcup_{r-1}} - \mu_{r2}). && (2.36) \end{aligned}$$

The conditional prior for A_r is thus

$$P(A_r|A^*_{r-1}, A^*_{r-2}, \ldots, A^*_1) \propto P(\Gamma'_r|\bar{\Gamma}^{*\prime}_{\bigcup_{r-1}}). \tag{2.37}$$

Therefore, the second half of Equation (2.23), the negative logarithm of the conditional prior, is of the form

$$\begin{aligned} &-\hat{\rho}_r \log P(A_r|A^*_{r-1}, A^*_{r-2}, \ldots, A^*_1) \\ &= \rho_r(\Gamma'_r - \Gamma'_{r,ML})^{\mathsf{T}}(\Sigma_{r11} - \Sigma_{r12}\Sigma_{r22}^{-1}\Sigma_{r21})^{-1}(\Gamma'_r - \Gamma'_{r,ML}). \quad (2.38) \end{aligned}$$

Note that not all Γ'_r corresponds to a valid A_r. To find the most likely value A_r given $A^*_{r-1}, A^*_{r-2}, \ldots, A^*_1$, we first find the most likely value for Γ'_r using Equation (2.36), then use the affine parameter A^*_r such that the stroke parameters $\Gamma^{*\prime}_r$ corresponding to A^*_r is closest to $\Gamma'_{r,ML}$.

This conditional prior is used as a regularization for the template matching. It also helps to initialize the optimization to the right place.

Learning the ground truth transformations

In Section 2.4.2, we need the ground truth transformations to learn the mean μ_r and covariance Σ_r in the multivariate normal distribution in Equation (2.32). Specifically, we need the ground truth $A_{i,GT}$ for each template T_i to compute the ground truth stroke parameters

$$\Gamma_{i,GT} = \text{ExtractStrokes}(T_i, A_{i,GT}). \tag{2.39}$$

With the structurally labeled training examples, we first segment the image pixels according to the labeled stroke that the pixel is closest to. We then mask out pixels closest to strokes not used in T_i, leaving only pixels closest to a stroke used in T_i. A template matching using T_i is then performed on this masked image, and the resulting affine transformation parameter $A_{i,GT}$ that best matches the masked image is used as the ground truth.

2.4.3 Learning the parameters for template matchings

In the optimization in Equation (2.23), for each template T_r, there are 3 parameters that need to be learned: $\sigma_{f,r}$, the standard deviation for the Gaussian filter that smooths T_r and X_r, $\sigma_{b,r}$ in Equation (2.25), the standard deviation for the Gaussian filter that smooths the previously matched strokes not used in T_r, and ρ_r in Equation (2.38), the strength of the conditional prior. We denote $\Phi_{shape,r}$ to be the set of the three parameters above:

$$\Phi_{shape,r} = \langle \sigma_{f,r}, \sigma_{b,r}, \rho_r \rangle, \tag{2.40}$$

as $\Phi_{shape,r}$ decides the shape of the objective function in Equation (2.23). Additionally, the initial point for the gradient-based optimization in Equation (2.23) also needs to be learned. We use $\Phi_{init,r}$ to represent the parameters that decide where to place the initial A_r for the gradient-based optimization. $\Phi_{init,r}$ does not affect the shape of the objective function. We cover $\Phi_{init,r}$ in detail in Section 2.5.

Quality of the match

To optimize $\Phi_{shape,r}$ and $\Phi_{init,r}$, we first need to define the quality of the match. Given a structurally labeled training character image X^i for example i, we segment the stroke pixels in the image according to the labeled stroke that the pixel is closest to, obtaining a stroke image X^i_j for each stroke j. Given the set of affine parameters A^{i*}_r obtained by matching its associated template T_r to X^i_r using the gradient-based optimization in Equation (2.23) with $\Phi_{shape,r}$ and $\Phi_{init,r}$ as its parameters, we extract the stroke parameters for each stroke in the transformed template

$$\Gamma^{i*}_r = \text{ExtractStrokes}(\text{Transform}(T_r, A^{i*}_r)) \tag{2.41}$$

$$= \langle \Psi^{i*}_{s_{r,1}}, \Psi^{i*}_{s_{r,2}}, \ldots, \Psi^{i*}_{s_{r,l_r}} \rangle, \tag{2.42}$$

where $\Psi^{i*}_{s_{r,j}}$ is the stroke parameters corresponding to stroke $s_{r,j}$ in the character, and there are l_r strokes in T_r. Note that X^i_r is properly discounted using $A^{i*}_{r-1}, A^{i*}_{r-2}, \ldots, A^{i*}_1$ as described in 2.4.1.

For each stroke $s_{r,j}$ in Transform(T_r, A_r^{i*}), we define a per-stroke loss as

$$loss^i_{stroke,s_{r,j}} = \left\| \text{Smooth}(\text{Render}(\Psi^{i*}_{s_{r,j}}), \sigma_l) - \text{Smooth}(X^i_{s_{r,j}}, \sigma_l) \right\|^2. \quad (2.43)$$

σ_l is chosen to define a Gaussian loss model suitable for the domain.

The loss for matching Transform(T_r, A_r^{i*}) to X_r^i is defined as

$$loss^i_{radical,r} = \sum_{i=1}^{l_r} loss^i_{stroke,s_{r,j}}. \quad (2.44)$$

The loss among all n structurally labeled training examples for radical r using $\Phi_{shape,r}$ and $\Phi_{init,r}$ is

$$loss_{radical,r} = \frac{1}{n} \sum_{i=1}^{n} loss^i_{radical,r}. \quad (2.45)$$

Algorithm 2 outlines the algorithm that learns $\Phi_{shape,r}$ and $\Phi_{init,r}$ for each match operation r in a sequence. For each match operation in the

Algorithm 2 Learning parameters for match operations.

procedure OPTIMIZEPARAMETERS
  for $r \leftarrow 1 \ldots m$ **do**
    $\Phi_{init,r} \leftarrow 0$ ▷ Initialize the optimization to the most likely location according to the conditional prior.
    $\Phi_{shape,r} \leftarrow$ Use Pattern Search to optimize $\Phi_{shape,r}$ to minimize $loss_{radical,r}$.
    $\Phi_{init,r} \leftarrow$ Optimize $\Phi_{init,r}$ according to Section 2.5.
  end for
end procedure

match sequence, we first optimize $\Phi_{shape,r}$ to minimize $loss_{radical,r}$, using the most likely A_r as the initial point for for the template match. Then $\Phi_{init,r}$

(a) Initial location

(b) Matched stroke

Figure 2.19: Initial location of a single-stroke template and its matching result on a character image with irrelevant strokes properly discounted.

is optimized to maximize the probability of success of the match operation, as described in 2.5.

2.5 Initial Point for the Optimization

As is made evident in Section 2.3 and Section 2.4, the objective function for matching templates to character images is not convex. The success of using gradient-based optimization to match the template to the correct structure is sensitive to where we initialize the optimization. Figure 2.19 shows that the initial point can greatly affect the result of the match even on character images with distracting strokes properly discounted. If the initial point is placed within the desired locally convex basin of the objective function, the optimization can reach the desired local optimum by following the local gradient. In the following, we look at the shape of the objective function, the initial point, and probability of success of the optimization in detail.

2.5.1 Shape of the objective function

The objective function in Equation (2.23) of each of our template-matching operation is of the form

$$\begin{aligned} & f_{obj}(A_r) \\ &= \|\text{Smooth}(\text{Transform}(T_r, A_r), \sigma) - \text{Smooth}(X_r, \sigma)\|^2 \\ &\quad + \rho_r(\Gamma_r(A_r) - \Gamma_{r,ML})^{\mathsf{T}}\Sigma^{-1}(\Gamma_r(A_r) - \Gamma_{r,ML}). \end{aligned} \tag{2.46}$$

In Equation (2.46), $\Gamma_r(A_r)$ represents the stroke parameters extracted from the transformed template $\text{Transform}(T_r, A_r)$, and $\Gamma_{r,ML}$ is the most likely stroke configurations predicted by the conditional prior.

Equation (2.46) is a sum of squared differences. It is smooth and differentiable everywhere because all pixel values in the smoothed and transformed template $\text{Smooth}(\text{Transform}(T_r, A_r), \sigma)$ and all stroke parameter values in $\Gamma_r(A_r)$ are smooth and differentiable in A_r. The second half of Equation (2.46), $(\Gamma_r(A_r) - \Gamma_{r,ML})^{\mathsf{T}}\Sigma^{-1}(\Gamma_r(A_r) - \Gamma_{r,ML})$, which is a squared Mahalanobis distance, is always convex in $\Gamma_r(A_r)$. However, the first half, the sum of squared Gaussian-smoothed differences of two images consisting of strokes, is not convex.

In general, there may be a local optimum in a locally convex region in the objective function whenever a sub-structure in the template gets close to matching a sub-structure in the image. These locally convex regions are a smooth function of the spatial configuration of the structures of the character image because of the Gaussian smoothing; when stroke pixels

move in the character image, the shape of the objective function changes smoothly. The sizes of the locally convex regions depend on the sizes of the sub-structures. With extensive Gaussian smoothing to the image that filters out high-frequency finer structures, the locally convex regions can be widened, at the expense of losing the ability of accurately matching the finer structures, as is shown in Figure 2.10 and Figure 2.11.

2.5.2 Success of the gradient-based optimizations

An idealized gradient-based optimization algorithm that follows local gradients always converges to the local optimum that lies in the same locally convex region that the optimization is initialized in. The locally convex regions work as *basins of attraction* of the optimization. We define the success of the optimization as finding the "desired" local optimum, which is not necessarily the global optimum. Therefore, for an idealized gradient-based optimization algorithm as defined above, the optimization succeeds if and only if it is initialized in the desired locally convex region that contains the desired optimum. For our purpose of finding structures in character images by matching affine-transformed templates to the image, the desired locally convex region is the one that contains the ground truth affine transformation for the image.

One intuitive choice of the initialization would be the affine transformation corresponding to the most likely stroke parameters conditioned on the structures found by previous matching operations in the sequence of optimizations according to the conditional prior model in Equation (2.38). While this may be a good choice in predicting the location of the structure

(a) Ground truth obtained using the method in Section 2.4.2

(b) The result that the optimization converges to when initialized to Figure 2.20a

Figure 2.20: Determining the basin of attraction. Figure 2.20a is the ground truth determined using the method in Section 2.4.2. We initialized the optimization to Figure 2.20a, and Figure 2.20b is where the optimization converges to. The basin of attraction is defined as the set of initial points such that the optimization converges to a point close to Figure 2.20b.

that the current matching operation aims to find, it may not be the best initialization for the optimization. The most likely prediction could be close to the boundary of the desired locally convex region. It could even be outside the desired region due to the model error in the conditional prior.

Furthermore, the actual implementation of the optimization algorithm does not necessarily always stays within the same locally convex region that it is initialized in. Sometimes the line search in the algorithm can search past the ridges in the objective function and end up in a different local optimum. This happens more easily when the optimization is initialized near the boundary of the locally convex region. Therefore, we prefer that the optimizations be initialized in the desired locally convex region and as far away from the boundary of the locally convex region as possible. This increases the chance that the optimizer stays in the desired locally convex region and finds the desired local optimum.

Figure 2.20 shows how the *basin of attraction* is determined using the manually labeled ground truth for a stroke. We first initialize the opti-

Figure 2.21: An example of the basins of attraction for the two end points of a single-stroke template. The dotted areas are initial points for the end point that the optimization converges close to the desired local optimum. The plus signs denote the predicted most likely end points.

mization to the ground truth transformation determined using the method described in Section 2.4.2. When the optimization converges, the result of the optimization is the local optimum that lies in the same locally convex region as the ground truth. The *basin of attraction* is defined as the set of initial points such that the optimization converges to a point close to the point that it would converge to when initialized to the ground truth. Figure 2.21 shows an example of the basins of attraction for the two end points of a stroke.

We propose to move the initialization from the predicted most likely location, so that it lies closer to the center of desired the basin of attraction, and that the optimization is more likely to succeed. $\Phi_{init,r}$, the parameter for initializing the optimization for matching T_r is an *offset* to the predicted most likely location of the template. For a template consisting of multiple

strokes, it is the offset in the two coordinate locations to place the template:

$$\Phi_{init,r} = \langle d_x, d_y \rangle. \tag{2.47}$$

For a template containing a single stroke, we allow both end points to be moved separately:

$$\Phi_{init,r} = \langle d_{x1}, d_{y1}, d_{x2}, d_{y2} \rangle. \tag{2.48}$$

2.5.3 Modeling the probability of success and moving the initial point

Let B_r^j be the correct basin of attraction for matching T_r to the jth character example X_r^j, given the predicted most likely affine parameters A_r^* we want to choose an offset $\Phi_{init,r}$ such that the probability $P(\text{Move}(A_r^*, \Phi_{init,r}) \in B_r^j)$ is maximized across all j.

For a given $\Phi_{init,r}$, the success of converging to the desired optimum of each example j is a Bernoulli trial with a probability of success q_{b_r}.

Since the correct basin of attraction B_r^j is a smooth function of the spatial configuration of the structures of the character image, q_{b_r}, the probability that the moved initialization lies in B_r^j, should also be smooth with regard to $\Phi_{init,r}$, the spatial offset from the location of the predicted structure.

Assuming that the relative spatial locations between B_r^j and the predicted most likely location of the template A_r^* are Gaussian distributed, and that B_r^j forms similar spatial shapes for each j, the distribution of q_{b_r} will resemble sum of shifted Gaussians. Further assuming that B_r^j form chunks of continuous regions, the distribution of q_{b_r} will take the form of Gaussian

smoothed chunks of continuous regions. For q_{b_r} distributed in such a way, everything else being equal, picking a $\Phi_{init,r}$ in the center of a large chunk of B_r^j results in a higher probability of success q_{b_r}.

Because that the actual desired local optimum should always lie within B_r^j, and that the predicted configuration A_r^* should not be far from the actual optimum, we assume that there is always a large chunk of success region B_r^j reasonably close to the predicted configuration A_r^*. We aim to find and model such success regions for the structurally labeled training examples, and choose the $\Phi_{init,r}$ that maximizes the estimated q_{b_r} based on the model. Algorithm 3 outlines learning $\Phi_{init,r}$. We first probe a few locations around the predicted most likely location A_r^* and make a very crude estimate of the correct basin around A_r^* for the structurally labeled training examples, then compute for each offset $\Phi_{init,r}$ the probability of success. The probability of success is then smoothed with a Gaussian filter. The best $\Phi_{init,r}$ is chosen to maximize the smoothed probability.

Algorithm 3 Learning $\Phi_{init,r}$

function LEARNINITIALOFFSET($\mathbf{X}_r, \mathbf{A}^*$)
  for $j \leftarrow 1 \ldots n$ **do**
    $A_r^{j*} \leftarrow \arg\max_{A_r^j} P(A_r^j | A_{r-1}^{j*}, A_{r-2}^{j*}, \ldots, A_1^{j*})$
    $B_r^j \leftarrow$ ESTIMATEBASIN(A_r^{j*}, X_r^j)
  end for
  $Q_r \leftarrow$ SMOOTH($\frac{1}{n}\sum_{j=1}^{n} B_r^j, \sigma$)
  $\Phi_{init,r} \leftarrow \arg\max Q_r$
  return $\Phi_{init,r}$
end function

2.5.4 Estimating the Basin of Attraction

Finding the actual basin of attraction by probing each initial offset point to see if it converges close to the ground truth local optimum is extremely computationally expensive. Instead, we make use of the fact that the basin of attraction is usually a continuous region around the predicted most likely location, and make a rough estimate of the region. Algorithm 4 shows how we estimate the basin of attraction for an character example. We first probe

Algorithm 4 Estimating the basin of attraction

function EstimateBasin(A_r^{j*}, X_r^j)
  $Boundary \leftarrow \emptyset$
  for $dir \leftarrow 0°, 45°, 90°, \ldots, 315°$ **do**
    $Boundary \leftarrow Boundary \cup \{\text{ProbeBasinBoundary}(X_r^j, A_r^{j*}, dir)\}$
  end for
  return MakeRegion($Boundary$)
end function

the boundary of the basin in 8 directions for a limited distance centered at the predicted most likely location, then we connect the boundary points and make a closed region, using it as the rough estimate of the basin. Figure 2.22 shows an example of the estimated basin.

2.6 Stroke Correspondences and Match Sequences

Given two classes of similar Chinese characters with some in each class having their strokes labeled according to Section 2.1.2, we want to find the structures in the character images and classify them according to their class label. The structures are found through a sequence of matching affine-transformed templates to the character images using techniques in Section 2.3 and Sec-

Figure 2.22: Estimated basin for examples in Figure 2.21 based on Algorithm 4.

tion 2.4. After the structures in the characters are located, we mask out parts of the image that do not contribute much information to the class label, and classify the examples based on the important part of the image using a conventional statistical classifier.

In order to generate a sequence of template matching operations for use in finding the structures in the image, we first create correspondences between individual strokes in the two classes of characters. After the correspondences are found, we identify strokes that do not have a good counterpart in the other class, treating them as target strokes to look for. We then group strokes highly correlated to each other together, using them to create templates for the matching. A sequence for matching the templates is then generated so that the templates are used in a hierarchical way to find the structures in the images.

2.6.1 Stroke correspondence and target strokes

Given the stroke parameters of structurally labeled training examples of two classes

$$\begin{aligned}
&\langle \mathbf{\Gamma}_1, \mathbf{\Gamma}_2 \rangle \\
&= \left\langle \begin{pmatrix} \Gamma_1^1 \\ \Gamma_1^2 \\ \vdots \\ \Gamma_1^{n_1} \end{pmatrix}, \begin{pmatrix} \Gamma_2^1 \\ \Gamma_2^2 \\ \vdots \\ \Gamma_2^{n_2} \end{pmatrix} \right\rangle \qquad (2.49) \\
&= \left\langle \begin{pmatrix} \left\langle \Psi_{1,1}^1, \ldots, \Psi_{l_1,1}^1, w_1^1 \right\rangle \\ \left\langle \Psi_{1,1}^2, \ldots, \Psi_{l_1,1}^2, w_1^2 \right\rangle \\ \vdots \\ \left\langle \Psi_{1,1}^{n_1}, \ldots, \Psi_{l_1,1}^{n_1}, w_1^{n_1} \right\rangle \end{pmatrix}, \begin{pmatrix} \left\langle \Psi_{1,2}^1, \ldots, \Psi_{l_2,2}^1, w_2^1 \right\rangle \\ \left\langle \Psi_{1,2}^2, \ldots, \Psi_{l_2,2}^2, w_2^2 \right\rangle \\ \vdots \\ \left\langle \Psi_{1,2}^{n_2}, \ldots, \Psi_{l_2,2}^{n_2}, w_2^{n_2} \right\rangle \end{pmatrix} \right\rangle \qquad (2.50)
\end{aligned}$$

we compute their mean values among the structurally labeled training examples

$$\langle \bar{\mathbf{\Gamma}}_1, \bar{\mathbf{\Gamma}}_2 \rangle = \left\langle \begin{pmatrix} \bar{\Psi}_{1,1} \\ \bar{\Psi}_{2,1} \\ \vdots \\ \bar{\Psi}_{l_1,1} \\ \bar{w}_1 \end{pmatrix}, \begin{pmatrix} \bar{\Psi}_{1,2} \\ \bar{\Psi}_{2,2} \\ \vdots \\ \bar{\Psi}_{l_2,2} \\ \bar{w}_2 \end{pmatrix} \right\rangle. \qquad (2.51)$$

For each pair of strokes from the two classes, $\mathbf{\Psi}_{i,1}$ from class 1 and $\mathbf{\Psi}_{j,2}$ from class 2, we compute the covariance among the stroke parameters, treating

them as if they were of the same multivariate normal distribution:

$$
\begin{aligned}
&\Sigma_{i,j} \\
&= \operatorname{cov}\left(\begin{array}{cccccccc} \Psi_{i,1}^1 & \Psi_{i,1}^2 & \cdots & \Psi_{i,1}^{n_1} & \Psi_{j,2}^1 & \Psi_{j,2}^2 & \cdots & \Psi_{j,2}^{n_2} \end{array} \right) \qquad (2.52) \\
&= \operatorname{cov}\left(\begin{array}{cccccccc} x_{1,i,1}^1 & x_{1,i,1}^2 & \cdots & x_{1,i,1}^{n_1} & x_{1,j,2}^1 & x_{1,j,2}^2 & \cdots & x_{1,j,2}^{n_2} \\ y_{1,i,1}^1 & y_{1,i,1}^2 & \cdots & y_{1,i,1}^{n_1} & y_{1,j,2}^1 & y_{1,j,2}^2 & \cdots & y_{1,j,2}^{n_2} \\ x_{2,i,1}^1 & x_{2,i,1}^2 & \cdots & x_{2,i,1}^{n_1} & x_{2,j,2}^1 & x_{2,j,2}^2 & \cdots & x_{2,j,2}^{n_2} \\ y_{2,i,1}^1 & y_{2,i,1}^2 & \cdots & y_{2,i,1}^{n_1} & y_{2,j,2}^1 & y_{2,j,2}^2 & \cdots & y_{2,j,2}^{n_2} \end{array} \right) \qquad (2.53)
\end{aligned}
$$

where in Equation (2.53) each column $\begin{pmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \end{pmatrix}$ represents the coordinates of the two end points of one stroke in one example. There are n_1 examples in class 1, and n_2 examples in class 2.

With the means $\bar{\Psi}_{i,1}$ and $\bar{\Psi}_{j,2}$ computed using Equation (2.51) and the covariance $\Sigma_{i,j}$ computed using Equation (2.53), we compute the squared Mahalanobis distance between the pair of strokes $\mathbf{\Psi}_{i,1}$ and $\mathbf{\Psi}_{j,2}$:

$$
D_M(\mathbf{\Psi}_{i,1}, \mathbf{\Psi}_{j,2}) = \left(\bar{\Psi}_{i,1} - \bar{\Psi}_{j,2}\right)^{\mathsf{T}} \Sigma_{i,j}^{-1} \left(\bar{\Psi}_{i,1} - \bar{\Psi}_{j,2}\right). \qquad (2.54)
$$

The Mahalanobis distance measures the covariance-normalized distance in standard deviations.

Using the squared Mahalanobis distance computed in Equation (2.54), we find the stroke correspondences between stokes in two classes of Chinese

characters using Algorithm 5. For each stroke $\mathbf{\Psi}_{i,1}$ in the first character

Algorithm 5 Finding stroke correspondences between two character classes.

```
function StrokeCorrespondence(Γ_1, Γ_2)
    C ← ∅                                          ▷ Set of correspondences
    D ← []                                         ▷ Squared Mahalanobis distances
    for all Ψ_{i,1} ∈ Γ_1 do
        d_{i,j*} ← min_j D_M(Ψ_{i,1}, Ψ_{j,2})
        if d_{i,j*} < d_θ then                     ▷ Maximal threshold distance
            j* ← arg min_j D_M(Ψ_{i,1}, Ψ_{j,2})
            if ⟨·, j*⟩ ∈ C then
                if d_{i,j*} < D[⟨·, j⟩] then
                    C ← C \ {⟨·, j*⟩}
                    delete D[⟨·, j*⟩]
                    D[⟨i, j*⟩] ← d_{i,j*}
                    C ← C ∪ {⟨i, j*⟩}
                end if
            else
                D[⟨i, j*⟩] ← d_{i,j*}
                C ← C ∪ {⟨i, j*⟩}
            end if
        end if
    end for
    return C
end function
```

class, we find the stroke $\mathbf{\Psi}_{j^*,2}$ in the second class closest to $\mathbf{\Psi}_{i,1}$ in squared Mahalanobis distance. If the distance d_{i,j^*} is under the maximal threshold distance d_θ, we consider the two strokes to be potentially corresponding to each other. If $\mathbf{\Psi}_{j^*,2}$ does not have a corresponding stroke yet, we create the correspondence $\langle i, j^* \rangle$. If $\mathbf{\Psi}_{j^*,2}$ already has a correspondence, and d_{i,j^*} is smaller than the distance between $\mathbf{\Psi}_{j^*,2}$ and its corresponding stroke, we replace the existing correspondence with $\langle i, j^* \rangle$. The maximal threshold distance d_θ is chosen so that it is adequate for the domain. When the

algorithm completes, all strokes without corresponding strokes in the other class are treated as *target strokes* that bear the most information about the difference between the classes.

After the correspondences are found, we re-number the strokes according to their correspondence:

$$\mathbf{\Gamma}'_1 = \langle \mathbf{\Psi}_{c_1(1),1}, \ldots, \mathbf{\Psi}_{c_1(k),1}, \mathbf{\Psi}_{c_1(k+1),1}, \ldots, \mathbf{\Psi}_{c_1(l_1),1}, \mathbf{w}_1 \rangle, \quad (2.55)$$

$$\mathbf{\Gamma}'_2 = \langle \mathbf{\Psi}_{c_2(1),2}, \ldots, \mathbf{\Psi}_{c_2(k),2}, \mathbf{\Psi}_{c_2(k+1),2}, \ldots, \mathbf{\Psi}_{c_2(l_2),2}, \mathbf{w}_2 \rangle, \quad (2.56)$$

where $c_1(\cdot)$ and $c_2(\cdot)$ map the stroke numbers according to their correspondence such that $\mathbf{\Psi}_{c_1(i),1}$ and $\mathbf{\Psi}_{c_2(i),2}$ correspond to each other for $i \leq k$, and k is the number of common strokes shared between the two classes. $\mathbf{\Psi}_{c_1(k+1),1}, \ldots, \mathbf{\Psi}_{c_1(l_1),1}$ and $\mathbf{\Psi}_{c_2(k+1),2}, \ldots, \mathbf{\Psi}_{c_2(l_2),2}$ are the strokes unique to each class, and are treated as target strokes.

2.6.2 Hierarchical clustering of strokes

With the common strokes and stroke correspondences identified for the two classes of characters, we divide the common strokes into groups by performing a hierarchical clustering on them. These groups will be used to generate character templates for use in matching the structures in character images as described below.

We use a mutual information-based hierarchical clustering algorithm as described in [15]. The algorithm is outlined in Algorithm 6. In Algorithm 6, $D_{MI}(\cdot)$ is the mutual information-based distance measure described below.

Algorithm 6 Hierarchical clustering

function HIERARCHICALCLUSTERING($\mathbf{\Psi}$)
  $C \leftarrow \{\Psi_{c(1)}, \Psi_{c(2)}, \ldots, \Psi_{c(k)}\}$
  while $|C| > 1$ **do**
    $c_i^*, c_j^* \leftarrow \arg\min_{c_i, c_j \in C; c_i \neq c_j} D_{MI}(c_i, c_j)$
    $C \leftarrow C \setminus \{c_i, c_j\}$
    $C \leftarrow C \cup \{\{c_i, c_j\}\}$
  end while
  return C
end function

Distance measure

The distance measure $D_{MI}(\cdot)$ used in Algorithm 6 is based on mutual information, and is defined as follows:

$$D_{MI}(X, Y) = 1 - \frac{I(X;Y)}{h(X,Y)}, \tag{2.57}$$

where $I(X;Y)$ is the mutual information between X and Y defined as

$$I(X;Y) = h(X) + h(Y) - h(X,Y), \tag{2.58}$$

$h(X)$ is the joint differential entropy of X, $h(Y)$ is the joint differential entropy of Y, and $h(X,Y)$ is the joint differential entropy of X, Y.

We model the strokes common to both classes using a multivariate normal distribution, and thus the joint differential entropy is easily computed as

$$h(X) = \frac{1}{2} \log \left((2\pi e)^d \left| \Sigma_X \right| \right), \tag{2.59}$$

where d is the number of variables in X, which in our case is 4 times the

number of strokes. Σ_X is the sub-matrix corresponding to variables in X of the covariance matrix Σ, and $|\Sigma_X|$ is the determinant of Σ_X. Σ is the covariance matrix among the common strokes defined as follows:

$$\Sigma = \text{cov}\begin{pmatrix} \Psi^1_{c_1(1),1} & \Psi^2_{c_1(1),1} & \cdots & \Psi^{n_1}_{c_1(1),1} & \Psi^1_{c_2(1),2} & \Psi^2_{c_2(1),2} & \cdots & \Psi^{n_2}_{c_2(1),2} \\ \Psi^1_{c_1(2),1} & \Psi^2_{c_1(2),1} & \cdots & \Psi^{n_1}_{c_1(2),1} & \Psi^1_{c_2(2),2} & \Psi^2_{c_2(2),2} & \cdots & \Psi^{n_2}_{c_2(2),2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \Psi^1_{c_1(k),1} & \Psi^2_{c_1(k),1} & \cdots & \Psi^{n_1}_{c_1(k),1} & \Psi^1_{c_2(k),2} & \Psi^2_{c_2(k),2} & \cdots & \Psi^{n_2}_{c_2(k),2} \end{pmatrix}, \tag{2.60}$$

where each column represents the k common strokes for one example. There are n_1 examples in class 1, and n_2 examples in class 2. Σ is smoothed by adding $\frac{1}{100}\lambda_1\mathbf{I}$ to it when we compute the entropy in order to avoid numerical difficulties due to the underranked covariance matrix.

D_{ML} as defined above is a proper metric.

Theorem 1 (Kraskov and Grassberger [15, Theorem 5.1]) *The quantity*

$$D_{ML}(X,Y) = 1 - \frac{I(X;Y)}{h(X,Y)} \tag{2.61}$$

is a metric, with $D_{ML}(X,X) = 0$ *and* $D(X,Y) \leq 1$ *for all pairs* (X,Y).

Figure 2.23 shows the result of the hierarchical clustering of the strokes of the same character as Figure 2.2 using Algorithm 6. For simplicity Figure 2.23 shows the hierarchical clustering of all strokes in one class of characters, rather than shared common strokes between two classes of characters.

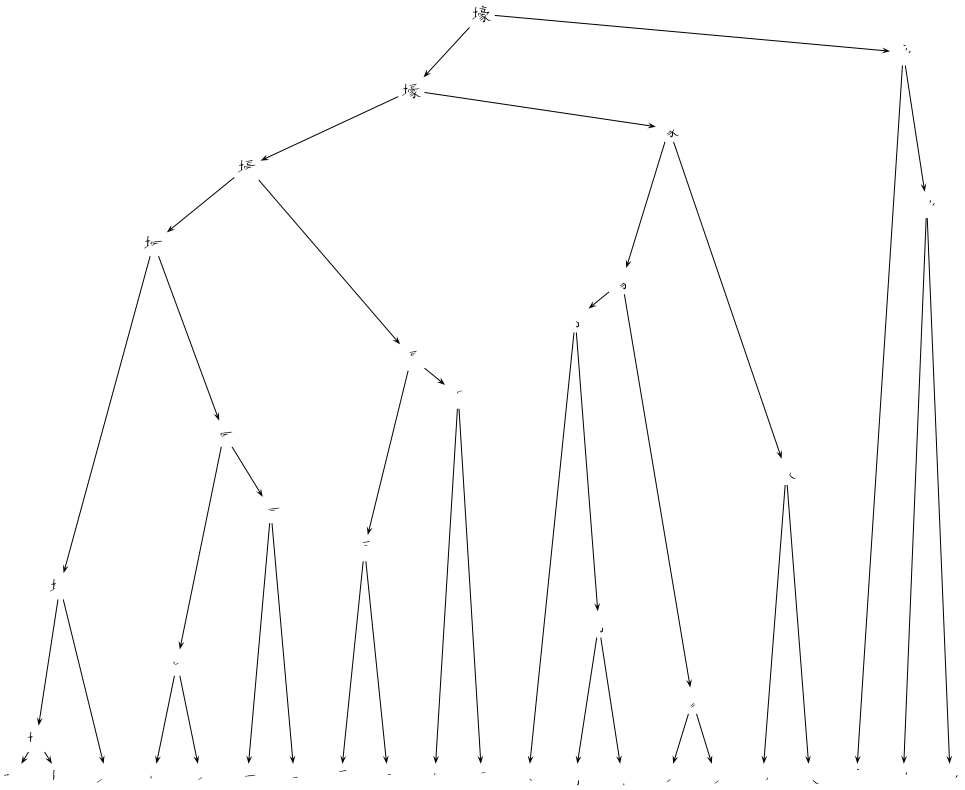


Figure 2.23: Result of the hierarchical clustering on a character using Algorithm 6. Compare to the decomposition by radical hierarchy in Figure 2.2.

While Figure 2.23 shows a different decomposition than Figure 2.2, it is nonetheless a reasonable one.

Strokes unique to each class

Given two classes of characters, Algorithm 6 only works for common strokes shared between the classes. We deal with the unique strokes to each class, i.e. the target strokes, by "attaching" them to the shared stroke closest to them in D_{MI}. Whenever a cluster is selected to generate a template for matching operations that includes a stroke that a target stroke attaches

to, the target stroke is also included in the template. Target strokes are rendered with half pixel intensity in the template.

2.6.3 Match sequence

Given a hierarchy generated by Algorithm 6, we make a sequence of sets of strokes that are used to make templates for the matching operations by traversing the hierarchy recursively. This is outlined in Algorithm 7. In Algorithm 7, TraverseInto$(\cdot)$ determines whether we should further split a cluster. We only further split a cluster if there are more than 3 strokes in the cluster. IncludeCluster$(\cdot)$ determines whether a cluster should be included in the sequence of match operations. Single-stroke clusters are only included if the stroke is sufficiently long or if a target stroke attaches to it. TraverseChild1First($child1$, $child2$) decides whether we should traverse *child1* or *child2* first. If one child contains a stroke attached by a target stroke and the other child does not, the child that does not contain such stroke is traversed first. If none of the children contains a stroke attached by a target stroke, the child further away to the strokes attached by the target stroke in D_{MI} is traversed first.

Figure 2.24 shows the match sequence according to the hierarchy in Figure 2.23 using Algorithm 7. For simplicity, instead of showing common strokes from two classes of character, Figure 2.24 only shows the strokes from a single class. The left three strokes (the three strokes in the last template in the sequence) are the target strokes.

Algorithm 7 Generating the match sequence using the result of hierarchical clustering.

```
procedure MatchSequenceRec(hier, current, target, result)
    child1, child2 ← hier[current]
    if TraverseChild1First(child1, child2) then
        if IncludeCluster(child1) then
            result ← result + child1
        end if
        if TraverseInto(child1) then
            MatchSequenceRec(hier, child1, target, result)
        end if
        if IncludeCluster(child2) then
            result ← result + child2
        end if
        if TraverseInto(child2) then
            MatchSequenceRec(hier, child2, target, result)
        end if
    else
        if IncludeCluster(child2) then
            result ← result + child2
        end if
        if TraverseInto(child2) then
            MatchSequenceRec(hier, child2, target, result)
        end if
        if IncludeCluster(child1) then
            result ← result + child1
        end if
        if TraverseInto(child1) then
            MatchSequenceRec(hier, child1, target, result)
        end if
    end if
end procedure
function MatchSequence(hier, target)
    result ← [Root(hier)]
    MatchSequenceRec(hier, Root(hier), target, result)
    return result
end function
```

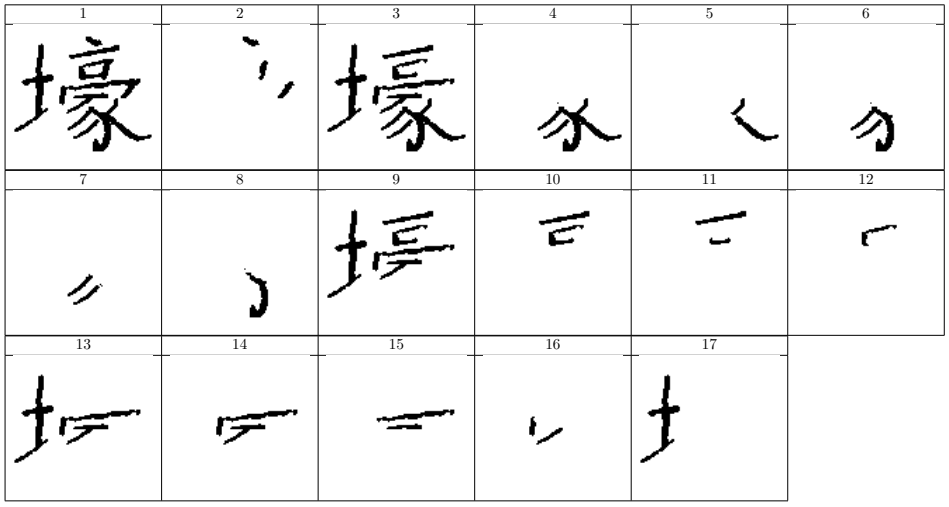


Figure 2.24: The match sequence according to the hierarchy in Figure 2.23 using Algorithm 7.

2.7 Target Region and Gradient Features

After we find the structures in a character image by performing the match operations in the sequence generated in Section 2.6.3 using techniques in Section 2.3 and Section 2.4, we generate discriminative features for the statistical classifier. This is done by first preparing the image for the target region that contains the discriminative structure in the image as described in Section 2.7.1, then collect smoothed histograms of gradients as described in Section 2.7.3.

2.7.1 Target region

Modeling the error of the match for each stroke

With the strokes parameters found by performing the template matchings in the sequence on the structurally labeled training examples and the ground

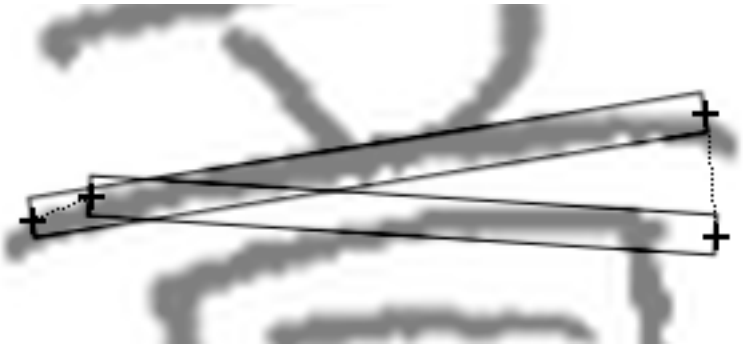


Figure 2.25: The error between the ground truth stroke and the stroke found by performing the template matchings in the sequence.

truth stroke parameters obtained using Section 2.4.2, we build an error model for each stroke. Figure 2.25 shows an example of the difference between the stroke found through the template matchings and the ground truth stroke.

For each stroke, we model the difference between the end points of the ground truth stroke and the stroke found using the template matchings in terms of offsets in x and y coordinates. Collecting these offsets among the structurally labeled training examples and combining both end points, we compute the mean squared errors for the two coordinate directions:

$$\sigma_x^2 = \frac{1}{2n}\left(\sum_{i=1}^{n}(x_{gt1}^i - x_{m1}^i)^2 + \sum_{i=1}^{n}(x_{gt2}^i - x_{m2}^i)^2\right), \quad (2.62)$$

$$\sigma_y^2 = \frac{1}{2n}\left(\sum_{i=1}^{n}(y_{gt1}^i - y_{m1}^i)^2 + \sum_{i=1}^{n}(y_{gt2}^i - y_{m2}^i)^2\right), \quad (2.63)$$

where x_{gtk}^i is the ground truth x coordinate of the kth end point in the ith structurally labeled example, x_{mk}^i is the x coordinate of the kth end point found through template matchings in the ith structurally labeled example,

y^i_{gtk} is the ground truth y coordinate of the kth end point in the ith structurally labeled example, and y^i_{mk} is the y coordinate of the kth end point found through template matchings in the ith structurally labeled example. We then define the distribution that the actual stroke pixel in the character image corresponding to each stroke pixel (x_0, y_0) found using template matchings is located at (x, y) in the image as

$$f_{x_0,y_0}(x,y) = \frac{1}{2\pi|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-x_0 \quad y-y_0)\Sigma^{-1}\binom{x-x_0}{y-y_0}}, \tag{2.64}$$

where

$$\Sigma = \begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{pmatrix}. \tag{2.65}$$

The probability that a pixel in the image corresponds to any stroke pixel in a specific stroke is

$$f_s(x,y) = \sum_{(x_0,y_0)\in S} f_{x_0,y_0}(x,y), \tag{2.66}$$

where S is the set of stroke pixels in stroke s found using template matchings.

Equation (2.66) can be computed efficiently by applying an axis-aligned Gaussian filter with standard deviations σ_x and σ_y for the two axes respectively to the stroke image rendered using the stroke parameters obtained through the template matchings. Figure 2.26 shows the result of smoothing each of the rendered strokes using the Gaussian filter of its respective standard deviations.

Figure 2.26: Rendered strokes smoothed differently according to their error models.

Discounting common strokes

With the distribution in Equation (2.66) defined for each stroke, we compute the likelihood ratio that pixel (x, y) belongs to a target stroke as

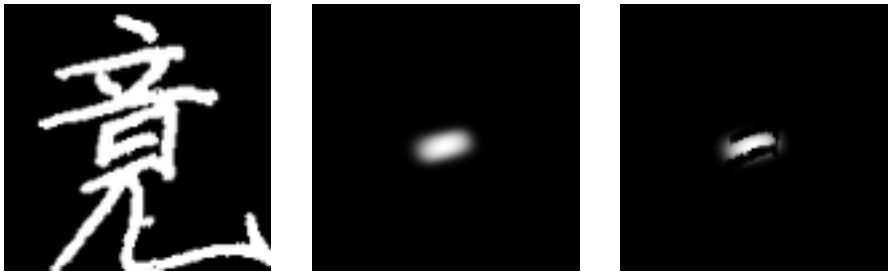
$$L_T(x, y) = \frac{\sum_{s \in T} f_s(x, y)}{\sum_s f_s(x, y)}, \tag{2.67}$$

where $f_s(x, y)$ is the distribution for stroke s and T is the set of target strokes. We than normalize $L_T(x, y)$ into

$$\tilde{L}_T(x, y) = \frac{L_T(x, y)}{\max_{x,y} L_T(x, y)}, \tag{2.68}$$

such that the maximal value in $\tilde{L}_T(x, y)$ is normalized to 1.

We discount pixels in the character image belonging to common strokes according to $\tilde{L}_T(x, y)$ in Equation (2.68). Figure 2.27 shows the character image with pixels in common strokes discounted.



(a) (b) (c)

Figure 2.27: A character image discounted using the normalized likelihood ratio in Equation (2.68). Figure 2.27a is the original character image. The target stroke is the horizontal stroke in the middle box. Figure 2.27b is the normalized likelihood ratio computed using the Gaussian filtered rendered strokes in Figure 2.26. Figure 2.27c is the resulting image with pixels in the common strokes discounted, leaving only pixels likely to belong to the target stroke.

Cropping the target region

After discounting pixels belonging to common strokes, we cut an axis-aligned rectangular window containing the target strokes out of the discounted character image. This rectangular window will be used to generate gradient features for the statistical classifier. The window is decided such that all of the end points of the target strokes are contained in it with high probability. Specifically, the boundaries of the target window is 3 standard deviations away from the end points of the target strokes, where the standard deviations are computed using Equation (2.63). Figure 2.28 shows the target window for the example in Figure 2.27.

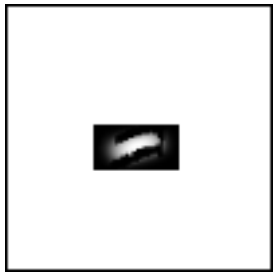


Figure 2.28: The target window in relation to the whole image for the discounted character image in Figure 2.27.

2.7.2 The original WDH features

The weighted direction code histogram (WDH) features introduced by Kimura [14] are one of the most successful set of features for offline handwritten Chinese character recognition. This set of features are suitable for binary images of whole characters. The WDH features are generated as follows:

1. **Nonlinear normalization**

   Before generating the WDH features, the image of the character is stretched nonlinearly to normalize the line density [31]. Figure 2.29 shows the effect of nonlinear normalization. The histograms of edges are equalized in each coordinate direction. We do not use nonlinear normalization in our approach.

2. **Contour detection**

   The contour (edge) of the character is detected as the set of all stroke pixels adjacent to any blank pixel in its 8 directions. Figure 2.30 shows

(a) Original (b) Normalized

Figure 2.29: Effect of the nonlinear normalization. Figure 2.29a is the original image. Figure 2.29b is the result of applying the nonlinear normalization before generating the WDH features. The histograms of edges are equalized in each coordinate direction.

Figure 2.30: Detected contour pixels for the normalized image in Figure 2.29b.

the contour pixels detected for the normalized image in Figure 2.29b.

3. **Chain coding**

   For the 3×3 block around each contour pixel, lines of the 16 possible orientations are detected by finding pairs of contour pixels in that orientation in the block.

4. **Local direction histograms**

   The normalized character image is divided into 169 blocks, with 13

blocks on each side. The number of contour pixels in each orientation is accumulated in each block, producing 169 local direction code histograms of 16 directions.

5. **Down sampling**

   The histograms in 13×13 blocks are smoothed using a 5×5 Gaussian filter. The spatial resolution is then reduced from 13×13 to 7×7 by down sampling every other block. Similarly, the histograms in the 16 directions are smoothed using a weight vector $\begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$. The directional resolution is then reduced from 16 to 8. This produces a feature vector of size 392, in 7 horizontal, 7 vertical, and 8 directional resolutions.

6. **Square root of feature values**

   Square root of each feature value is taken to make the distribution of features Gaussian-like [30, 11].

2.7.3 Our variant of gradient features

The gradient features that we use in our approach are similar to the WDH features, with a few key differences:

- **No nonlinear normalization**

  The nonlinear normalization for the WDH features is suitable for moving the same strokes in different examples to approximately the same locations in the whole character image. However, since we use our own method to find and align strokes in the characters, and discount strokes

irrelevant to classification, their histogram-based normalization is not suitable and not used in our approach.

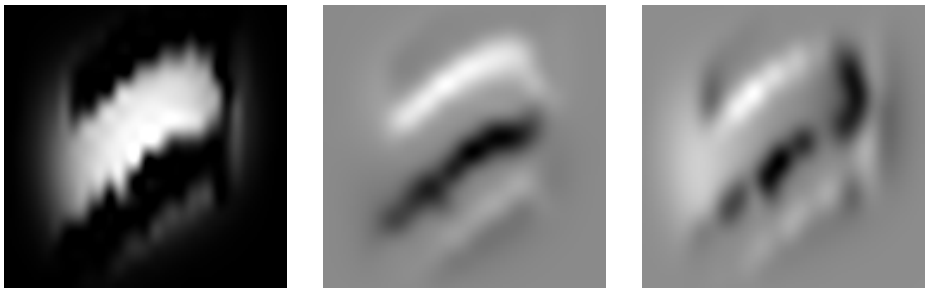
- **Gaussian-smoothed gradient operators**

  The contour detection and chain code in the WDH features only works on binary images. Since we discount irrelevant parts of the character, the image is no longer binary. Therefore, we use Gaussian-smoothed gradient operators to generate the direction code features.

- **Variable number and size of blocks**

  Since we cut a target window around the target strokes, the optimal number and size of blocks can be very different than that for the whole character image, and can be different for different pairs of characters. For each pair of characters, we use cross validation to select the optimal number and size of blocks to compute the histogram.

With the properly discounted and cropped target window image generated according to Section 2.7.1, we generate gradient-based features for classification. The target window is first normalized into a square, and Gaussian-smoothed gradient operators are applied to the normalized image. The size of the square equals the number of blocks we divide the image into times the size of each block. Figure 2.31 shows the normalized target region and the gradients in the two directions. With the strength of the gradients in x and y directions, we compute the strength of edge in each of the 16 directions for each pixel in the normalized target window.



(a) Normalized target window (b) Vertical gradients (c) Horizontal gradients

(d)

(e)

Figure 2.31: Figure 2.31a shows the normalized target window. Figure 2.31b shows the vertical gradients (horizontal edges) detected in Figure 2.31a. Figure 2.31c shows the horizontal gradients (vertical edges) detected in Figure 2.31a. Figure 2.31d and Figure 2.31e are the operators used in detecting the gradients.

Once the strengths of edges are computed, we divide the target window into blocks, and the computation of histogram features, including accumulating the local gradient histograms, downsampling, and taking square root of the resulting features, is exactly the same as that for the WDH features in Section 2.7.2.

The number of blocks the image is divided into, the size of each block, and the standard deviation of the Gaussian smoothing are parameters that we optimize using the technique in Section 2.8.

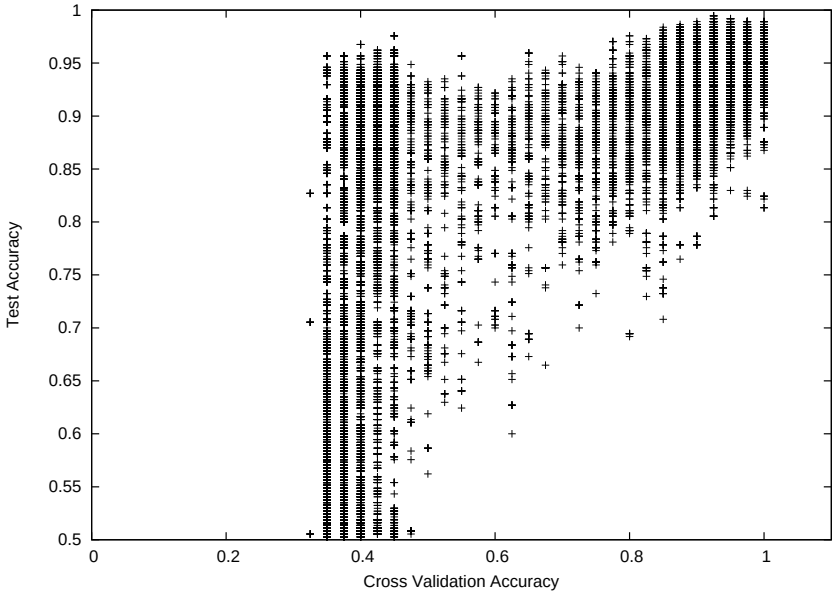
2.8 SVM Model Selection

After we generate the gradient features for the target regions, we use the support vector machine (SVM) to train a discriminative classifier. The SVM implementation that we use is the LIBSVM [2] package.

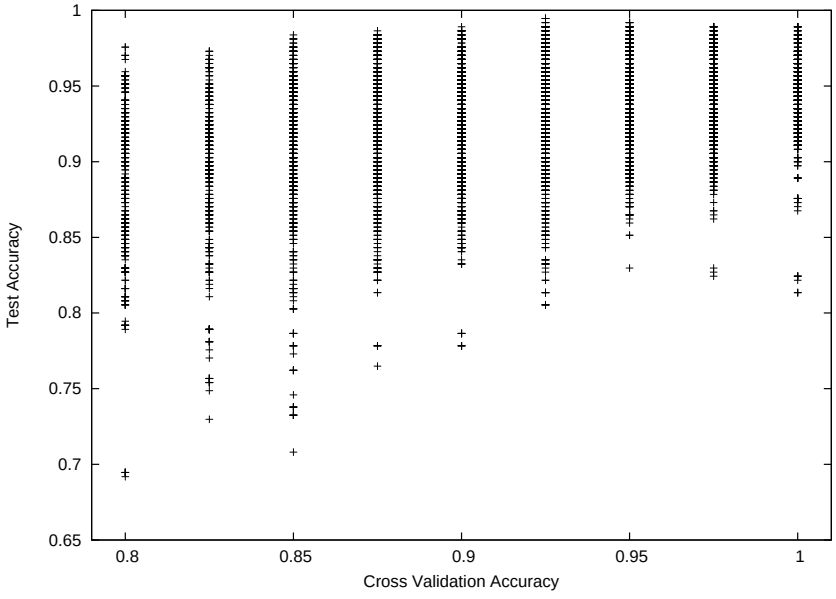
There are many parameters to optimize for the discriminative classifier in order to achieve a good generalization performance. For the RBF SVM, there are two kernel parameters, γ, the width of the radial basis function, and C, the cost of the soft loss. Additionally, we also need to choose the parameters for generating the gradient features in Section 2.7.3, including the number of blocks to divide the image, the size of each block, and the amount of smoothing applied to the gradient filter.

One of the most commonly used criterion for model selection is the cross validation accuracy. By using the cross validation accuracy on the training examples, one aims to predict the generalization accuracy of the given model. However, besides having an accurate prediction of the generalization accuracy, we also want the prediction to be confident. With this many parameters to optimize, we do not have nearly enough training examples to confidently select a good model using the cross validation accuracy.

Figure 2.32 plots the cross validation accuracy versus the actual test accuracy on the hold-out set for a pairwise classification task. The classifier is an SVM with the RBF kernel. 40 training examples are sampled from two classes. The parameters to choose include the kernel parameters and the parameters for generating the gradient features. The correlation coefficient between the cross validation accuracy and the test accuracy on the hold-



(a)



(b)

Figure 2.32: Cross validation accuracy vs. the test accuracy on the hold-out set. The Pearson product-moment correlation coefficient is 0.55. Figure 2.32b zooms into the region with high cross validation accuracy.

Cross Validation Accuracy	Mean Test Accuracy	Standard Deviation of Test Accuracy
0.5–0.6	0.8161	0.0854
0.6–0.7	0.8563	0.0650
0.7–0.8	0.8668	0.0493
0.8–0.9	0.9104	0.0464
0.9–1.0	0.9364	0.0283

(a) Using cross validation accuracy

Cross Validation Mean Posterior Probability	Mean Test Accuracy	Standard Deviation of Test Accuracy
0.5–0.6	0.8398	0.1125
0.6–0.7	0.8880	0.0425
0.7–0.8	0.9224	0.0379
0.8–0.9	0.9392	0.0262
0.9–1.0	0.9413	0.0191

(b) Using cross validation posterior probability

Table 2.1: Means and standard deviations of the test accuracy on the hold-out set at different levels of cross validation accuracy and mean cross validation posterior probability.

out set is 0.55. As can be seen in Figure 2.32, because the set of training examples is small, there are not many distinctions that the cross validation accuracy can make. Furthermore, even if we choose a model with a cross validation accuracy of 100%, the actual test accuracy on the hold-out set can still range anywhere from around 80% to 100%. Table 2.1a shows the means and standard deviations of the test accuracies on the hold-out set at different levels of cross validation accuracy.

Instead of the cross validation accuracy, we propose to use the *cross validation posterior probability* as the criterion for model selection. Support vector machines can be trained to compute the posterior probabilities for the class labels given the input features. Platt proposes to compute the

posterior probability by fitting a sigmoid function to the kernel value [22]:

$$P(y=1|x) \approx P_{A,B}(f) \tag{2.69}$$

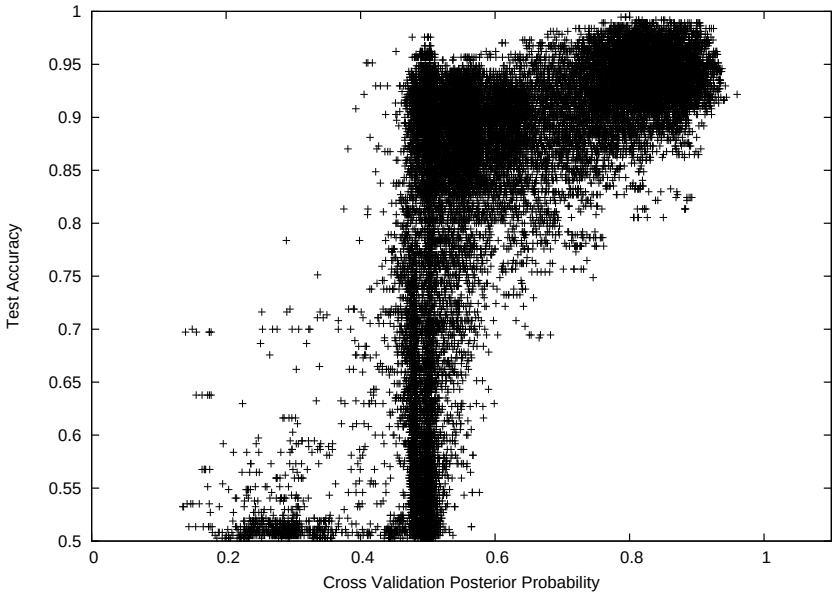
$$\equiv \frac{1}{1+\exp(Af+B)} \text{ , where } f = \mathrm{f}(x). \tag{2.70}$$

$\mathrm{f}(x)$ is the kernel function.

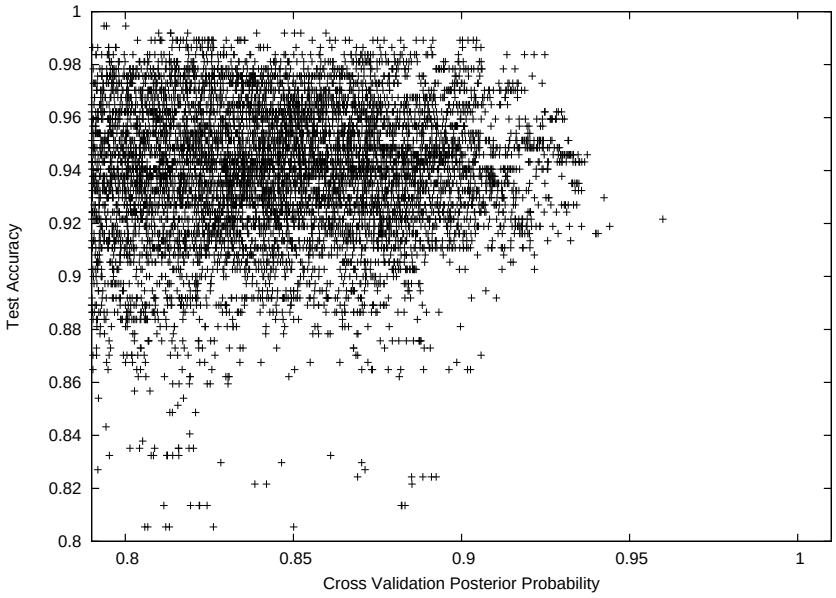
We use the implementation of Platt's posterior probability output in LIBSVM [19]. When we perform cross validation, instead of simply computing the testing accuracy on the validation set, we compute the posterior probability of each example in the validation set belonging to its correct class. We then use the joint posterior probability of each example in the training set belonging to the correct class as our criterion for model selection.

Figure 2.33 plots the geometric mean of the cross validation posterior probability versus the actual test accuracy on the hold-out set for the same pairwise classification task. The correlation coefficient between the cross validation mean posterior probability and the test accuracy on the hold-out set is 0.63. The test accuracy for most data points of which the geometric mean of the cross validation posterior probability is greater than 90% is at least 90% in Figure 2.33.

Table 2.1b shows the means and standard deviations of the test accuracies on the hold-out set at different levels of cross validation mean posterior probability. Comparing to Table 2.1a, the standard deviations of the test accuracies at higher levels of cross validation mean posterior probability are lower than the standard deviations of the test accuracies at higher levels



(a)

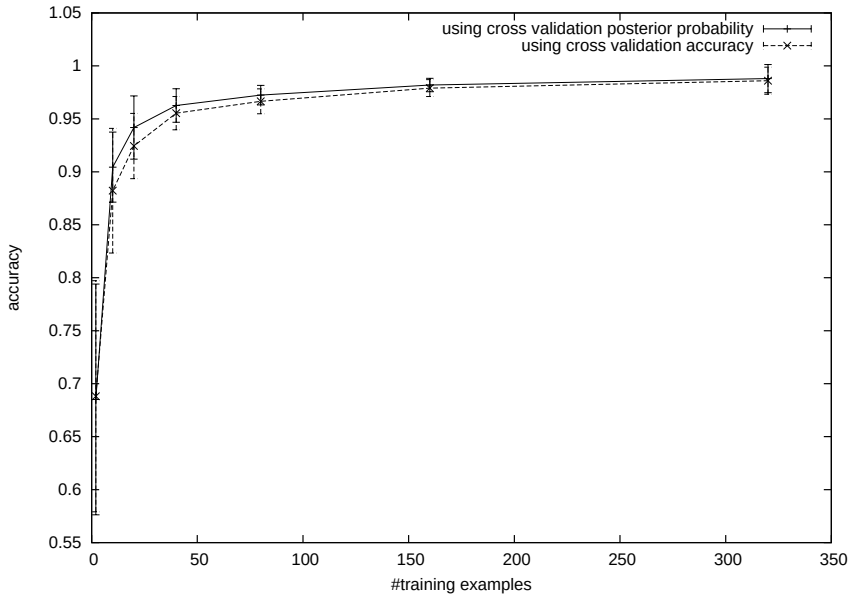


(b)

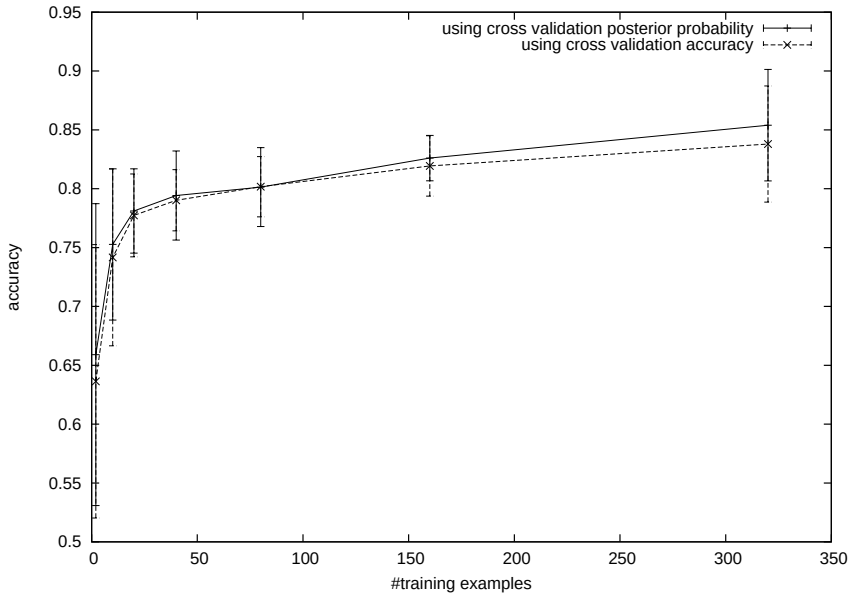
Figure 2.33: Geometric mean of the cross validation posterior probability vs. the test accuracy on the hold-out set. The Pearson product-moment correlation coefficient is 0.63. Figure 2.32b zooms into the region with high cross validation posterior probability.

of cross validation accuracy. Therefore, using the cross validation posterior probability results in a more confident selected model.

In Figure 2.34, we plot the learning curves for the pair 狠 vs. 狼 and the pair 睛 vs. 晴 using both the cross validation posterior probability and the cross validation accuracy for model selection. It can be seen that using the cross validation posterior probability as the model selection criterion results in slightly better models being selected at all levels of the training set sizes.



(a) Learning curve for the pair 狠 vs. 狼



(b) Learning curve for the pair 晴 vs. 晴

Figure 2.34: Learning curves for our method using the cross validation posterior probability and the cross validation accuracy for model selection. The errorbars are standard deviations of the accuracies.

Chapter 3

Theoretical Framework

Our approach concerns classification problems for objects that are both *structured* and *structurally similar*.

3.1 Problem Definition

3.1.1 Structured objects

A structured object is a composition of the noisy renditions of its correlated constituents. Let $\mathbf{x} \in \mathcal{X}$ be a structured object. It can be written as

$$\begin{aligned} \mathbf{x} &= \text{Render}(\mathbf{m}^*, \theta^*) && (3.1) \\ &= \text{Render}(m_1^*, \theta_1^*) \oplus \text{Render}(m_2^*, \theta_2^*) \oplus \ldots \oplus \text{Render}(m_l^*, \theta_l^*), && (3.2) \end{aligned}$$

where $\text{Render}(m_i^*, \theta_i^*)$ realizes the model m_i^* using parameters θ_i^* and represents it in the space of $\mathcal{X}$. $\oplus$ composes the rendered constituents, and is domain dependent. One of the simplest and commonly used example of the

composition operator is numerical addition of vector values. The parameters $\theta^* = \theta_1^*, \theta_2^*, \ldots, \theta_l^*$ are correlated, and sampled from the joint distribution

$$\theta^* \sim p_{\theta^*}. \tag{3.3}$$

In general, given an example $\mathbf{x}$, we do not know its underlying parameters $\theta_1^*, \theta_2^*, \ldots, \theta_l^*$. The only observable is the vector $\mathbf{x}$ itself.

Modeling structured objects

The constituents $m_1^*, m_2^*, \ldots, m_l^*$ are usually too complex to model exactly. Instead, we use simplified proxies $m_1, m_2, \ldots, m_l$, and treat the discrepancy as noise:

$$\begin{aligned}
\mathbf{x} &= \text{Render}(m_1^*, \theta_1^*) \oplus \text{Render}(m_2^*, \theta_2^*) \oplus \ldots \oplus \text{Render}(m_l^*, \theta_l^*) && (3.4)\\
&= (\text{Render}(m_1, \theta_1) + \xi_1) \oplus (\text{Render}(m_2, \theta_2) + \xi_2) \oplus \ldots \\
&\quad \oplus (\text{Render}(m_l, \theta_l) + \xi_l) && (3.5)\\
&= (\text{Render}(m_1, \theta_1) \oplus \text{Render}(m_2, \theta_2) \oplus \ldots \\
&\quad \oplus \text{Render}(m_l, \theta_l)) + \xi_l && (3.6)\\
&= \text{Render}(\mathbf{m}, \theta) + \xi. && (3.7)
\end{aligned}$$

The parameters $\theta = \theta_1, \theta_2, \ldots, \theta_l$ are correlated, and sampled from the joint distribution

$$\theta \sim p_\theta. \tag{3.8}$$

The inverse function $\text{Render}^{-1} : \mathbf{m} \to \mathbf{x}$ is not necessarily accessible.

Even if it exists, it can be difficult to compute or unreliable. The use of composition operations $\oplus$ and the existence of noise ξ further adds to the difficulty of obtaining Render^{-1}.

3.1.2 Structurally similar objects

Consider two classes of structured objects

$$\mathbf{X}_1 = \mathbf{x}_1^1, \mathbf{x}_1^2, \ldots, \mathbf{x}_1^{n_1} \tag{3.9}$$

and

$$\mathbf{X}_2 = \mathbf{x}_2^1, \mathbf{x}_2^2, \ldots, \mathbf{x}_2^{n_2}, \tag{3.10}$$

where each example $\mathbf{x}_i^j$ is a structured object

$$\mathbf{x}_i^j = \text{Render}(m_{i,1}*, \theta_{i,1}^{j*}) \oplus \text{Render}(m_{i,2}*, \theta_{i,2}^{j*}) \oplus \ldots \oplus \text{Render}(m_{i,l_i}*, \theta_{i,l_i}^{j*}), \tag{3.11}$$

and the parameters are sampled from the joint distributions

$$\theta_i^* \sim p_i^*. \tag{3.12}$$

$\mathbf{X}_1$ and $\mathbf{X}_2$ are considered structurally similar if most of the constituents in each class has a similar counterpart in the other class. Specifically, there exist mapping functions $c_1(.)$ and $c_2(.)$ such that

$$m_{1,c_1(j)}^* = m_{2,c_2(j)}^*, \quad 1 \leq j \leq l_c, \tag{3.13}$$

where $l_c \leq l_1, l_2$, and l_c is close to both l_1 and l_2. Furthermore, the joint distribution $p^*_{\theta_{c1}}$ for the parameters $\theta^*_{1,c_1(1)}, \theta^*_{1,c_1(2)}, \ldots, \theta^*_{1,c_1(l_c)}$ and the joint distribution $p^*_{\theta_{c2}}$ of the parameters $\theta^*_{2,c_2(1)}, \theta^*_{2,c_2(2)}, \ldots, \theta_{2,c_2(l_c)}$ are similar.

The rendition of structurally similar objects in $\mathcal{X}$ can be very similar because the distributions for most of its constituents are similar for both classes. The constituents that do not have similar counterparts in the other class are the discriminating constituents.

In practice, instead of the true underlying models m^*, we use the simplified proxies m.

3.1.3 Classification

Given two classes of structurally similar objects, we want to learn a model to classify the examples according to their class labels. Algorithm 8 outlines the abstract learning task.

Algorithm 8 Learning to classify structurally similar objects

procedure LEARN($\mathbf{X}_1, \mathbf{X}_2$) ▷ $\mathbf{X}_1, \mathbf{X}_2$ are structurally labeled examples
Identify correspondences between constituents in the two classes.
Hierarchically cluster the constituents based on their mutual information.
Construct features based on the clusters in hierarchy.
Select a subset of features and form an order of evaluation for them.
Optimize parameters for evaluating the features.
Learn a discriminative classifier based on information about the discriminating constituents.
end procedure

Algorithm 9 outlines the procedure that classifies unknown test examples.

Algorithm 9 Learning to classify structurally similar objects

procedure CLASSIFY($\mathbf{x}$) ▷ $\mathbf{x}$ is an unknown test example.
Evaluate the learned features in order.
Classify $\mathbf{x}$ using the discriminative classifier using information about the discriminating constituents.
end procedure

3.1.4 Requirements

There are a few requirements to apply our framework:

Structured objects The objects must be composed of correlated constituents.

Structurally similar objects Most of the constituents must have similar counterparts in the other class.

Models for the constituents We must have reasonable parameterized models for the constituents.

Structurally labeled examples We need a small set of examples with the parameters of their constituents annotated.

Reduced representation for sets of constituents We need to be able to represent sets of correlated constituents with reduced numbers of parameters.

Conditional probabilistic models for parameters We need to be able to build conditional probabilistic models between parameters of sets of constituents.

Differentiable loss function The mismatch between the rendition of the modeled constituents and an actual example has to be differentiable or empirically differentiable with respect to the parameters of the the model.

Explanations in the examples discountable There needs to be a way of discounting information in the representation of examples that is explained by the modeled constituents.

3.2 Features

We find the structure in the structured objects by evaluating a special kind of features. These features are composed of one or more correlated constituents of the example. Evaluating the feature is equivalent to estimating parameters of the models of the constituents to fit the example.

The simplest feature consists of the model of a single constituent. The parameters of the model of the constituent is the same as the value of the feature:

$$\theta_i = F_i. \tag{3.14}$$

Other features are of the form of reduced representations of parameters of models of highly correlated constituents. The parameters of the models are a smooth function of the the feature:

$$\langle \theta_r \rangle = T_{\mathbf{r}}(F_{\mathbf{r}}), \quad r \in \mathbf{r}, \tag{3.15}$$

where $\langle \theta_r \rangle, r \in \mathbf{r}$ is the vector of parameters of the constituents included in

feature $F_{\mathbf{r}}$. By using the feature $F_{\mathbf{r}}$ instead of parameters $\langle\theta_r\rangle$, we reduce the number of free scalar variables to estimate from the number of scalar variables in $\langle\theta_r\rangle$ to the number of scalar variables in $F_{\mathbf{r}}$. Note that not all values of $\langle\theta_r\rangle$ corresponds to a valid $F_{\mathbf{r}}$, since $F_{\mathbf{r}}$ is a reduced representation of $\langle\theta_r\rangle$. However, we can always find the value for $F_{\mathbf{r}}$ such that $T_{\mathbf{r}}(F_{\mathbf{r}})$ is closest to any given $\langle\theta_r\rangle$.

3.2.1 Evaluating the feature

We evaluate the features in an ordered sequence. Since the reverse mapping from $\mathbf{x}$ to θ is not readily available, the features are evaluated through an optimization of the following form:

$$
\begin{aligned}
F^*_{\mathbf{r}} &= \arg\max_{F_{\mathbf{r}}} P(F_{\mathbf{r}}|\mathbf{x}, F_{\mathbf{r}-1}, F_{\mathbf{r}-2}, \ldots, F_1) && (3.16)\\
&= \arg\max_{F_{\mathbf{r}}} P(\mathbf{x}|F_{\mathbf{r}}, F_{\mathbf{r}-1}, F_{\mathbf{r}-2}, \ldots, F_1) \\
&\quad P(F_{\mathbf{r}}|F_{\mathbf{r}-1}, F_{\mathbf{r}-2}, \ldots, F_1) && (3.17)\\
&= P(\mathbf{x}_{\mathbf{r}}|F_{\mathbf{r}})P(F_{\mathbf{r}}|F_{\mathbf{r}-1}, F_{\mathbf{r}-2}, \ldots, F_1) && (3.18)\\
&= \arg\min_{F_{\mathbf{r}}} \mathrm{Loss}(\mathbf{x}_{\mathbf{r}}, \mathrm{Render}(m_{\mathbf{r}}, T_{\mathbf{r}}(F_{\mathbf{r}}))) \\
&\quad - \rho_{\mathbf{r}} \log P(F_{\mathbf{r}}|F_{\mathbf{r}-1}, F_{\mathbf{r}-2}, \ldots, F_1), && (3.19)
\end{aligned}
$$

where

$$
\mathbf{x}_{\mathbf{r}} = \mathrm{Explain}(\mathbf{x}, F_{\mathbf{r}-1}, F_{\mathbf{r}-2}, \ldots, F_1), \tag{3.20}
$$

in which $\mathrm{Explain}(\mathbf{x}, F_{\mathbf{r}-1}, F_{\mathbf{r}-2}, \ldots, F_1)$ discounts information in $\mathbf{x}$ explained by constituents in features $F_{\mathbf{r}-1}, F_{\mathbf{r}-2}, \ldots, F_1$ that $F_{\mathbf{r}}$ does not include.

$\mathrm{Loss}(\cdot)$ and $\rho_{\mathbf{r}}$ In Equation (3.19), as well as $\mathrm{Explain}(\cdot)$ in Equation (3.20)

need to be optimized for the match between the model $\text{Render}(m_{\mathbf{r}}, T_{\mathbf{r}}(F_{\mathbf{r}}))$ and the underlying $\text{Render}(m^*_{\mathbf{r}}, \theta^*_{\mathbf{r}})$.

3.3 Useful Information for Estimating the Parameters

Given an example $\mathbf{x} \in \mathcal{X}$, the useful information in $\mathbf{x}$ for estimating parameters $\langle \theta_r \rangle, r \in \mathbf{r}$ is the mutual information between $\mathbf{x}$ and the constituents $\bigoplus_{r \in \mathbf{r}} \text{Render}(m^*_r, \theta^*_r)$. Assuming an additive composition operator such that

$$\bigoplus_{r \in \mathbf{r}} \text{Render}(m^*_r, \theta^*_r) \equiv \sum_{r \in \mathbf{r}} \text{Render}(m^*_r, \theta^*_r), \tag{3.21}$$

then

$$\mathbf{x} = \sum_{r \in \mathbf{r}} \text{Render}(m^*_r, \theta^*_r) + \sum_{r \notin \mathbf{r}} \text{Render}(m^*_r, \theta^*_r), \tag{3.22}$$

and thus

$$\begin{aligned}
& I\left(\mathbf{x}; \sum_{r \in \mathbf{r}} \text{Render}(m^*_r, \theta^*_r)\right) \\
&= h(\mathbf{x}) - h\left(\mathbf{x} \middle| \sum_{r \in \mathbf{r}} \text{Render}(m^*_r, \theta^*_r)\right) && (3.23) \\
&= h(\mathbf{x}) - h\left(\sum_{r \in \mathbf{r}} \text{Render}(m^*_r, \theta^*_r) + \sum_{r \notin \mathbf{r}} \text{Render}(m^*_r, \theta^*_r) \right. \\
&\qquad \left. \middle| \sum_{r \in \mathbf{r}} \text{Render}(m^*_r, \theta^*_r)\right) && (3.24) \\
&= h(\mathbf{x}) - h\left(\sum_{r \notin \mathbf{r}} \text{Render}(m^*_r, \theta^*_r) \middle| \sum_{r \in \mathbf{r}} \text{Render}(m^*_r, \theta^*_r)\right). && (3.25)
\end{aligned}$$

In Equation (3.22), $\sum_{r\in\mathbf{r}} \text{Render}(m^*_r, \theta^*_r)$ is the *signal* and $\sum_{r\notin\mathbf{r}} \text{Render}(m^*_r, \theta^*_r)$ is the *noise* for the purpose of estimating $\langle\theta_r\rangle, r \in \mathbf{r}$.

3.3.1 Suppressing the noise using explanation

When estimating $\langle\theta_r\rangle, r \in \mathbf{r}$, we use $\mathbf{x_r} = \text{Explain}(\mathbf{x}, F_{\mathbf{r}-1}, F_{\mathbf{r}-2}, \ldots, F_1)$ instead of $\mathbf{x}$. $\text{Explain}(\mathbf{x}, F_{\mathbf{r}-1}, F_{\mathbf{r}-2}, \ldots, F_1)$ improves the useful information as defined in Equation (3.25) by explaining information in $\mathbf{x}$ using previously evaluated features $F_{\mathbf{r}-1}, F_{\mathbf{r}-2}, \ldots, F_1$, and suppressing the noise term in Equation (3.22). A "perfect" Explain$(\cdot)$ given perfect models $m \equiv m^*$ and perfect parameters $\theta \equiv \theta^*$ is defined as follows:

$$\begin{aligned}
\mathbf{x_r} &= \text{Explain}(\mathbf{x}, F_{\mathbf{r}-1}, F_{\mathbf{r}-2}, \ldots, F_1) && (3.26)\\
&= \sum_{r\in\mathbf{r}} \text{Render}(m^*_r, \theta^*_r) + \sum_{r\notin\mathbf{r}} \text{Render}(m^*_r, \theta^*_r) \\
&\quad - \sum_{r\notin\mathbf{r}} \text{Render}(m_r, \theta_r) && (3.27)\\
&= \sum_{r\in\mathbf{r}} \text{Render}(m^*_r, \theta^*_r) && (3.28)\\
&= \sum_{r\in\mathbf{r}} \text{Render}(m_r, \theta_r). && (3.29)
\end{aligned}$$

That is, the perfect $\mathbf{x_r}$ contains only information relevant to $\langle\theta_r\rangle$. The useful information in this perfect case is

$$\begin{aligned} & I\left(\mathbf{x_r}; \sum_{r\in\mathbf{r}} \text{Render}(m_r^*, \theta_r^*)\right) \\ &= I(\mathbf{x_r}; \mathbf{x_r}) && (3.30) \\ &= h(\mathbf{x_r}) && (3.31) \\ &= h\left(\sum_{r\in\mathbf{r}} \text{Render}(m_r, \theta_r)\right). && (3.32) \end{aligned}$$

In practice, with an imperfect Explain$(\cdot)$, imperfect models m, and inaccurate parameters θ, the explained example with the irrelevant parts discounted would be

$$\begin{aligned} \mathbf{x_r} &= \text{Explain}(\mathbf{x}, F_{\mathbf{r}-1}, F_{\mathbf{r}-2}, \ldots, F_1) && (3.33) \\ &= \sum_{r\in\mathbf{r}} \text{Render}(m_r^*, \theta_r^*) + \sum_{r\notin\mathbf{r}} \text{Render}(m_r^*, \theta_r^*) \\ &\quad - \text{s}\left(\sum_{r\notin\mathbf{r}} \text{Render}(m_r, \theta_r)\right), && (3.34) \end{aligned}$$

where s$(\cdot)$ is a function that decides how Explain$(\cdot)$ suppresses information from the imperfect rendition $\sum_{r\notin\mathbf{r}} \text{Render}(m_r, \theta_r)$, and is dependent on how good $\sum_{r\notin\mathbf{r}} \text{Render}(m_r, \theta_r)$ is in approximating $\sum_{r\notin\mathbf{r}} \text{Render}(m_r^*, \theta_r^*)$.

The useful information in $\mathbf{x_r}$ is thus

$$
\begin{aligned}
& I\left(\mathbf{x_r}; \sum_{r\in\mathbf{r}} \text{Render}(m_r^*, \theta_r^*)\right) \\
&= h(\mathbf{x_r}) - h\left(\mathbf{x_r} \left| \sum_{r\in\mathbf{r}} \text{Render}(m_r^*, \theta_r^*)\right.\right) && (3.35) \\
&= h(\mathbf{x_r}) - h\left(\sum_{r\in\mathbf{r}} \text{Render}(m_r^*, \theta_r^*) + \sum_{r\notin\mathbf{r}} \text{Render}(m_r^*, \theta_r^*)\right. \\
&\quad \left. - \text{s}\left(\sum_{r\notin\mathbf{r}} \text{Render}(m_r, \theta_r)\right) \left| \sum_{r\in\mathbf{r}} \text{Render}(m_r^*, \theta_r^*)\right.\right) && (3.36) \\
&= h(\mathbf{x_r}) - h\left(\sum_{r\notin\mathbf{r}} \text{Render}(m_r^*, \theta_r^*) - \text{s}\left(\sum_{r\notin\mathbf{r}} \text{Render}(m_r, \theta_r)\right)\right. \\
&\quad \left. \left| \sum_{r\in\mathbf{r}} \text{Render}(m_r^*, \theta_r^*)\right.\right). && (3.37)
\end{aligned}
$$

Equation (3.37) equals $h(\mathbf{x_r})$ when

$$
\text{s}\left(\sum_{r\notin\mathbf{r}} \text{Render}(m_r, \theta_r)\right) = \sum_{r\notin\mathbf{r}} \text{Render}(m_r^*, \theta_r^*). \qquad (3.38)
$$

3.4 Quality of Features

3.4.1 Entropy of the feature

The value of a feature $F_\mathbf{r}$ is an estimation to the parameters of model $m_\mathbf{r}$, such that the rendition $\sum_{r\in\mathbf{r}} \text{Render}(m_\mathbf{r}, T_\mathbf{r}(F_\mathbf{r}))$ best approximate the rendition of the underlying model $\sum_{r\in\mathbf{r}} \text{Render}(m_r^*, \theta_r^*)$ that generates it. Therefore, $F_\mathbf{r}$ is a function of $\sum_{r\in\mathbf{r}} \text{Render}(m_r^*, \theta_r^*)$. For any feature $F_\mathbf{r}$, whether $F_\mathbf{r}$ is a single constituent feature or a feature composed of the

reduced representation of more than one constituents, $F_{\mathbf{r}}$ is conditionally independent of $\mathbf{x_r}$ given $\sum_{r\in\mathbf{r}} \text{Render}(m_r^*, \theta_r^*)$. That is,

$$\begin{aligned} & p\left(\mathbf{x_r}, \sum_{r\in\mathbf{r}} \text{Render}(m_r^*, \theta_r^*), F_{\mathbf{r}}\right) \\ = \;& p(\mathbf{x_r}) p\left(\sum_{r\in\mathbf{r}} \text{Render}(m_r^*, \theta_r^*) \middle| \mathbf{x_r}\right) \\ & p\left(F_{\mathbf{r}} \middle| \sum_{r\in\mathbf{r}} \text{Render}(m_r^*, \theta_r^*)\right). \end{aligned} \tag{3.39}$$

According to the data-processing inequality, we have

$$I(\mathbf{x_r}; F_{\mathbf{r}}) \leq I\left(\mathbf{x_r}; \sum_{r\in\mathbf{r}} \text{Render}(m_r^*, \theta_r^*)\right). \tag{3.40}$$

Therefore, the conditional entropy $h(F_{\mathbf{r}}|\mathbf{x_r})$ is lower bounded as:

$$h(F_{\mathbf{r}}|\mathbf{x_r}) = h(F_{\mathbf{r}}) - I(\mathbf{x_r}; F_{\mathbf{r}}) \tag{3.41}$$

$$\geq h(F_{\mathbf{r}}) - I\left(\mathbf{x_r}; \sum_{r\in\mathbf{r}} \text{Render}(m_r^*, \theta_r^*)\right). \tag{3.42}$$

Given the same $\mathbf{x_r}$, the conditional entropy $h(F_{\mathbf{r}}|\mathbf{x_r})$ is largely influenced by the entropy $h(F_{\mathbf{r}})$ itself. Therefore, given the same $\mathbf{x_r}$, evaluating the feature with reduced representation of multiple constituents, which has a lower $h(F_{\mathbf{r}})$ because of the reduced representation, results in a lower variance than evaluating each of the constituents with its full representation separately. Furthermore, the more restricted $F_{\mathbf{r}}$ is for the same constituents, the lower the variance is.

3.4.2 Accuracy of the feature

Given $\theta_{\mathbf{r}}$ and its reduced representation $F_{\mathbf{r}}$, the set of possible renditions using $\text{Render}(m_{\mathbf{r}}, T_{\mathbf{r}}(F_{\mathbf{r}}))$ is a subset of the set of possible renditions using $\text{Render}(m_{\mathbf{r}}, \theta_{\mathbf{r}})$. Therefore, $\text{Render}(m_{\mathbf{r}}, \theta_{\mathbf{r}})$ is a more expressive model and can match the underlying $\text{Render}(m^*_{\mathbf{r}}, \theta^*_{\mathbf{r}})$ more accurately than $\text{Render}(m_{\mathbf{r}}, T_{\mathbf{r}}(F_{\mathbf{r}}))$ can.

3.4.3 Sequence of feature evaluations

The choice of suitable features is a tradeoff between the variance of the feature and the accuracy of the feature. When $\sum_{r \notin \mathbf{r}} \text{Render}(m_r, \theta_r)$ poorly approximates $\sum_{r \notin \mathbf{r}} \text{Render}(m^*_r, \theta^*_r)$ in Equation (3.37), $\text{Explain}(\cdot)$ suppresses the noise in information badly, and the useful information $I\left(\mathbf{x}_{\mathbf{r}}; \sum_{r \in \mathbf{r}} \text{Render}(m^*_r, \theta^*_r)\right)$ is low. In this case, in order to achieve a low conditional entropy $h(F_{\mathbf{r}}|\mathbf{x}_{\mathbf{r}})$ in Equation (3.42), it is desirable to use a more restrictive $F_{\mathbf{r}}$, or a reduced representation feature that includes more constituents, such that the the entropy $h(F_{\mathbf{r}})$ is low.

After we have more accurate $\sum_{r \notin \mathbf{r}} \text{Render}(m_r, \theta_r)$ such that $\text{Explain}(\cdot)$ suppresses noise well for $\mathbf{x}_{\mathbf{r}}$ and that the useful information $I\left(\mathbf{x}_{\mathbf{r}}; \sum_{r \in \mathbf{r}} \text{Render}(m^*_r, \theta^*_r)\right)$ is high enough, we can afford to use a more expressive feature to compute more accurate estimations for the constituents.

Chapter 4

Empirical Results

4.1 Overview

Our approach improves the classification by concentrating information present in the examples, making the classification problem easier. By focusing on portions of the example with high mutual information with the class label, we improve the signal to noise ratio in the example, making the pattern easier to pick up by the classifier. However, the act of concentrating information also throws away information at the same time, and could possibly hurt the classification in certain cases.

The success of our approach depends on a few factors. We expect our approach to help the most on classification problems that the signal is too dilute in the example for the statistical classifier alone to pick up, but that it can be improved by focusing on the right place, provided that the prior knowledge is reasonably good at locating the right place to look. Therefore, we test our approach with the task of classifying pairs of Chinese characters

that are likely to have such property. Specifically, we test our approach on pairs of characters that ordinary statistical classifiers do not do well on, and that humans do better on. Native Chinese readers with prior domain knowledge about the characters generally do a better job at finding the right place to look when recognizing Chinese characters. We are not interested in classification tasks that cannot be improved by concentrating information or that purely statistical classifiers already do well on.

The state of the art in the literature of multiclass offline handwritten Chinese character recognition uses the multiclass linear discriminant analysis on the weighted direction code histogram (WDH) features. This state of the art is described in Section 4.2. Section 4.3 establishes a human baseline for comparison. Section 4.4 provides a pairwise classification approach using the support vector machine on the WDH features, which is the state of the art for machine learning. We show that this significantly outperforms the multiclass LDA system of Section 4.2.

The pairs of characters we aim to improve on using our approach are those that the accuracies of the support vector machine are lower than 98% of the human baseline.

4.1.1 Empirical questions that we address

We address empirically the following questions about our approach:

Does prior knowledge help where predicted? (Section 4.5)

The prior knowledge in our approach is a simplified and imperfect stand-in for additional training examples. In Section 4.5 we compare our approach

against the RBF SVM. We expect our approach to do better on most pairs that there is a large gap in accuracies between the SVM accuracy and the human accuracy, which likely indicates the classification can be improved by focusing information. Our approach may not work in some cases. It may fail to improve the classification if the quality of the prior knowledge is not good enough, such that the information loss by the process of concentrating it outweighs the effect of the concentrated information. It may also not be able to improve if the statistical classifier is flexible enough to pick up useful information that we do not concentrated on, and that there is enough training examples such that the flexible statistical classifier does not overfit.

Does our approach concentrate classification information? (Section 4.6)

Our approach makes the learning problem easier by concentrating information in the examples, making the pattern easier to pick up by the classifier. This being the case, the classifier should learn faster with fewer training examples, producing a steeper learning curve than a purely statistical approach. We think that with reasonably good prior knowledge our approach concentrates information in all cases, and that the learning curve of our approach will be steeper even in cases that we do not improve upon in terms of the final classification accuracy in the end.

What is the value of the prior knowledge? (Section 4.7)

Since the prior knowledge in our approach is a simplified and imperfect stand-in for additional training examples, we want to measure how much

the prior knowledge is worth in terms of number of additional class-labeled training examples. The value of the prior knowledge depends on its quality, as well as the difficulty of the task. The better the prior knowledge is in concentrating the right information, the more additional training examples the knowledge is worth.

4.1.2 Offline handwritten Chinese character database

There are a few commonly used offline handwritten Chinese databases. One important database is the ETL9B [24] data set. It contains Chinese characters taken from the Japanese kanji. Much important research is based on this data set, including that from Sun et al. [29, 28, 27], Kato et al. [13], Liu and Ding [20], etc. However, examples in this data set tend to be carefully made and artificially clean that they do not reflect well the variability seen in real world handwritten documents.

We test our approach on the HITPU [26, 25] offline handwritten Chinese character database. This is a newer database, and the examples display a much larger range of variability in this database than in the ETL9B data set. Lim et al. [17, 18] used this database in their work. The database contains roughly 751,000 handwritten Chinese characters in 3,755 classes written by 200 different writers. Figure 2.1 contains samples of characters in this database. Each character in the database is a binary image. The widths and heights of the character images are not fixed, and range from 40 pixels to 192 pixels on each side. In our experiments, we zoom and center the examples to 100-pixel by 100-pixel images.

4.2 Multiclass LDA Baseline

One of the most successful set of features used in offline handwritten Chinese character classification is the weighted direction code histogram (WDH) feature [14] [4] as described in Section 2.7.2. We generate the WDH features for all 3,775 classes of examples, and train multiclass Fisher's discriminant models using the FDA module in [34] to do 5-fold cross-validation.

After obtaining the confusion matrix for the 3,775 classes, we identify pairs of character classes that are most confusing, and compute their pairwise classification accuracies. The pairs of character classes with the lowest pairwise classification accuracies are listed in Table 4.1. These pairs of characters are candidate classification tasks for our investigation.

4.3 Human Baseline

It is not possible for every classification problem to achieve a 100% accuracy. In order to determine a reasonable target accuracy for classifying each pair of characters in Table 4.1, as well as the possible gap for improvement over statistical classifiers, we use crowdsourcing and establish a human baseline accuracy for classifying the difficult pairs of characters.

We build an web-based online system for this purpose. The user is asked to participate in an online test for recognizing handwritten Chinese characters. In each question during the test, the image of a random character is presented to the user, and the user is asked to choose among the options of two similar character classes, "Obviously none of the above," "Really can't tell," and "Skip." Upon recording the choice, the next character image is

Pair	竟 竞	晴 晴	日 曰
LDA Accuracy	75.61%	78.05%	78.92%
Pair	乌 乌	孟 盂	已 已
LDA Accuracy	79.76%	81.13%	84.60%
Pair	呜 呜	扶 抉	酒 洒
LDA Accuracy	84.88%	86.10%	86.10%
Pair	已 已	大 犬	白 自
LDA Accuracy	86.31%	86.34%	87.26%
Pair	侯 候	伸 仲	拔 拨
LDA Accuracy	87.32%	87.50%	87.56%
Pair	戒 戎	木 朮	免 兔
LDA Accuracy	87.81%	88.05%	88.05%
Pair	狠 狼	澜 谰	便 使
LDA Accuracy	88.29%	88.29%	88.54%
Pair	干 千	菜 莱	潭 谭
LDA Accuracy	89.02%	89.27%	89.51%
Pair	壕 嚎	季 李	
LDA Accuracy	89.51%	90.00%	

Table 4.1: Pairwise classification accuracies for the hardest pairs of character classes according to multiclass Fisher's discriminant analysis using the WDH features.

presented. In order to ensure recognition within reasonable time and prevent the user from studying the character image too hard, the image disappears after three seconds. This time limit does not cause much problem for native Chinese readers, which is the case for all of our users. The user is asked to choose "Obviously none of the above" only when the character presented is obviously not from the two options presented, and "Really can't tell" only when the user really cannot tell the character class. The user can choose to skip the question if he or she is distracted when the image is shown. "Obviously none of the above" and "Really can't tell" answers are treated as half correct, and skipped questions are not included.

There is a 1 in 20 chance that a "trick question" is given. During the trick question, the character image presented is different from both classes available in the options. The user is expected to choose "Obviously none of the above" for the trick question. Trick questions are meant to identify and exclude malicious responses from the result.

We use browser cookies to keep track of individual users. During the survey, 73 individual users responded, with a total of 11,808 responses. We only use responses from users who answered at least 5 trick questions, and their answers to trick questions must be "Obviously none of the above" at least 97% of the time. 1,378 responses from 11 users are excluded from the results. Of the 10,430 remaining responses, 9,937 are not trick questions. There are 26 pairs of similar characters in the survey, and each pair received at least 320 valid responses. Table 4.2 shows the resulting human recognition accuracy and the 95% confidence interval for Bernoulli trials.

Pair	竟 竞	晴 晴	日 曰
Human Accuracy	$94.63 \pm 2.26\%$	$92.90 \pm 2.58\%$	$81.71 \pm 3.95\%$
Pair	鸟 乌	孟 盂	己 已
Human Accuracy	$91.23 \pm 2.95\%$	$94.32 \pm 2.42\%$	$92.12 \pm 2.21\%$
Pair	呜 呜	扶 抉	酒 洒
Human Accuracy	$91.64 \pm 2.84\%$	$97.66 \pm 1.56\%$	$94.97 \pm 2.20\%$
Pair	巳 已	大 犬	白 自
Human Accuracy	$87.87 \pm 2.68\%$	$99.71 \pm 0.57\%$	$98.97 \pm 1.00\%$
Pair	侯 候	伸 仲	拔 拨
Human Accuracy	$94.69 \pm 2.30\%$	$99.74 \pm 0.51\%$	$82.61 \pm 3.87\%$
Pair	戒 戎	木 朮	免 兔
Human Accuracy	$99.44 \pm 0.78\%$	$97.34 \pm 1.77\%$	$97.74 \pm 1.50\%$
Pair	狠 狼	澜 谰	便 使
Human Accuracy	$99.22 \pm 0.88\%$	$93.60 \pm 2.51\%$	$98.82 \pm 1.09\%$
Pair	干 千	菜 莱	潭 谭
Human Accuracy	$83.57 \pm 3.85\%$	$88.93 \pm 3.18\%$	$90.93 \pm 2.95\%$
Pair	壕 嚎	季 李	
Human Accuracy	$98.69 \pm 1.14\%$	$99.27 \pm 0.90\%$	

Table 4.2: Human recognition accuracy for the hardest pairs of characters and the 95% confidence interval.

4.4 Support Vector Machine Baseline

In order to know how well a purely statistical pairwise classifier performs on the hardest pairs of characters as determined in Section 4.2, we obtain baseline accuracies using the support vector machine with RBF kernels trained on standard WDH features on whole character images as described in Section 2.7.2. We generate standard WDH features for the pairs of characters in Table 4.1, and do 5-fold cross validation using the support vector machine with an RBF kernel. Best kernel parameters are chosen for each fold using a separate cross validation. The SVM implementation that we use is LIBSVM [2]. Table 4.3 shows cross validations accuracies using the RBF SVM. The "relative accuracy" for SVM in Table 4.3 is defined as the SVM cross validation accuracy divided by the human accuracy. A relative accuracy higher than 100% means the SVM outperforms human in that case.

4.5 Experiment 1: Does prior knowledge help where predicted?

Overview

Our approach improves the classification by using the imperfect prior knowledge to concentrate information in the examples. However, it may not always help the classification, as we also throw away information at the same time we concentrate it. We want to empirically determine if and when the prior knowledge helps. We think our approach works better if there is a larger gap between the accuracies of the purely statistical approach and human,

Pair	RBF SVM Accuracy	Accuracy Relative to Human
晴 晴	84.15%	90.57%
狠 狼	90.98%	91.70%
壕 嚎	91.71%	92.93%
季 李	93.17%	93.86%
呜 呜	86.59%	94.49%
木 朮	92.20%	94.72%
便 使	94.15%	95.27%
扶 抉	93.66%	95.90%
伸 仲	95.83%	96.08%
竟 竞	90.98%	96.14%
大 犬	97.32%	97.60%
兔 免	95.61%	97.82%
鸟 乌	89.27%	97.85%
酒 洒	93.41%	98.36%
候 侯	93.17%	98.39%
澜 谰	92.20%	98.50%
白 自	97.55%	98.57%
孟 盂	93.63%	99.27%
戒 戎	99.51%	100.07%
己 已	92.91%	100.86%
潭 谭	92.68%	101.92%
巳 已	89.73%	102.12%
日 曰	83.58%	102.29%
菜 莱	91.22%	102.58%
干 千	89.02%	106.52%
拔 拨	94.15%	113.97%

Table 4.3: Accuracies for the RBF SVM. The "relative" accuracies are relative to the human accuracy. Relative accuracies higher than 100% are cases that the SVM outperforms human.

indicating possible room of improvement by focusing information. Our approach may not work if the quality of the prior knowledge is not good enough in concentrating the useful information, or if there is enough examples for a flexible enough statistical classifier to pick up patterns that we do not concentrate on.

Experimental design

We test our approach on the pairs of characters that the RBF SVM accuracy relative to the human accuracy is lower than 100%, and aim to improve on most of the pairs that the relative SVM accuracy to human is lower than 98%, which indicates possible room of improvement by focusing information.

For each pairwise classification task, we use 40 structurally labeled training examples for our approach, with 20 in each class. These structurally labeled training examples are used to learn and optimize the sequence of match operations that finds structures in unknown test examples. Furthermore, these training examples are also used in searching for the best parameters for generating our variant of gradient features described in Section 2.7.3 based on the cross validation posterior probability introduced in Section 2.8.

After our model for finding structures is learned, we apply the model to the remaining examples excluding the 40 structurally labeled training examples. We then do 5-fold cross validation on the remaining examples using RBF SVM with the gradient features for the located target region in the character images. The SVM kernel parameters are optimized using a

separate cross validation in each fold based on the cross validation posterior probability.

We compare our cross validation accuracy relative to human accuracy against the SVM baseline obtained in Section 4.4, and compute the *relative improvement* as the difference between our relative accuracy and the relative accuracy of the RBF SVM divided by the relative error of the RBF SVM:

$$RelativeImprovement = \frac{RelativeAccuracy_{Our} - RelativeAccuracy_{SVM}}{1 - RelativeAccuracy_{SVM}}. \tag{4.1}$$

We think our approach should work better when there is a larger gap between the accuracies of the purely statistical approach and human. This is tested by performing the statistical significance test for the nonparametric Kendall's tau rank correlation between SVM accuracy relative to the human accuracy and the relative improvement. The null hypothesis is that the SVM relative accuracy and the relative improvement are not negatively correlated. The alternative hypothesis is that the SVM relative accuracy and the relative improvement are negatively correlated.

Empirical results

Table 4.4 shows the cross validation accuracies and the cross validation accuracies relative to human accuracies using our approach, as well as the relative improvement defined in Equation (4.1). The p-value for the alternative hypothesis that the SVM relative accuracy and the relative improvement are negatively correlated is 0.006865.

Pair	RBF SVM	RBF SVM Rel.	Our Approach	Our Approach Rel.	Rel. Improvement
晴 晴	84.15%	90.57%	82.97%	89.31%	-13.36%
狠 狠	90.98%	91.70%	98.38%	99.15%	+89.76%
壕 嚎	91.71%	92.93%	98.11%	99.41%	+91.65%
季 季	93.17%	93.86%	95.95%	96.66%	+45.60%
鸣 呜	86.59%	94.49%	87.03%	94.97%	+8.87%
木 朩	92.20%	94.72%	95.68%	98.29%	+67.61%
便 使	94.15%	95.27%	98.38%	99.55%	+90.49%
扶 抉	93.66%	95.90%	97.03%	99.35%	+84.15%
伸 仲	95.83%	96.08%	95.65%	95.90%	-4.59%
竟 竞	90.98%	96.14%	91.62%	96.82%	+17.62%
大 犬	97.32%	97.60%	95.14%	95.52%	-90.80%
兔 免	95.61%	97.82%	95.68%	97.86%	+3.21%
鸟 乌	89.27%	97.85%	87.03%	95.40%	-113.95%
酒 洒	93.41%	98.36%	85.95%	90.50%	-479.27%
候 侯	93.17%	98.39%	88.38%	93.34%	-313.66%
澜 谰	92.20%	98.50%	95.95%	102.51%	+267.33%
白 自	97.55%	98.57%	95.92%	96.92%	-115.38%
孟 孟	93.63%	99.27%	92.39%	97.95%	-180.82%

Table 4.4: Accuracies for the RBF SVM and accuracies using our approach. The "relative" accuracies are relative to the human accuracy. Relative accuracies higher than 100% are cases that the specific method outperforms human.

Interpretation

Among the 13 pairs with the RBF SVM accuracies lower than 98% of their human accuracies, we outperform the SVM on 9 pairs. Furthermore, we outperform the SVM on 8 of the 10 pairs of which the relative SVM accuracies are lower than 97%. Among them, we outperform the SVM by more than 40% on 6 pairs. In these cases the prior knowledge does help the classification task.

However, as the gap between the accuracies of the SVM and the human gets smaller, there is less possible room for improvement by concentrating information, and we do not improve over the SVM as often.

The p-value for the alternative hypothesis that the SVM relative accu-

racy and the relative improvement are negatively correlated is 0.006865. Any p-value below 0.05 is considered significant. Therefore, our approach does work better when there is a larger gap between the SVM and the human accuracies.

Other than cases with insufficient room of improvement by concentrating information, which can be seen by looking at the relative SVM accuracy to human accuracy, we do not improve over the purely statistical approach when the quality of prior knowledge itself is not good enough at localizing the important region of the example. This includes the 睛 vs. 晴 pair, in which one of the short horizontal strokes in the left box radical in 睛 is inadequately found to be consistent to the short horizontal stroke in the left box radical in 晴, and as a result the target region does not always include all of the important information in some examples after these two short horizontal strokes are masked out. Another such case is the 大 vs. 犬 pair, in which we treat the long stroke curved to the left at a straight stroke. This model mismatch prevents us from properly masking out irrelevant information, and hinders us in accurately locating the target region.

4.6 Experiment 2: Does our approach concentrate classification information?

4.6.1 Experiment 2A: Learning curves

Overview

We claim that our approach concentrates information in the examples. More concentrated information should result in an easier learning problem. This in turn should produce a steeper learning curve, which starts at the same accuracy but initially improves much faster with additional training examples. We look at the learning curves of our approach, and compare it against the SVM baseline in this section.

Experimental design

In order to better understand how the prior knowledge in our approach affects the learning rate, we look at the learning curves of characters with regard to the number of class-labeled training examples.

For our approach, after learning the best structural model using the 40 structurally labeled training examples, we repeatedly sample sets of training examples of sizes 2, 10, 20, 40, 80, 160, and 320 from the class-labeled examples excluding the 40 structurally labeled examples. The remaining unused examples form the hold-out set for testing. SVM kernel parameters are found by performing cross validation on the sampled training set using the cross validation posterior probability except for the 2-example training set, of which the parameters are take from the 40 structurally labeled train-

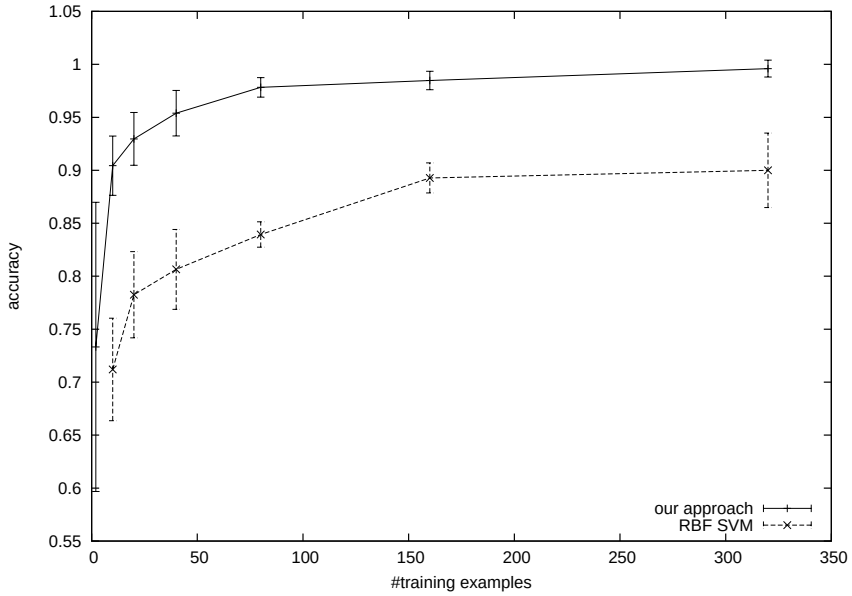
ing examples. An RBF SVM is then trained using the gradient features for the target regions of the sampled training set, and tested on the hold-out set. We plot the test accuracy on the hold-out set versus the number of class-labeled training examples as the learning curve for our approach.

For the learning curve for the RBF SVM, we repeatedly sample sets of training examples of sizes 10, 20, 40, 80, 160, and 320 from the whole set of available examples. SVM kernel parameters are found using cross validation on the sampled training set. The RBF SVM is trained using the standard WDH features for the whole character images of the sampled training set, and tested on the hold-out set. We plot the test accuracy on the hold-out set versus the number of class-labeled training examples as the learning curve for the SVM.

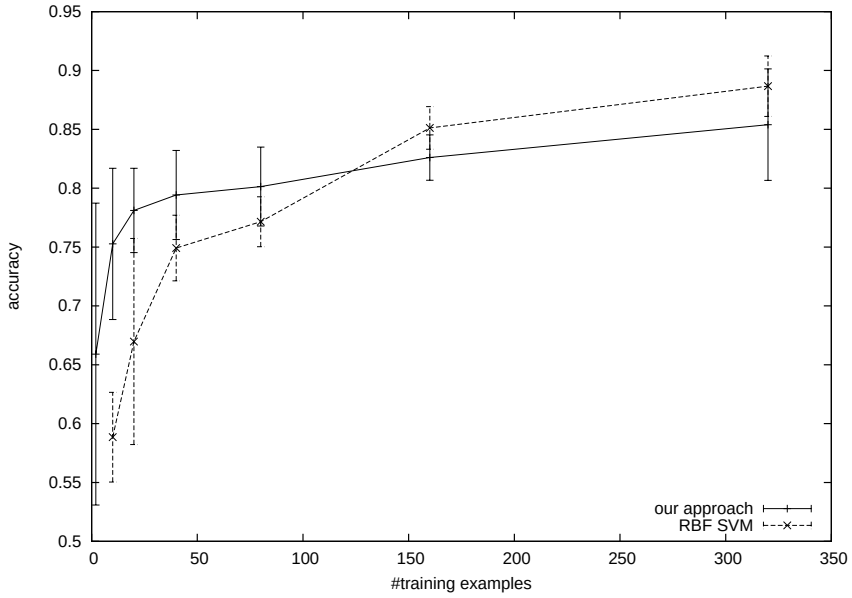
Empirical results

Figure 4.1 shows the learning curves for two pairs of characters whose SVM accuracies are low relative to their human accuracies. Figure 4.2 shows the learning curves for two pairs of characters whose SVM accuracies are close to their human accuracies. The error bars are standard deviations of the accuracies.

Figure 4.1a shows the learning curves for the pair 壕 vs. 嚎, and Figure 4.2a shows the learning curves for the pair 澜 vs. 谰, which our method significantly outperforms the SVM on both pairs. Figure 4.1b shows the learning curves for the pair 睛 vs. 晴, and Figure 4.2b shows the learning curves for the pair 大 vs. 犬, which we perform worse than the SVM in the end.

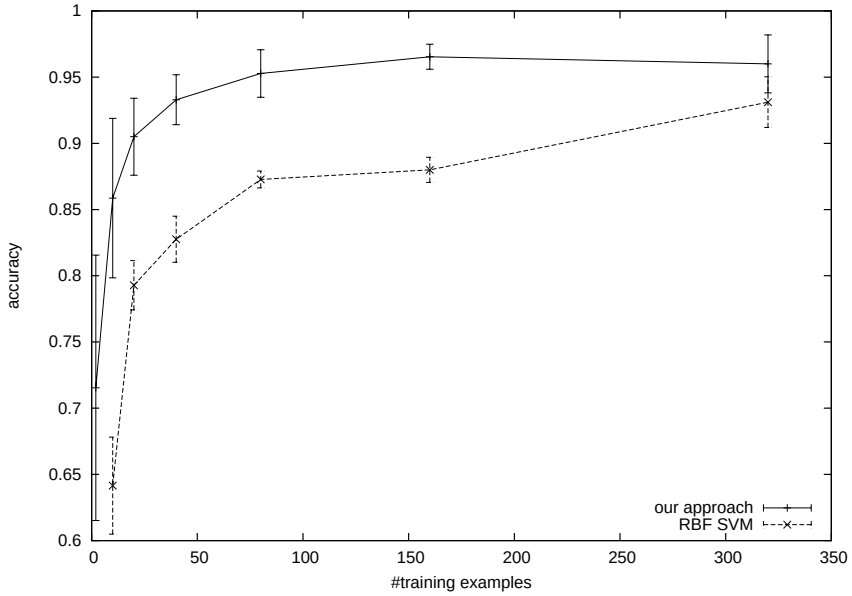


(a) Learning curve for the pair 墜 vs. 墜

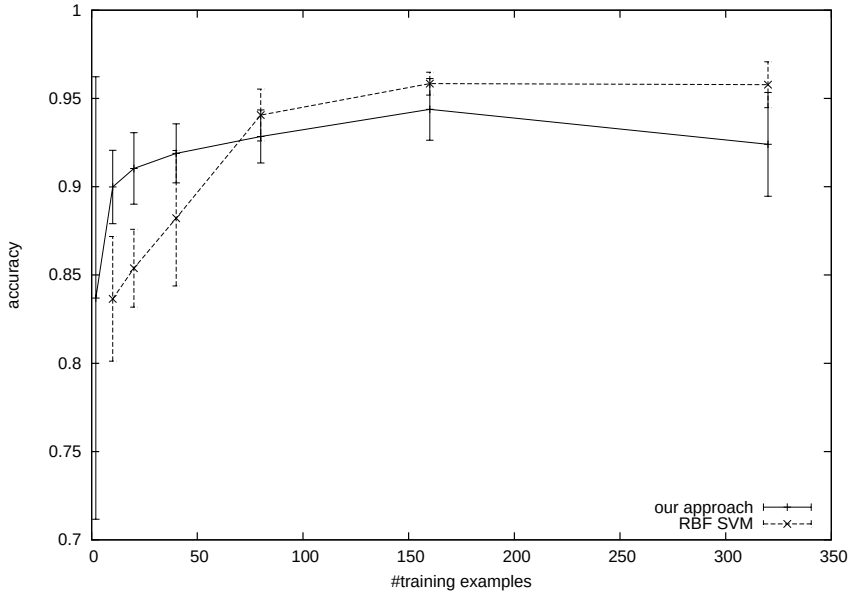


(b) Learning curve for the pair 晴 vs. 晴

Figure 4.1: Learning curves for our method and learning curves for the RBF SVM on pairs of characters of which the SVM accuracies are relatively low compared to the human accuracies. The errorbars are standard deviations of the accuracies.



(a) Learning curve for the pair 涧 vs. 涸



(b) Learning curve for the pair 大 vs. 犬

Figure 4.2: Learning curves for our method and learning curves for the RBF SVM on pairs of characters of which the SVM accuracies are close to the human accuracies. The errorbars are standard deviations of the accuracies.

Interpretation

As can be seen in Figure 4.1 and Figure 4.2, our approach always grows faster initially and significantly outperforms the standard RBF SVM when the size of the training set is small even for cases that we lose to the SVM in the end. However, if there are enough training examples, a purely statistical classifier with enough expressivity such as the RBF SVM may be capable of eliminating any advantage that the possibly inaccurate prior knowledge carries.

4.6.2 Experiment 2B: Steepness of the learning curves

Overview

By looking at the learning curves in Section 4.6.1, it can be seen that the improvement in accuracy with additional training examples for our approach is always faster than the SVM when the size of the training set is small. In order to better study the steepness of the learning curves, we fit a model to the learning curves in this section. We think, through concentrating information in the examples, our approach has a steeper learning curve than a purely statistical approach.

Experimental design

To quantify the steepness of the learning curves and have a better comparison between our approach and the SVM, we fit a simple model to the learning curves. Assuming a 50% accuracy at 0 training example and an exponentially decaying growth rate of accuracy with additional training examples,

we fit the following model to the learning curves obtained in Section 4.6.1:

$$y = -(0.5 - \alpha)e^{-\lambda x} + (1 - \alpha), \tag{4.2}$$

where x is the number of training examples and y is the testing accuracy. $y = 1 - \alpha$ is the asymptote of Equation (4.2), and λ is the decay constant of the growth rate of the accuracy. A larger λ indicates a steeper learning curve. Figure 4.3 and Figure 4.4 show the fit model to the learning curves in Figure 4.1 and Figure 4.2.

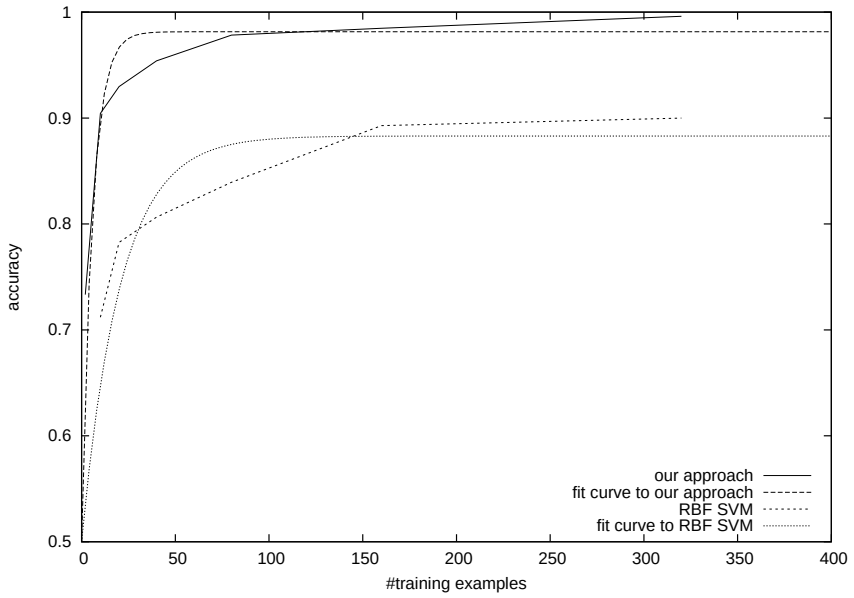
This model for the learning curves is not perfect. It does not always achieve a 100% accuracy even with infinite training. However, we think this simple model is accurate enough for the range we care about, and suitable for our purpose.

Empirical results

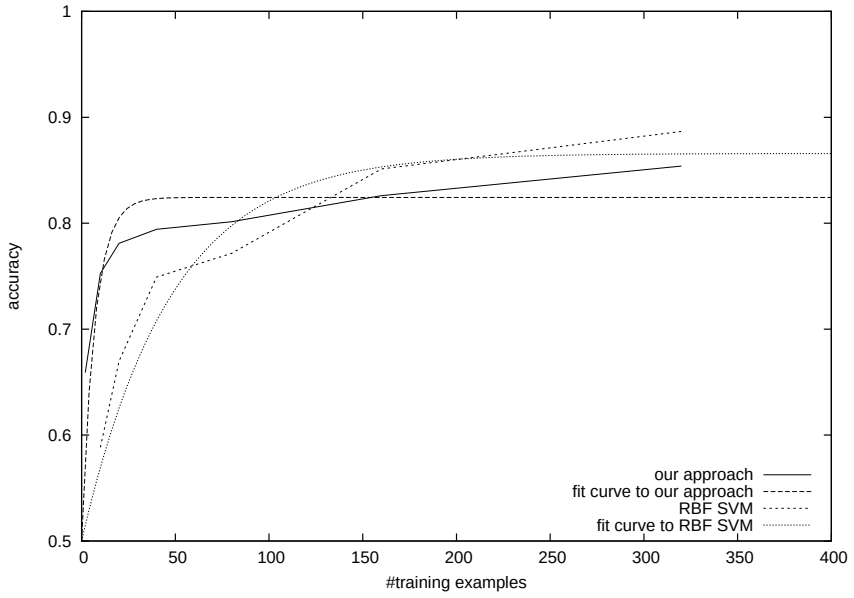
Table 4.5 lists the asymptotes $y = 1 - \alpha$ and the decay constants of the accuracy growth rates λ of the models fit to the learning curves of the RBF SVM and our approach. The mean decay constant of the accuracy growth rate is 0.0432 with a standard deviation of 0.0208 for the RBF SVM, and 0.2058 with a standard deviation of 0.1417 for our approach.

Interpretation

As can be seen in Table 4.5, our approach produces a steeper learning curve for all pairs of characters in terms of the decay constant of the accuracy growth rate λ, even for the pairs that we fail to improve in the end. This

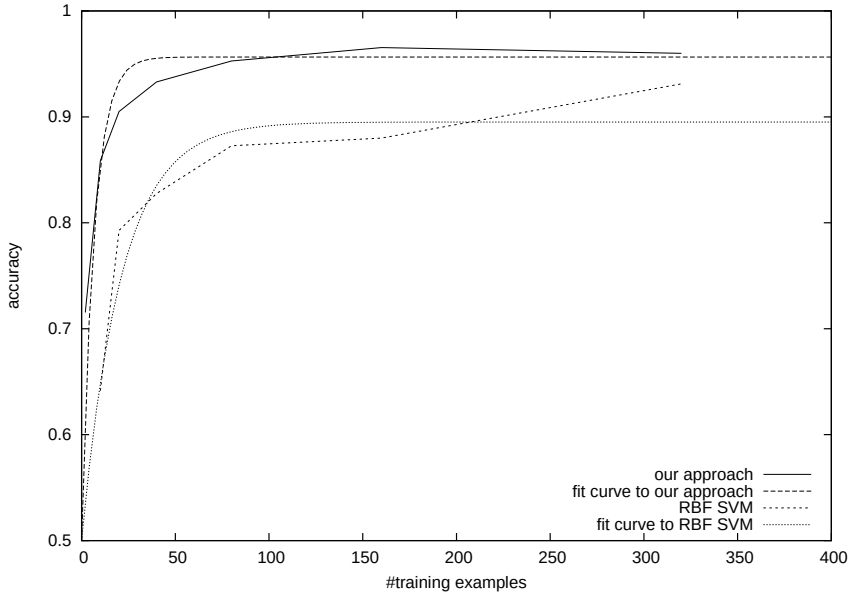


(a) Fit model for the learning curve of the pair 壕 vs. 嚎

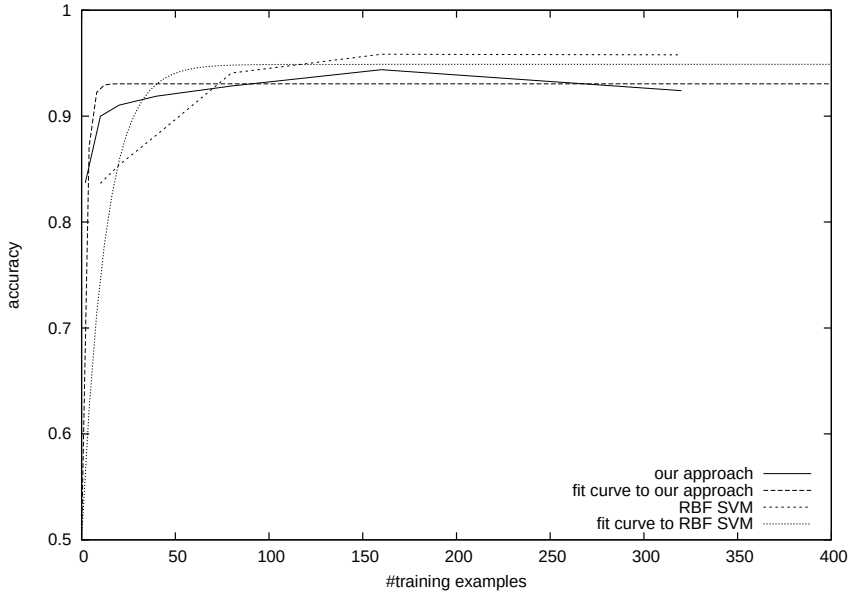


(b) Fig model for the learning curve of the pair 晴 vs. 晴

Figure 4.3: Models fit to the learning curves for our method and for the RBF SVM on pairs of characters of which the SVM accuracies are relatively low compared to the human accuracies.



(a) Fit model for the learning curve of the pair 涧 vs. 谰



(b) Fit model for the learning curve of the pair 大 vs. 犬

Figure 4.4: Models fit to the learning curves for our method and for the RBF SVM on pairs of characters of which the SVM accuracies are close to the human accuracies.

Pair	$1-\alpha$, RBF SVM	λ, RBF SVM	$1-\alpha$, Our Approach	λ, Our Approach	Our $\lambda-$ SVM λ
睛 晴	0.8658	0.0210	0.8243	0.1415	0.1205
狠 狼	0.9013	0.0224	0.9787	0.1619	0.1395
壕 嚎	0.8830	0.0485	0.9814	0.1740	0.1255
李 李	0.9219	0.0618	0.9515	0.0944	0.0326
鸣 呜	0.8519	0.0219	0.8542	0.2211	0.1992
木 朮	0.8826	0.0967	0.9548	0.5756	0.4789
便 使	0.9240	0.0419	0.9677	0.1214	0.0795
扶 抉	0.9253	0.0496	0.9578	0.3606	0.3110
伸 仲	0.9519	0.0552	0.9471	0.1692	0.1140
竟 竞	0.8669	0.0176	0.8888	0.1510	0.1334
大 犬	0.9488	0.0798	0.9305	0.4903	0.4105
兔 兎	0.9362	0.0470	0.9419	0.3020	0.2550
鸟 乌	0.8607	0.0234	0.8703	0.2762	0.2528
酒 洒	0.9245	0.0342	0.8350	0.0349	0.0007
候 侯	0.8935	0.0351	0.8700	0.0899	0.0548
澜 谰	0.8951	0.0471	0.9565	0.1488	0.1017
白 自	0.9606	0.0509	0.9521	0.1419	0.0910
孟 孟	0.9055	0.0239	0.9066	0.0503	0.0264
mean		0.0432		0.2058	0.1626
standard deviation		0.0208		0.1417	

Table 4.5: Asymptotes $y = 1 - \alpha$ and the decay constants of the accuracy growth rates λ of the models fit to the learning curves of the RBF SVM and our approach. A larger λ indicates a steeper learning curve.

confirms our claim that our approach does indeed concentrate information, making the learning problem easier.

4.7 Experiment 3: What is the value of the prior knowledge?

4.7.1 The worth of the prior knowledge in the number of class-labeled training examples

Overview

The prior knowledge in our approach is a simplified and imperfect stand-in for additional training examples. We want to measure how much the prior knowledge is worth in terms of the number of additional examples. The

worth of the prior knowledge indicates the quality of the prior knowledge, as well as aspects of the difficulty of the classification problem.

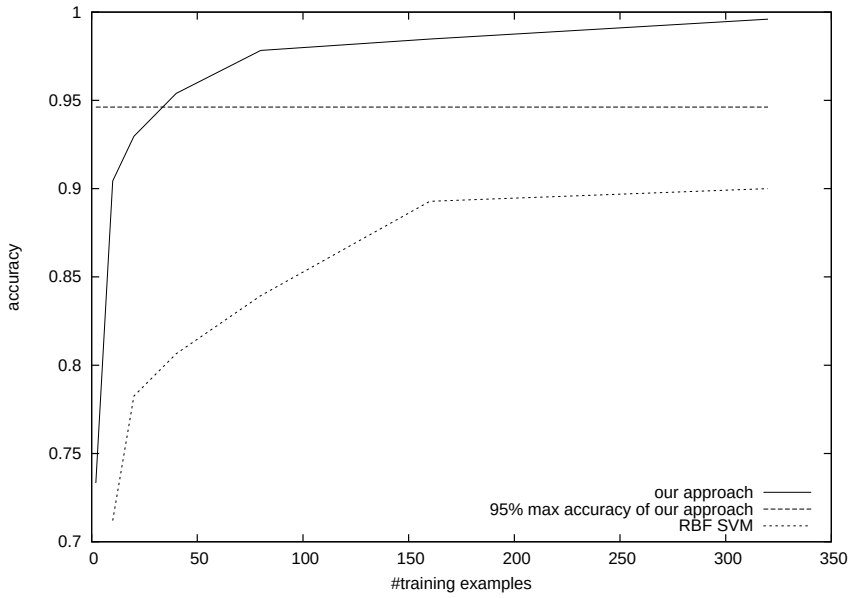
Experimental design

To quantify the number of class-labeled training examples our prior knowledge is worth, we use the learning curves obtained in Section 4.6.1, find the number of class-labeled training examples our approach needs to train on to achieve 95% of the highest accuracy of our approach in the learning curve, and compute the number of additional class-labeled training examples the RBF SVM needs to achieve the same accuracy. We treat the learning curves as piecewise linear curves during the computation, and extrapolate the curves if the model is unable to achieve the same accuracy within the total number of training examples available. Figure 4.5 and Figure 4.6 show the aforementioned 95% accuracy level along with the learning curves in Figure 4.1 and Figure 4.2. The length of the line segment of the 95% accuracy level between the two learning curves is our estimation of the number of class-labeled training examples the prior knowledge is worth in our approach.

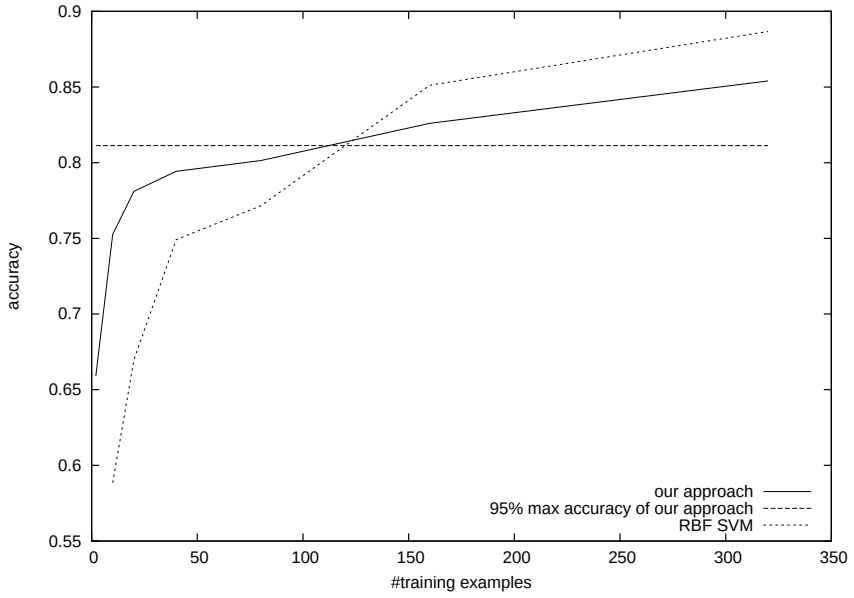
The choice of the 95% accuracy level is not absolute. The value of the prior knowledge changes depending on the accuracy level.

Empirical results

Table 4.6 shows the estimated number of training examples the prior knowledge is worth in our approach, sorted according to the estimation.

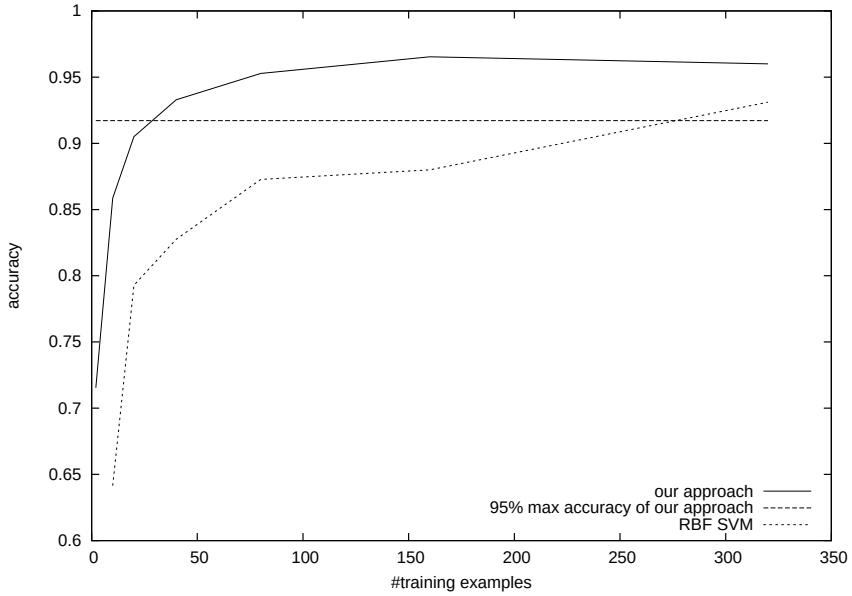


(a) Learning curve for the pair 壕 vs. 嚎

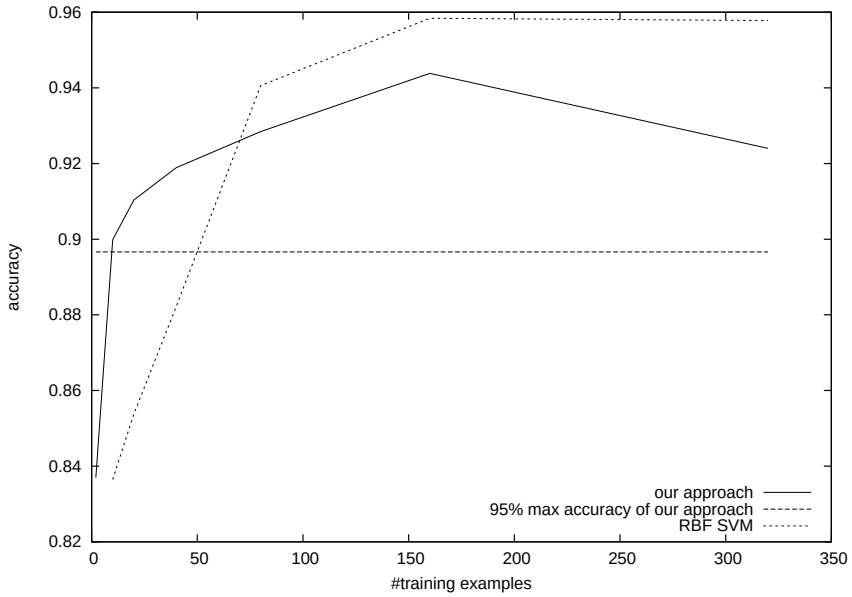


(b) Learning curve for the pair 晴 vs. 晴

Figure 4.5: Learning curves for our method and the RBF SVM along with the 95% accuracy level for our method on pairs of characters of which the SVM accuracies are relatively low compared to the human accuracies.



(a) Learning curve for the pair 涧 vs. 谰



(b) Learning curve for the pair 大 vs. 犬

Figure 4.6: Learning curves for our method and the RBF SVM along with the 95% accuracy level for our method on pairs of characters of which the SVM accuracies are close to the human accuracies.

壕 嚎	1313.04
木 朮	389.77
狠 狼	321.98
澜 谰	247.54
竟 竞	193.48
鸟 乌	132.79
扶 抉	124.61
便 使	121.74
鸣 呜	103.38
季 李	82.68
兔 免	58.71
孟 孟	45.46
大 犬	40.31
候 侯	36.74
白 自	33.39
伸 仲	31.07
睛 晴	7.75
酒 洒	-170.45
mean	173.00
standard deviation	302.29

Table 4.6: Estimated number of training examples the prior knowledge in our approach is worth. Shaded rows represent pairs of characters that the SVM accuracies relative to the human are higher than 98%.

Interpretation

Among the pairs of characters we tested on, the RBF SVM requires on average of about 173 additional class-labeled training examples to achieve the same 95% highest accuracy of our approach.

The examples in Table 4.6 can be roughly divided into 3 groups. The first group contains 5 pairs of characters. The prior knowledge in this group is worth hundreds or even thousands more additional training examples. The prior knowledge in the second group is not worth as much as that in the first group, and may benefit the classification if the number of available training examples is limited. The prior knowledge in the third group is not worth a lot of examples. For many pairs of characters in this group the SVM accuracies relative to the human are already high, and are represented as shaded rows in the table. In the case of the pair 酒 vs. 洒, the prior knowledge actually hurts. This usually means that the prior knowledge in our approach fails to focus on the right region accurately due to inadequate modeling, or that the statistical learner picks up useful information that we are not concentrating on.

4.7.2 Learning curves using reduced numbers of structurally labeled training examples

Overview

In Section 4.7.1 we measure the value of prior knowledge with a fixed number of structurally labeled training examples in terms of the equivalent number of additional class-labeled training examples. As can be seen in Table 4.6,

the worth of prior knowledge varies greatly among the different pairs of characters. In order to better understand the relation among prior knowledge, structurally labeled training examples, and class-labeled training examples, we investigate a few representative pairs of characters to study their behavior with reduced number of structurally labeled training examples.

There are two learning elements in our approach. One is the structural model, and the other is statistical classifier. Each of the elements requires a certain number of training examples to reach its optimality. We want to investigate the relationship between the number of training examples and the optimality of each learning element. We also want to investigate the fungibility between the structurally labeled training examples and the class-labeled training examples (i.e., to what extent a deficient number in one training set can be compensated by increasing the size of the other set).

Experimental design

We choose one pair from each of four different categories: 壕 vs. 嚎, a more complex pair of characters on which our approach works very well, 狠 vs. 狼, a pair of characters on which our approach works quite well, 鸣 vs. 鸣, a pair of characters on which our approach outperforms the SVM by a smaller margin, and 酒 vs. 洒, a pair of characters on which our approach does not work well.

For each pair of characters, we learn the best structural model and using 40, 30, and 20 structurally labeled training examples. We then repeatedly sample sets of training examples of sizes 2, 10, 20, 40, 80, 160, and 320 from class-labeled examples excluding the structurally labeled examples already

used. The remaining unused examples form a hold-out set for testing. As in Experiment 2A in Section 4.6.1, SVM kernel parameters are found by performing cross validation on the sampled training set using the cross validation posterior probability except for the 2-example training set, of which the parameters are take from the structurally labeled training examples. An RBF SVM is then trained using the gradient features for the target regions of the sampled training set, and tested on the hold-out set. We plot the test accuracy on the hold-out set versus the number of class-labeled training examples as the learning curve for our approach.

Empirical results

Figure 4.7, Figure 4.8, Figure 4.9, and Figure 4.10 show the resulting learning curve for the 4 pairs of characters.

Interpretation

The learning curves in Figure 4.7 show the result of the pair 狠 vs. 狠, and exhibit the more typical behavior that one would expect. With the reduced numbers of structurally labeled training examples, our approach concentrates information less effectively, resulting in lower accuracies. As the number of class-labeled training examples increases, the effects of information concentration brought forth by the prior knowledge in our approach become less prominent, and the learning curves eventually reach similar final accuracies.

Figure 4.8 shows the result of the pair 鸣 vs. 鸣. The accuracies also drop with reduced numbers of structurally labeled training examples. Further-

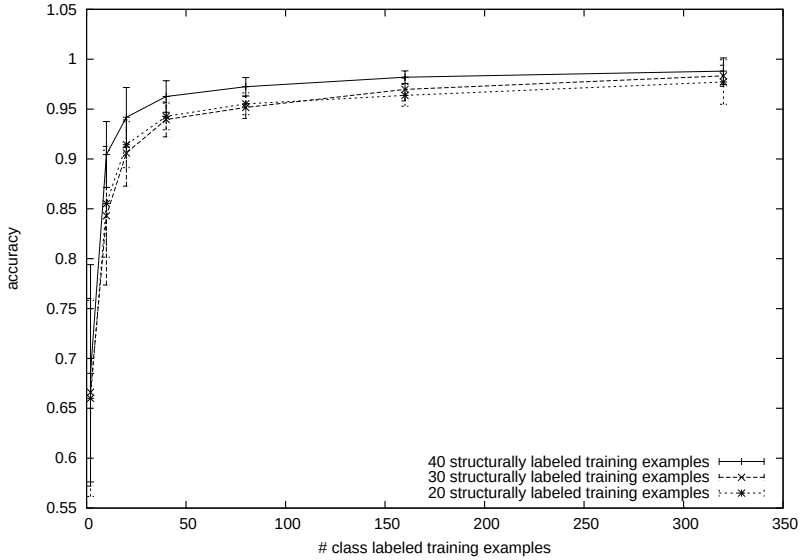


Figure 4.7: Learning curves for our method with 40, 30, and 20 structurally labeled training examples for the pair 狠 vs. 狼. The error bars are standard deviations of the accuracies.

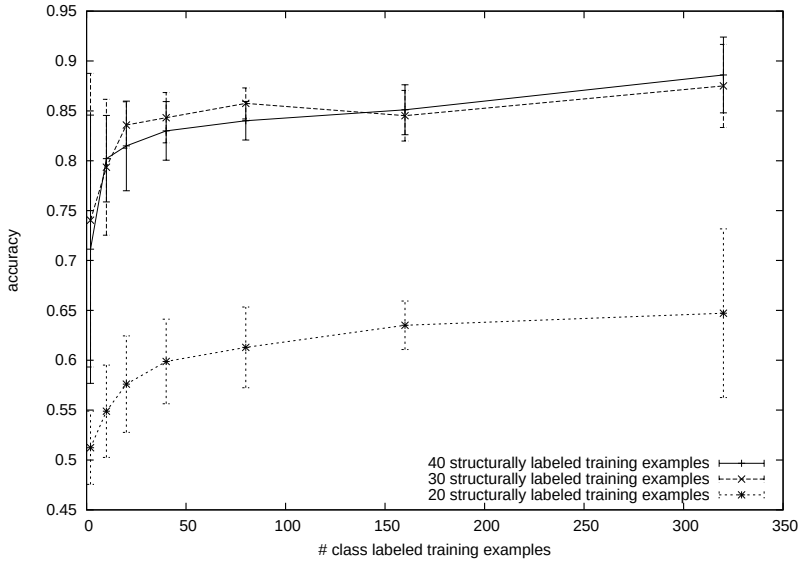


Figure 4.8: Learning curves for our method with 40, 30, and 20 structurally labeled training examples for the pair 鸣 vs. 鸣. The error bars are standard deviations of the accuracies.

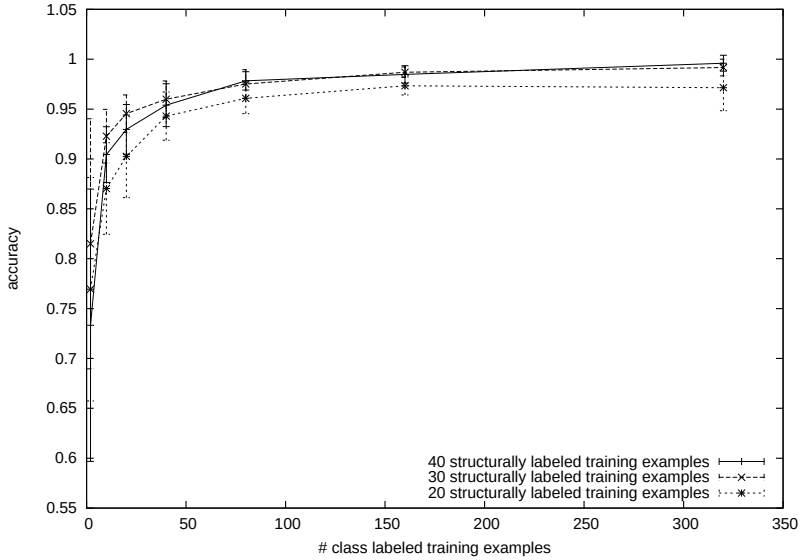


Figure 4.9: Learning curves for our method with 40, 30, and 20 structurally labeled training examples for the pair 壕 vs. 嚎. The error bars are standard deviations of the accuracies.

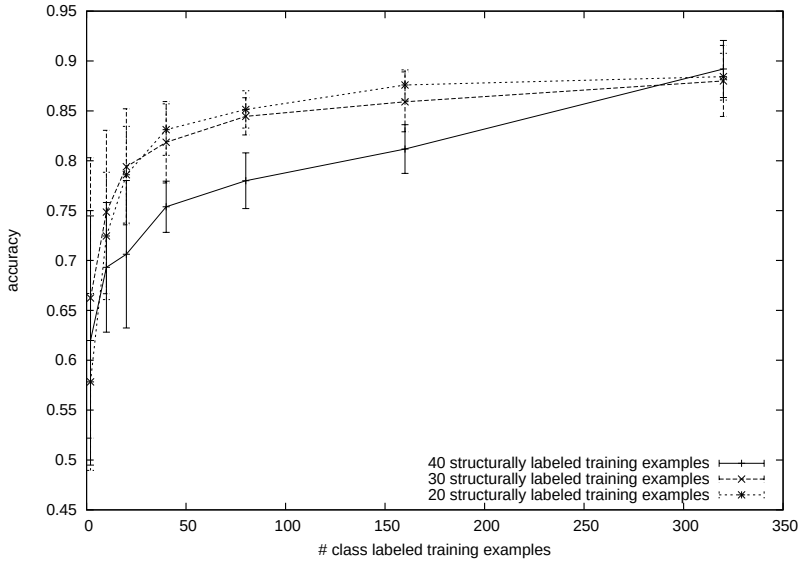


Figure 4.10: Learning curves for our method with 40, 30, and 20 structurally labeled training examples for the pair 酒 vs. 洒. The error bars are standard deviations of the accuracies.

more, with 20 structurally labeled training examples, our structural model is no longer able to concentrate useful information effectively, and the information it throws away overwhelms the information it concentrates, and the classification suffers greatly. This is due to the fact that the target region for this pair is harder to locate accurately, and thus our structural model requires more structurally labeled training examples to reach its optimality. The noise at the beginning of the learning curves is likely due to sampling error.

Figure 4.9 shows the result of the pair 壕 vs. 嚎. The learning cliff between 20 and 30 structurally labeled training examples for this pair also shows the similar phenomenon of reduced accuracies with reduced numbers of structurally labeled training examples. The learning curves for 30 structurally labeled training examples and 40 structurally labeled training examples almost overlap each other. This is an indication that our structural model is already optimal with 30 structurally labeled training examples, and additional structurally labeled training examples do not offer much more information that helps classification. Another interesting phenomenon is that the learning curve for 20 structurally labeled training examples flattens out after 160 class-labeled training examples. This shows that the concentrated information in our approach with 20 structurally labeled training examples is exhausted with 160 class-labeled training examples, and the statistical classifier is already optimal with 160 class-labeled training examples given the structural model. Additional class-labeled training examples do not further improve the accuracy due to the bias introduced along with the information concentration.

Figure 4.10 shows the result of the pair 酒 vs. 洒, a pair that our approach does not work well in the end in terms of final classification accuracy. The learning curves with fewer structurally labeled training examples are significantly steeper at the beginning. Our interpretation is that our structural model offers a stronger learning bias when calibrated with fewer training examples in this particular case. With more structurally labeled training examples, the learning bias from the structural model itself is weaker, meaning less concentrated information in the examples, and the approach requires more class-labeled training examples to overcome the variance and pick up the useful pattern.

Chapter 5

A Chinese Character Recognition System

In the previous chapters, we focus on improving the classification of pairs of structurally similar objects challenging to conventional statistical machine learners. In many multiclass classification problems, these challenging pairs account for a small portion of all distinctions, but contribute to a significant part of misclassifications. In this chapter, we use the offline handwritten Chinese character recognition system as an example, and provide a way of improving such multiclass classification by combining our approach and existing multiclass classification systems.

5.1 A Look at the Confusions

Given the multiclass LDA offline Chinese character recognition system described in [14], we compute the 3755×3755 confusion matrix for the HITPU

database. There are 9835 nonzero entries in the confusion matrix, excluding the correct classifications.

The overall recognition accuracy using multiclass LDA on this database is 92.62%. Among the 3755 character classes, there are 809 character classes with recognition accuracies lower than 90%, and 60 character classes with recognition accuracies lower than 80%.

We collect peak entries in the confusion matrix. An entry in the confusion matrix is considered a peak entry if it is being wrongly recognized as at least 5% of the time for any character. There are 346 peak entries in the confusion matrix, forming 255 pairs of character classes, consisting of 465 different character classes. We call the set of these 465 character classes *set A*. Among these peak confusions, there are 22 3-way confusions, 8 4-way confusions, and 1 5-way confusion.

Although the 346 peak entries are only 3.52% of the 9835 nonzero entries, they account for 10.13% of all errors. They account for even larger portion of the errors for characters with lower recognition rates. These peak entries account for 22.48% of the errors for characters with lower than 90% recognition rates, and 57.35% of the errors for characters with lower than 80% recognition rates.

Using the all-vs.-all voting strategy to deal with multi-way confusions, the SVM can improve the recognition accuracies of some of the peak confusions significantly. We call the subset of character classes in *set A* that the SVM can improve their recognition accuracies by over 40% *set B*.

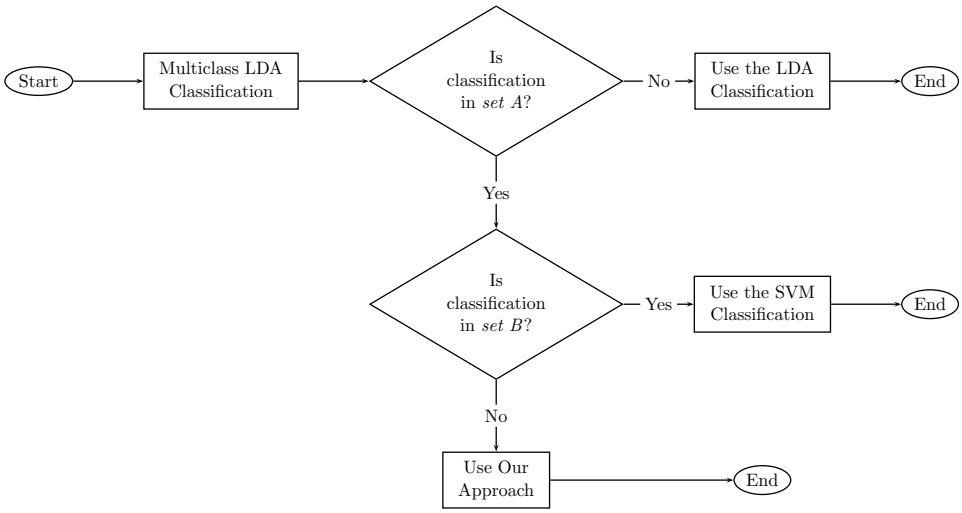


Figure 5.1: A full offline handwritten Chinese character recognition system using the combination of the multiclass LDA system, the SVM, and our approach.

5.2 A Full Offline Handwritten Chinese Character Recognition System

Figure 5.1 shows a combined classification strategy using the multiclass LDA system, the SVM, and our approach for offline handwritten Chinese characters. Given a test example, this system first applies the multiclass LDA system. If the classification is not among the character classes in *set A* as defined in Section 5.1, we use the LDA classification. If the classification is one of the character classes in *set B*, we apply the SVM to solve the confusion and use the SVM classification. Otherwise, we apply our approach and use its classification. We use the all-vs.-all voting strategy to solve multi-way confusions.

This combined classification strategy using the multiclass LDA, the SVM,

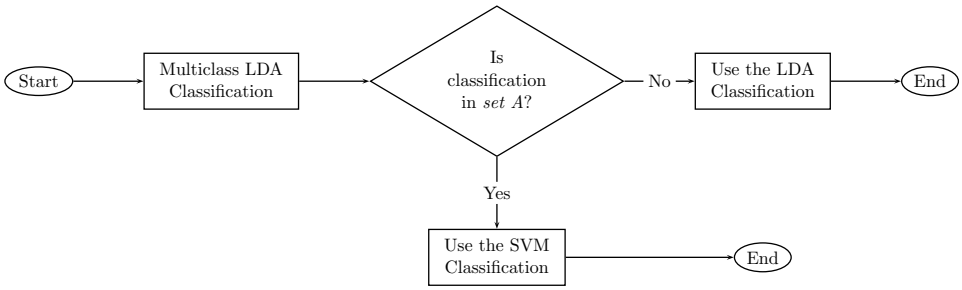


Figure 5.2: A full offline handwritten Chinese character recognition system using the combination of the multiclass LDA system and the SVM.

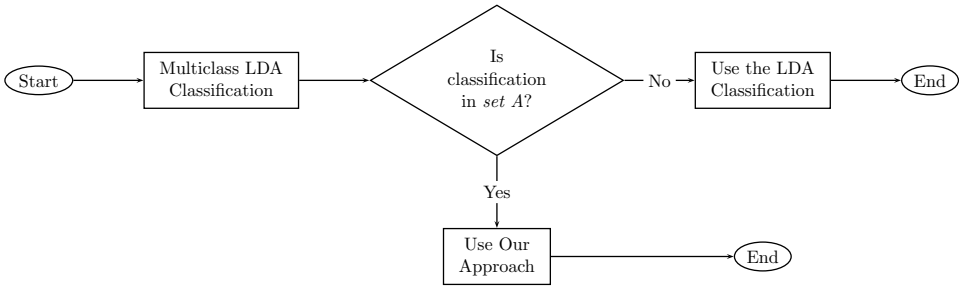


Figure 5.3: A full offline handwritten Chinese character recognition system using the combination of the multiclass LDA system and our approach.

and our approach not only outperforms the multiclass LDA system alone, but also does better than the combined system using only the multiclass LDA and the SVM, as depicted in Figure 5.2, and the system using only the multiclass LDA and our approach, as depicted in Figure 5.3.

5.3 Estimated Improvement

Among the peak confusions, we have experimental data for the SVM for 26 difficult pairs of characters, and for our approach for 18 difficult pairs of characters.

Using the experimental data that we have, the combined system using the multiclass LDA and the SVM improves over the multiclass LDA system by an average of 44.25% on the difficult pairs of characters. The combined system using the multiclass LDA and our approach improves over the multiclass LDA system by an average of 55.03% on the difficult pairs of characters.

Because our approach tends to do better on pairs of characters that the SVM does not significantly improve upon, if we use the combined strategy that utilizes the multiclass LDA, the SVM, and our approach as illustrated in Figure 5.1, the improvement over the multiclass LDA system on the difficult pairs of characters is 60.66%.

Assuming a more conservative estimate of 40% combined net improvement on the difficult pairs of characters using the strategy in Figure 5.1 and the distribution of errors described in Section 5.1, the estimated resulting improvement in error rate is 4.05% for all characters, 8.99% for characters with lower than 90% recognition accuracies, and 22.94% for characters with lower than 80% recognition accuracies.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

We proposed a new framework to improve classification of structurally similar objects by making use of prior domain knowledge to concentrate information. This concentration of information is achieved in two folds. First, the structures of the object are estimated through a sequence of model fittings. Then, once the structures are found, one crucial part of the example bearing the most discriminative information is identified, and information from the remaining parts of the example is discarded.

Each model in the sequence of model fitting operations is in the form of a generative model that renders part of the example. By performing an optimization we fit the variables in the generative model to the corresponding part in the example. These variables are the structural features that repre-

sent structures in the example. This optimization is not always convex, but by conditioning on the structural features found by previous model fittings in the sequence, the optimization is able to find the desired local optimum in a wide enough locally convex region. This conditioning is achieved by using a conditional prior as well as subtracting information of constituent parts of previously matched structures irrelevant to the current model fitting from the representation of the example. Parameters are optimized to enhance global sequential convexity and accuracy of the final estimate of the structures.

The approach concentrates information by discarding information from most of the example, leaving only the most crucial part of the example bearing the most information for classification. This concentration of information serves as a very strong learning bias that results in a much smaller hypothesis space. Due to the much smaller hypothesis space, the learning problem is easier, requiring much fewer training examples to learn the pattern. The quality of the final classifier depends on how well the information for classification is concentrated, which in turn depends on both how well our model is in localizing the crucial structure in the example and how the discriminative information is distributed within the example.

Using the structurally labeled training examples and prior knowledge, we find the correspondence between the structures of two classes of objects and identify the discriminative structures. Structurally labeled training examples are training examples with the parameters of its basic constituents annotated. We use mutual information among the constituents to construct the sequence of models suitable for estimating the structures.

The identification of the crucial structure as well as the construction of the sequence of models to evaluate the structural features are both automatic, given prior domain knowledge and a small set of structurally labeled training examples. Prior domain knowledge tells us how to model the most basic constituents of the objects, how to construct and evaluate the structural features out of the basic constituents, and how to subtract information of previously fitted structures from the representation of the example.

We applied this framework to the problem of classifying pairs of similar offline handwritten Chinese characters. We showed that by concentrating information, we are able to significantly improve the recognition rate of pairs of characters that the conventional purely statistical classifier does poorly compared to humans. We also showed that, regardless of whether our approach outperforms the purely statistical approach in the end, the concentrated information in our approach always results in easier learning problems, producing steeper learning curves, and consistently achieve higher accuracy when training data is limited. Our approach may not work well in the end if it cannot localize the crucial structures well because of model mismatch between the prior knowledge and the examples, or if the statistical machine learner is able to pick up discriminative information distributed in parts of the example that we do not concentrate on.

6.2 Future Work

6.2.1 Extending to classification problems of other structured objects

The framework we proposed is general, and can be applied to classification problems of other structured objects, including other real world objects in images as well as other structured objects in non-vision domains. To apply our framework, one requires the prior knowledge of how the object can be modeled using basic constituent parts, how to construct candidate structural features using the constituent parts, and how to condition the models in the sequence by using the conditional prior and by subtracting information of parts of the object from the representation of the example. Once we have this prior knowledge and the structurally labeled training examples, we can identify the discriminative structures, construct the sequence of generative models for the structural features, and evaluate the structural features by fitting the models.

Prior knowledge for offline handwritten Chinese characters is relatively simple because of the relatively simple way how the example is composed of its constituents. As an example, here is how one might model real world objects in an image, such as sail boats. The constituent models will need to include everything that can generate pixels in the image. The complexity of each model can vary depending on interest. One may want to use much simpler models for the sky and the sea than for the the hull and the sail. Compared to modeling the strokes in Chinese characters, the appearance model may additionally need to take into account color, lighting, angle,

and occlusion, etc. Structural features are constructed using simplified joint configurations of constituent parts. The models form segmentations of pixels in the image. Subtracting information from the image can be achieved by penalizing the model for matching pixels outside of its segment.

Unlike Chinese characters, in images of real world objects, one may need to take into account the possibility for a model to be completely absent from an example due to occlusion. This affects the identification of corresponding structures between classes, as well as the construction and evaluation of the structural features.

6.2.2 Using multiple target regions

In the approach we proposed, we only consider one target region for each classification problem. One can easily extend the approach to incorporate multiple target regions, generating a separate target region for each discriminative part. Discriminative features can be generated for each target region, and concatenated into a large feature vector for the statistical discriminative learner.

Compared to the alternative that uses one large target region that contains many discriminative parts, using many separate target regions the statistical learner can better discover discriminative patterns for each target region. It is an interesting problem to decide when to combine or separate the discriminative regions and to learn the weights for the target regions.

6.2.3 Learning a minimally sufficient model

In our approach, we always include all of the structural features deemed useful in estimating the structure in the sequence. This is sufficient for our approach, but not always necessary. One only needs to include the structural features necessary in locating the target region accurately enough. It is an interesting problem to construct a minimally sufficient model that locates the target region well enough for the classification task. A simpler classification task with classes more dissimilar from each other should require a simpler model.

6.2.4 Dynamically adapting the structural model

In addition to learning a minimally sufficient model as described in Section 6.2.3, one can instead consider deciding at *testing time* what models to use using "meta prior knowledge" that specifies when part of the prior knowledge is useful.

For example, in modeling Chinese characters, it may be sufficient to use a straight line to model each stroke for most of the examples. However, for some messier examples, it may be necessary to model the strokes as curves or to include extra recovery strokes to accurately detect the structures. One may decide to apply the more sophisticated model to the test example when the simple model is not good enough or by detecting the writing style of the character.

Dynamically adapting the structural model can also be one way of ac-

counting for the occluded parts in modeling real world objects alluded to earlier.

Bibliography

[1] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific and Statistical Computing*, 16(5):1190–1208, 1995.

[2] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27:1–27:27, April 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[3] Ming-Wei Chang, Lev Ratinov, Nicholas Rizzolo, and Dan Roth. Learning and inference with constraints. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, volume 3, pages 1513–1518. AAAI Press, 2008.

[4] Chi-Hau Chen and Patrick Shen-Pei Wang, editors. *Handbook of Pattern Recognition and Computer Vision*. World Scientific Publishing Co. Pte. Ltd., River Edge, NJ, USA, third edition, 2005.

[5] Navneet Dalal and Bill Triggs. Histogram of oriented gradient for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 886–893, 2005.

[6] Kai Ding, Zhibin Liu, Lianwen Jin, and Xinghua Zhu. A comparative study of Gabor feature and gradient feature for handwritten Chinese character recognition. In *International Conference on Wavelet Analysis and Pattern Recognition*, volume 3, pages 1182–1186, 2007.

[7] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: A benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 304–311, 2009.

[8] Kun Duan, Devi Parikh, David Crandall, and Kristen Grauman. Discovering localized attributes for fine-grained recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[9] Ian Endres, Vivek Srikumar, Ming-Wei Chang, and Derek Hoiem. Learning shared body plans. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[10] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(6):1627–1645, September 2010.

[11] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, second edition, 1990.

[12] Leonid Karlinsky, Michael Dinerstein, Daniel Harari, and Shimon Ullman. The chains model for detecting parts by their context. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 25–32, 2010.

[13] Nei Kato, Shin'ichiro Omachi, Hirotomo Aso, and Yoshiaki Nemoto. A handwritten character recognition system using directional element feature and asymmetric mahalanobis distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(3):258–262, March 1999.

[14] Fumitaka Kimura, Tetsushi Wakabayashi, Shinji Tsurouka, and Yasuji Miyake. Improvement of handwritten Japanese character recognition using weighted direction code histogram. *Pattern Recognition*, 30(8):1329–1337, 1997.

[15] Alexander Kraskov and Peter Grassberger. MIC: Mutual information based hierarchical clustering. In Frank Emmert-Streib and Matthias Dehmer, editors, *Information Theory and Statistical Learning*, chapter 5, pages 101–123. Springer, 2009.

[16] Geoffrey Levine and Gerald DeJong. Explanation-based object recognition. In *IEEE Workshop on Applications of Computer Vision*, pages 1–8, 2008.

[17] Shiau Hong Lim, Li-Lun Wang, and Gerald DeJong. Explanation-based feature construction. In *International Joint Conference on Artificial Intelligence*, pages 931–937, 2007.

[18] Shiau Hong Lim, Li-Lun Wang, and Gerald DeJong. Integrating prior domain knowledge into discriminative learning using phantom examples. In *Eleventh International Conference on Frontiers in Handwriting Recognition*, pages 523–528, 2008.

[19] Hsuan-Tien Lin, Chih-Jen Lin, and Ruby C. Weng. A note on Platt's probabilistic outputs for support vector machines. *Machine Learning*, 68(3):267–276, 2007.

[20] Hailong Liu and Xiaoqing Ding. Handwritten character recognition using gradient feature and quadric classifier with multiple discrimination schemes. In *Document Analysis and Recognition*, volume 1, pages 19–23, 2005.

[21] Enzhi Ni, Minjun Jiang, and Changle Zhou. Radical extraction for handwritten Chinese character recognition by using radical cascade classifier. In Xudong Wang, Fuzhong Wang, and Shaobo Zhong, editors, *Electrical, Information Engineering and Mechatronics 2011*, volume 138 of *Lecture Notes in Electrical Engineering*, chapter 49, pages 419–426. Springer, London, 2012.

[22] John Platt. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In Alexander J. Smola, Peter L. Bartlett, Bernhard Schölkopf, and Dale Schuurmans, editors, *Advances in Large-Margin Classifiers*. MIT Press, October 2000.

[23] Dan Roth and Wen-Tau Yih. Global inference for entity relation identification via a linear programming formulation. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*, chapter 20, pages 553–580. MIT Press, August 2007.

[24] T. Saito, H. Yamada, and K. Yamamoto. On the data base ETL9 of handprinted characters in JIS Chinese characters and its analysis. *IEICE Transactions on Information and Systems*, J68-D(4):757–764, April 1985. In Japanese.

[25] Daming Shi, Steve R. Gunn, and Robert I. Damper. Handwritten Chinese character recognition using nonlinear active shape models and the Viterbi algorithm. *Pattern Recognition Letters*, 23(14):1853–1862, 2002.

[26] Daming Shi, Steve R. Gunn, and Robert I. Damper. Handwritten Chinese radical recognition using nonlinear active shape models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):277–280, 2003.

[27] Qiang Sun and Gerald DeJong. Explanation-augmented SVM: An approach to incorporating domain knowledge into SVM learning. In *The 22th International Conference on Machine Learning*, 2005.

[28] Qiang Sun and Gerald DeJong. Feature kernel functions: Improving SVMs using high-level knowledge. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 177–183, 2005.

[29] Qiang Sun, Li-Lun Wang, and Gerald DeJong. Explanation-based learning for image understanding. In *Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence*, pages 1679–1682, 2006.

[30] T. Wakabayashi, S. Tsuruoka, F. Kimura, and Y. Miyake. On the size and variable transformation of feature vector for handwritten character recognition. *IEICE Transactions on Information and Systems*, J76-D2(12):2495–2503, December 1993. In Japanese.

[31] Hiromitsu Yamada, Kazuhiko Yamamoto, and Taiichi Saito. A nonlinear normalization method for handprinted kanji character recognition—line density equalization. *Pattern Recognition*, 23(9):1023–1029, 1990.

[32] Jia Zeng and Zhi-Qiang Liu. Type-2 fuzzy Markov random fields and their application to handwritten Chinese character recognition. *IEEE Transactions on Fuzzy Systems*, 16(3):747–760, June 2008.

[33] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, December 1997.

[34] Tiziano Zito, Niko Wilbert, Laurenz Wiskott, and Pietro Berkes. Modular toolkit for Data Processing (MDP): A Python data processing framework. *Frontiers in Neuroinformatics*, January 2009.