IEICE Transactions on Information and Systems,
vol.E93-D, no.10, pp.2690–2701, 2010.
Revised on June 26, 2011.

Superfast-Trainable Multi-Class Probabilistic Classifier by Least-Squares Posterior Fitting

Masashi Sugiyama (sugi@cs.titech.ac.jp)
Tokyo Institute of Technology
and
Japan Science and Technology Agency

Abstract

Kernel logistic regression (KLR) is a powerful and flexible classification algorithm, which possesses an ability to provide the confidence of class prediction. However, its training—typically carried out by (quasi-)Newton methods—is rather time-consuming. In this paper, we propose an alternative probabilistic classification algorithm called *Least-Squares Probabilistic Classifier* (LSPC). KLR models the class-posterior probability by the log-linear combination of kernel functions and its parameters are learned by (regularized) maximum likelihood. In contrast, LSPC employs the linear combination of kernel functions and its parameters are learned by regularized least-squares fitting of the true class-posterior probability. Thanks to this linear regularized least-squares formulation, the solution of LSPC can be computed analytically just by solving a regularized system of linear equations in a class-wise manner. Thus LSPC is computationally very efficient and numerically stable. Through experiments, we show that the computation time of LSPC is faster than that of KLR by orders of magnitude, with comparable classification accuracy.

Keywords

Probabilistic classification, kernel logistic regression, class-posterior probability, squared-loss.

1 Introduction

The *support vector machine* (SVM) [7, 33] is a popular method for classification. Various computationally efficient algorithms for training SVM with massive datasets have been proposed so far (see [24, 16, 5, 6, 29, 26, 32, 13, 11, 30, 17, 31, 12] and many other softwares available online). However, SVM cannot provide the *confidence* of class prediction since it only learns the decision boundaries between different classes. To cope with this problem,

several post-processing methods have been developed for approximately computing the class-posterior probability [25, 34].

On the other hand, *logistic regression* (LR) is a classification algorithm that can naturally give the confidence of class prediction since it directly learns the class-posterior probabilities [15]. Recently, various efficient algorithms for training LR models specialized in *sparse* data have been developed [22, 10].

Applying the *kernel trick* to LR as done in SVM, one can easily obtain a non-linear classifier with probabilistic outputs, called *kernel logistic regression* (KLR). Since the kernel matrix is often *dense* (e.g., Gaussian kernels), the state-of-the-art LR algorithms for sparse data are not applicable to KLR. Thus, in order to train KLR classifiers, standard non-linear optimization techniques such as Newton's method (more specifically, *iteratively reweighted least-squares*) and quasi-Newton methods (for example, the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) method) seem to be commonly used in practice [15, 23]. Although the performance of these general-purpose non-linear optimization techniques has been improved together with the evolution of computer environment in the last decade, computing the KLR solution is still challenging when the number of training samples is large. The purpose of this paper is to propose an alternative probabilistic classification method that can be trained very efficiently.

Our proposed method is called the *Least-Squares Probabilistic Classifier (LSPC)*. In LSPC, we use a linear combination of Gaussian kernels centered at training points as a model of class-posterior probabilities. Then we fit this model to the true class-posterior probability by least-squares[1]. An advantage of this linear least-squares formulation is that *consistency* is guaranteed without taking into account the normalization factor. In contrast, normalization is essential in the maximum-likelihood LR formulation; otherwise the likelihood tends to infinity. Thanks to the simplification brought by excluding the normalization factor from the optimization criterion, we can compute the globally optimal solution of LSPC *analytically* just by solving a system of linear equations.

Furthermore, we show that the use of a linear combination of kernel functions in LSPC allows us to learn the parameters in a class-wise manner. This highly contributes to further reducing the computational cost particularly in multi-class classification scenarios. Through experiments, we show that LSPC is computationally much more efficient than KLR with comparable accuracy.

2 Least-squares Approach to Probabilistic Classification

In this section, we formulate the problem of probabilistic classification and give a new method in the least-squares framework.

[1] A least-squares formulation has been employed for improving the computational efficiency of SVMs [29, 26, 13]. However, these approaches deal with deterministic classification, not probabilistic classification.

2.1 Problem Formulation

Let $\mathcal{X}$ ($\subset \mathbb{R}^d$) be the input domain, where d is the dimensionality of the input domain. Let $\mathcal{Y} = \{1, \ldots, c\}$ be the set of labels, where c is the number of classes. Let us consider a joint probability distribution on $\mathcal{X} \times \mathcal{Y}$ with joint probability density $p(\boldsymbol{x}, y)$. Suppose that we are given n independent and identically distributed (i.i.d.) paired samples of input $\boldsymbol{x}$ and output y:

$$\{(\boldsymbol{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^n.$$

The goal is to estimate the class-posterior probability $p(y|\boldsymbol{x})$ from the samples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. The class-posterior probability allows us to classify test sample $\boldsymbol{x}$ to class $\widehat{y}$ with confidence $p(\widehat{y}|\boldsymbol{x})$:

$$\widehat{y} := \operatorname*{argmax}_y p(y|\boldsymbol{x}).$$

Let us denote the marginal density of $\boldsymbol{x}$ by $p(\boldsymbol{x})$ and we assume that it is strictly positive:

$$p(\boldsymbol{x}) > 0 \text{ for all } \boldsymbol{x} \in \mathcal{X}.$$

Then, by definition, the class-posterior probability $p(y|\boldsymbol{x})$ can be expressed as

$$p(y|\boldsymbol{x}) = \frac{p(\boldsymbol{x}, y)}{p(\boldsymbol{x})}. \tag{1}$$

This expression will be utilized in the derivation of the proposed method below.

2.2 Linear Least-squares Fitting of Class-posterior Probability

Here we introduce our least-squares fitting idea. We begin with the formulation for learning the class-posterior probability $p(y|\boldsymbol{x})$ as a function of both $\boldsymbol{x}$ and y, i.e., the class-posterior probabilities for all classes are learned simultaneously. Then in Section 2.3, we show that this simultaneous learning problem can be decomposed into independent class-wise learning problems, which highly contributes to reducing the computational cost.

We model the class-posterior probability $p(y|\boldsymbol{x})$ by the following linear model:

$$q(y|\boldsymbol{x}; \boldsymbol{\alpha}) := \sum_{\ell=1}^{b} \alpha_\ell \phi_\ell(\boldsymbol{x}, y) = \boldsymbol{\alpha}^\top \boldsymbol{\phi}(\boldsymbol{x}, y),$$

where $^\top$ denotes the transpose of a matrix or a vector,

$$\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_b)^\top$$

are parameters to be learned from samples, and

$$\boldsymbol{\phi}(\boldsymbol{x}, y) = (\phi_1(\boldsymbol{x}, y), \ldots, \phi_b(\boldsymbol{x}, y))^\top$$

are basis functions such that

$$\boldsymbol{\phi}(\boldsymbol{x}, y) \geq \boldsymbol{0}_b \quad \text{for all } (\boldsymbol{x}, y) \in \mathcal{X} \times \mathcal{Y}.$$

$\mathbf{0}_b$ denotes the b-dimensional vector with all zeros and the inequality for vectors is applied in the element-wise manner. We explain how the basis functions $\boldsymbol{\phi}(\boldsymbol{x}, y)$ are practically chosen in Section 2.3.

We determine the parameter $\boldsymbol{\alpha}$ in the model $q(y|\boldsymbol{x}; \boldsymbol{\alpha})$ so that the following squared error J is minimized:

$$\begin{aligned} J(\boldsymbol{\alpha}) &:= \frac{1}{2}\sum_{y=1}^{c}\int \left(q(y|\boldsymbol{x};\boldsymbol{\alpha}) - p(y|\boldsymbol{x})\right)^2 p(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \\ &= \frac{1}{2}\sum_{y=1}^{c}\int q(y|\boldsymbol{x};\boldsymbol{\alpha})^2 p(\boldsymbol{x})\mathrm{d}\boldsymbol{x} - \sum_{y=1}^{c}\int q(y|\boldsymbol{x};\boldsymbol{\alpha})p(\boldsymbol{x},y)\mathrm{d}\boldsymbol{x} + \mathrm{Const.} \\ &= \frac{1}{2}\boldsymbol{\alpha}^\top \boldsymbol{H}\boldsymbol{\alpha} - \boldsymbol{h}^\top\boldsymbol{\alpha} + \mathrm{Const.}, \end{aligned}$$

where we used Eq.(1). The $b \times b$ matrix $\boldsymbol{H}$ and the b-dimensional vector $\boldsymbol{h}$ are defined as

$$\begin{aligned} \boldsymbol{H} &:= \sum_{y=1}^{c}\int \boldsymbol{\phi}(\boldsymbol{x},y)\boldsymbol{\phi}(\boldsymbol{x},y)^\top p(\boldsymbol{x})\mathrm{d}\boldsymbol{x}, \\ \boldsymbol{h} &:= \sum_{y=1}^{c}\int \boldsymbol{\phi}(\boldsymbol{x},y)p(\boldsymbol{x},y)\mathrm{d}\boldsymbol{x}. \end{aligned}$$

$\boldsymbol{H}$ and $\boldsymbol{h}$ contain the expectations over unknown densities $p(\boldsymbol{x})$ and $p(\boldsymbol{x}, y)$, so we approximate the expectations by sample averages. Then we have

$$\begin{aligned} \widehat{\boldsymbol{H}} &:= \frac{1}{n}\sum_{y=1}^{c}\sum_{i=1}^{n}\boldsymbol{\phi}(\boldsymbol{x}_i,y)\boldsymbol{\phi}(\boldsymbol{x}_i,y)^\top, \\ \widehat{\boldsymbol{h}} &:= \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{\phi}(\boldsymbol{x}_i,y_i). \end{aligned}$$

Now our optimization criterion is formulated as

$$\widehat{\boldsymbol{\alpha}} := \underset{\boldsymbol{\alpha}\in\mathbb{R}^b}{\mathrm{argmin}}\left[\frac{1}{2}\boldsymbol{\alpha}^\top\widehat{\boldsymbol{H}}\boldsymbol{\alpha} - \widehat{\boldsymbol{h}}^\top\boldsymbol{\alpha} + \lambda\boldsymbol{\alpha}^\top\boldsymbol{\alpha}\right],$$

where a regularizer $\lambda\boldsymbol{\alpha}^\top\boldsymbol{\alpha}$ ($\lambda > 0$) is included for regularization purposes. Taking the derivative of the above objective function and equating it to zero, we see that the solution $\widehat{\boldsymbol{\alpha}}$ can be obtained just by solving the following system of linear equations.

$$(\widehat{\boldsymbol{H}} + \lambda\boldsymbol{I}_b)\boldsymbol{\alpha} = \widehat{\boldsymbol{h}}, \tag{2}$$

where $\boldsymbol{I}_b$ denotes the b-dimensional identity matrix. Thus, the solution $\widehat{\boldsymbol{\alpha}}$ is given analytically as

$$\widehat{\boldsymbol{\alpha}} = (\widehat{\boldsymbol{H}} + \lambda\boldsymbol{I}_b)^{-1}\widehat{\boldsymbol{h}}.$$

In order to assure that the solution $q(y|\boldsymbol{x};\widehat{\boldsymbol{\alpha}})$ is a conditional probability, we round up negative outputs to zero [35] and renormalize the solution. Consequently, our final solution is expressed as

$$\widehat{p}(y|\boldsymbol{x}) = \frac{\max(0, \widehat{\boldsymbol{\alpha}}^\top \boldsymbol{\phi}(\boldsymbol{x}, y)}{\sum_{y'=1}^{c} \max(0, \widehat{\boldsymbol{\alpha}}^\top \boldsymbol{\phi}(\boldsymbol{x}, y'))}, \tag{3}$$

We call the above method *Least-Squares Probabilistic Classifier (LSPC)*. LSPC can be regarded as an application of a density ratio estimation method called the *unconstrained Least-Squares Importance Fitting (uLSIF)* [18] to probabilistic classification. Thus all the theoretical properties of uLSIF such as consistency, the rate of convergence, and numerical stability [19, 20] may be directly translated into the current context.

2.3 Basis Function Design

A naive choice of basis functions $\boldsymbol{\phi}(\boldsymbol{x}, y)$ would be a *kernel* model, i.e., for some kernel function K',

$$q(y|\boldsymbol{x};\boldsymbol{\alpha}) = \sum_{y'=1}^{c} \sum_{\ell=1}^{n} \alpha_\ell^{(y')} K'(\boldsymbol{x}, \boldsymbol{x}_\ell, y, y'), \tag{4}$$

which contains cn parameters. For this model, the computational complexity for solving Eq.(2) is $\mathcal{O}(c^3n^3)$.

Here we propose to separate input $\boldsymbol{x}$ and output y, and use the *delta kernel* for y (as in KLR):

$$q(y|\boldsymbol{x};\boldsymbol{\alpha}) = \sum_{y'=1}^{c} \sum_{\ell=1}^{n} \alpha_\ell^{(y')} K(\boldsymbol{x}, \boldsymbol{x}_\ell) \delta_{y,y'},$$

where K is a kernel function for $\boldsymbol{x}$ and $\delta_{y,y'}$ is the *Kronecker delta*:

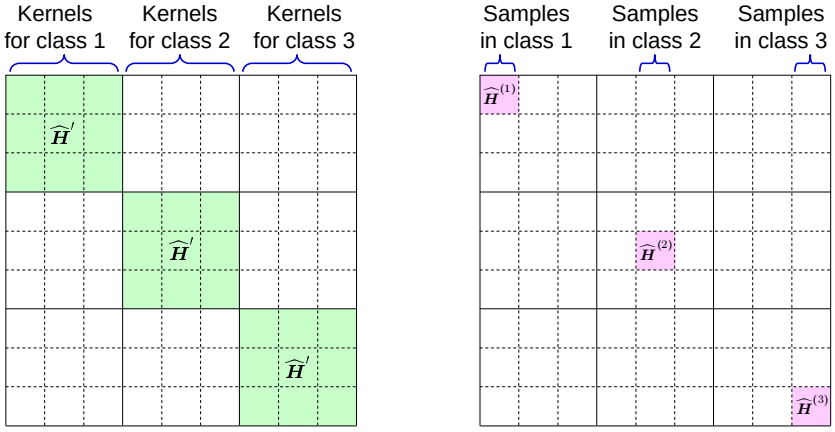
$$\delta_{y,y'} = \begin{cases} 1 & \text{if } y = y', \\ 0 & \text{otherwise.} \end{cases}$$

This model choice actually allows us to speed up the computation of LSPC significantly since all the calculations can be carried out *separately* in a class-wise manner. Indeed, the above model for class y is expressed as

$$q(y|\boldsymbol{x};\boldsymbol{\alpha}) = \sum_{\ell=1}^{n} \alpha_\ell^{(y)} K(\boldsymbol{x}, \boldsymbol{x}_\ell). \tag{5}$$

Then the matrix $\widehat{\boldsymbol{H}}$ becomes block-diagonal, as illustrated in Figure 1(a). Thus we only need to train a model with n parameters separately c times for each class y, by solving the following equation:

$$(\widehat{\boldsymbol{H}}' + \lambda \boldsymbol{I}_n)\boldsymbol{\alpha}^{(y)} = \widetilde{\boldsymbol{h}}^{(y)},$$



(a) Model (5) (b) Model (7)

Figure 1: Structure of matrix $\widehat{\boldsymbol{H}}$ for model (5) and model (7). The number of classes is $c = 3$. Suppose training samples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ are sorted according to label y. Colored blocks are non-zero and others are zeros. For model (5) consisting of c sets of n basis functions, the matrix $\widehat{\boldsymbol{H}}$ becomes block-diagonal (with common block matrix $\widehat{\boldsymbol{H}}'$), and thus training can be carried out separately for each block. For model (7) consisting of c sets of n_y basis functions, the size of the target block is further reduced.

where $\widehat{\boldsymbol{H}}'$ is the $n \times n$ matrix and $\widetilde{\boldsymbol{h}}^{(y)}$ is the n-dimensional vector defined as

$$\widehat{H}'_{\ell,\ell'} := \frac{1}{n}\sum_{i=1}^{n} K(\boldsymbol{x}_i, \boldsymbol{x}_\ell) K(\boldsymbol{x}_i, \boldsymbol{x}_{\ell'}),$$

$$\widetilde{h}^{(y)}_{\ell} := \frac{1}{n}\sum_{i=1}^{n} K(\boldsymbol{x}_i, \boldsymbol{x}_\ell) \delta_{y,y_i}.$$

Since $\widehat{\boldsymbol{H}}'$ is common to all y, we only need to compute $(\widehat{\boldsymbol{H}}' + \lambda \boldsymbol{I}_n)^{-1}$ once. Then the computational complexity for obtaining the solution is $\mathcal{O}(n^3 + cn^2)$, which is smaller than the case with general kernel model (4). Thus this approach would be computationally efficient when the number of classes c is large.

Here, we further propose to reduce the number of kernels in model (5). To this end, we focus on a kernel function $K(\boldsymbol{x}, \boldsymbol{x}')$ that is "localized". Examples of such localized kernels include the popular *Gaussian kernel* [28]:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2\sigma^2}\right). \tag{6}$$

Our idea is to reduce the number of kernels by locating the kernels only at samples belonging to the *target* class:

$$q(y|\boldsymbol{x}; \boldsymbol{\alpha}) = \sum_{\ell=1}^{n_y} \alpha^{(y)}_\ell K(\boldsymbol{x}, \boldsymbol{x}^{(y)}_\ell), \tag{7}$$

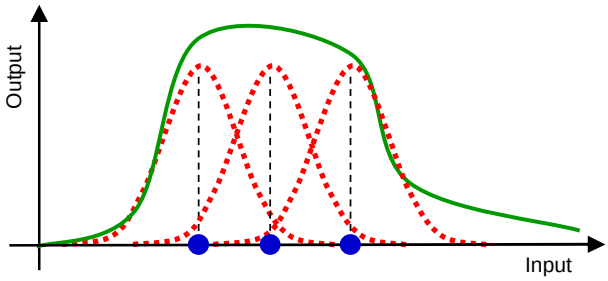


Figure 2: Heuristic of reducing the number of basis functions—locate Gaussian kernels only at the samples of the target class.

where n_y is the number of training samples in class y, and $\{\boldsymbol{x}_i^{(y)}\}_{i=1}^{n_y}$ is the training input samples in class y.

The rationale behind this model simplification is as follows (Figure 2). By definition, the class-posterior probability $p(y|\boldsymbol{x})$ takes large values in the regions where samples in class y are dense; conversely, $p(y|\boldsymbol{x})$ takes smaller values (i.e., close to zero) in the regions where samples in class y are sparse. When a non-negative function is approximated by a Gaussian kernel model, many kernels may be needed in the region where the output of the target function is large; on the other hand, only a small number of kernels would be enough in the region where the output of the target function is close to zero. Following this heuristic, many kernels are allocated in the region where $p(y|\boldsymbol{x})$ takes large values, which can be achieved by Eq.(7).

This model simplification allows us to further reduce the computational cost since the size of the target blocks in matrix $\widehat{\boldsymbol{H}}$ is further reduced, as illustrated in Figure 1(b). In order to learn the n_y-dimensional parameter vector

$$\boldsymbol{\alpha}^{(y)} = (\alpha_1^{(y)}, \ldots, \alpha_{n_y}^{(y)})^\top$$

for each class y, we only need to solve the following system of n_y linear equations:

$$(\widehat{\boldsymbol{H}}^{(y)} + \lambda \boldsymbol{I}_{n_y})\boldsymbol{\alpha}^{(y)} = \widehat{\boldsymbol{h}}^{(y)}, \tag{8}$$

where $\widehat{\boldsymbol{H}}^{(y)}$ is the $n_y \times n_y$ matrix and $\widehat{\boldsymbol{h}}^{(y)}$ is the n_y-dimensional vector defined as

$$\widehat{H}_{\ell,\ell'}^{(y)} := \frac{1}{n}\sum_{i=1}^{n} K(\boldsymbol{x}_i, \boldsymbol{x}_\ell^{(y)})K(\boldsymbol{x}_i, \boldsymbol{x}_{\ell'}^{(y)}), \tag{9}$$

$$\widehat{h}_\ell^{(y)} := \frac{1}{n}\sum_{i=1}^{n_y} K(\boldsymbol{x}_i^{(y)}, \boldsymbol{x}_\ell^{(y)}).$$

Let $\widehat{\boldsymbol{\alpha}}^{(y)}$ be the solution of Eq.(8). Then our final solution is given by

$$\widehat{p}(y|\boldsymbol{x}) = \frac{\max(0, \sum_{\ell=1}^{n_y} \widehat{\alpha}_\ell^{(y)} K(\boldsymbol{x}, \boldsymbol{x}_\ell^{(y)}))}{\sum_{y'=1}^{c} \max(0, \sum_{\ell=1}^{n_{y'}} \widehat{\alpha}_\ell^{(y')} K(\boldsymbol{x}, \boldsymbol{x}_\ell^{(y')}))}. \tag{10}$$

Input: Labeled training samples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$
(equivalently, $\{\boldsymbol{x}_i^{(y)}\}_{i=1}^{n_y}$ for class $y = 1, \ldots, c$),
Gaussian width σ, and regularization parameter λ;

Output: Class-posterior probability $\widehat{p}(y|\boldsymbol{x})$;

for $y = 1, \ldots, c$

$$\widehat{H}_{\ell,\ell'}^{(y)} \longleftarrow \frac{1}{n}\sum_{i=1}^{n} \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_\ell^{(y)}\|^2 + \|\boldsymbol{x}_i - \boldsymbol{x}_{\ell'}^{(y)}\|^2}{2\sigma^2}\right) \quad \text{for } \ell, \ell' = 1, \ldots, n_y;$$

$$\widehat{h}_\ell^{(y)} \longleftarrow \frac{1}{n}\sum_{i=1}^{n_y} \exp\left(-\frac{\|\boldsymbol{x}_i^{(y)} - \boldsymbol{x}_\ell^{(y)}\|^2}{2\sigma^2}\right) \quad \text{for } \ell = 1, \ldots, n_y;$$

Solve linear equation $(\widehat{\boldsymbol{H}}^{(y)} + \lambda \boldsymbol{I}_{n_y})\boldsymbol{\alpha}^{(y)} = \widehat{\boldsymbol{h}}^{(y)}$ and obtain $\widehat{\boldsymbol{\alpha}}^{(y)}$;

end

$$\widehat{p}(y|\boldsymbol{x}) \longleftarrow \frac{\max\left(0, \sum_{\ell=1}^{n_y} \widehat{\alpha}_\ell^{(y)} \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}_\ell^{(y)}\|^2}{2\sigma^2}\right)\right)}{\sum_{y'=1}^{c} \max\left(0, \sum_{\ell=1}^{n_{y'}} \widehat{\alpha}_\ell^{(y')} \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}_\ell^{(y')}\|^2}{2\sigma^2}\right)\right)};$$

Figure 3: Pseudo code of LSPC for simplified model (7) with Gaussian kernel (6).

For the simplified model (7), the computational complexity for obtaining the solution is $\mathcal{O}(cn_y^2 n)$—when $n_y = n/c$ for all y, this is equal to $\mathcal{O}(c^{-1}n^3)$. Thus this approach is computationally highly efficient for multi-class problems.

A pseudo code of the simplest LSPC implementation for Gaussian kernels is summarized in Figure 3. Its MATLAB® implementation is available from

`http://sugiyama-www.cs.titech.ac.jp/~sugi/software/LSPC/`

3 Experiments

In this section, we experimentally compare the performance of the following classification methods:

- **LSPC**: LSPC with model (7).
- **LSPC(full)**: LSPC with model (5).
- **KLR**: ℓ_2-penalized kernel logistic regression with Gaussian kernels. We used a MATLAB® implementation included in the '*minFunc*' package [27], which uses limited-memory BFGS updates with Shanno-Phua scaling in computing the step direction and a bracketing line-search for a point satisfying the strong Wolfe conditions to compute the step size.

When we fed data to learning algorithms, the input samples were normalized in the element-wise manner so that each element has mean zero and unit variance. The Gaussian width σ and the regularization parameter λ for all the methods are chosen based on 2-fold cross-validation from

$$\sigma \in \{\tfrac{1}{10}m, \tfrac{1}{5}m, \tfrac{1}{2}m, \tfrac{2}{3}m, m, \tfrac{3}{2}m, 2m, 5m, 10m\},$$
$$\lambda \in \{10^{-2}, 10^{-1.5}, 10^{-1}, 10^{-0.5}, 10^{0}\},$$

where

$$m := \mathrm{median}(\{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|\}_{i,j=1}^{n}).$$

3.1 Illustrative Examples

First, we illustrate the behavior of each method using a toy dataset.

We set the dimension of the input space to $d = 2$ and the number of classes to $c = 3$. We independently drew samples in each class from the following class-conditional sample densities (see Figure 4):

$$p(\boldsymbol{x}|y=1) = N\left(\boldsymbol{x}; \begin{bmatrix}-2\\0\end{bmatrix}, \begin{bmatrix}1&0\\0&1\end{bmatrix}\right),$$
$$p(\boldsymbol{x}|y=2) = N\left(\boldsymbol{x}; \begin{bmatrix}2\\0\end{bmatrix}, \begin{bmatrix}1&0\\0&1\end{bmatrix}\right),$$
$$p(\boldsymbol{x}|y=3) = \frac{1}{2}N\left(\boldsymbol{x}; \begin{bmatrix}0\\-3\end{bmatrix}, \begin{bmatrix}4&0\\0&1\end{bmatrix}\right) + \frac{1}{2}N\left(\boldsymbol{x}; \begin{bmatrix}0\\2\end{bmatrix}, \begin{bmatrix}1&0\\0&4\end{bmatrix}\right),$$

where $N(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ denotes the Gaussian density with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. We set the class-prior probabilities $p(y)$ as

$$p(y) = \begin{cases} 1/4 & \text{if } y = 1, 2, \\ 1/2 & \text{if } y = 3, \end{cases}$$

and we set the number of training samples to $n = 200$. Generated samples are plotted in Figure 5.

The true class-posterior probabilities $p(y|\boldsymbol{x})$ ($\propto p(\boldsymbol{x}|y)p(y)$), their estimates obtained by LSPC, LSPC(full), and KLR are depicted in Figure 6. The plots show that all the methods approximate the true class-posterior probabilities well in the training region (say, $[-5, 5]^2$). However, the output outside the training region is substantially different in LSPC and KLR. This is induced by the difference of the models—a linear combination of Gaussian kernels is used in LSPC, while its exponent is used in KLR. Outside the training region, there is no kernel, and thus a linear combination of Gaussian kernels takes values close to zero (note that the values are not exactly zero since Gaussian tails extended from training regions remain everywhere); then typically one of the classes takes a value close to one, and the others tend to zero outside the training regions. On the other

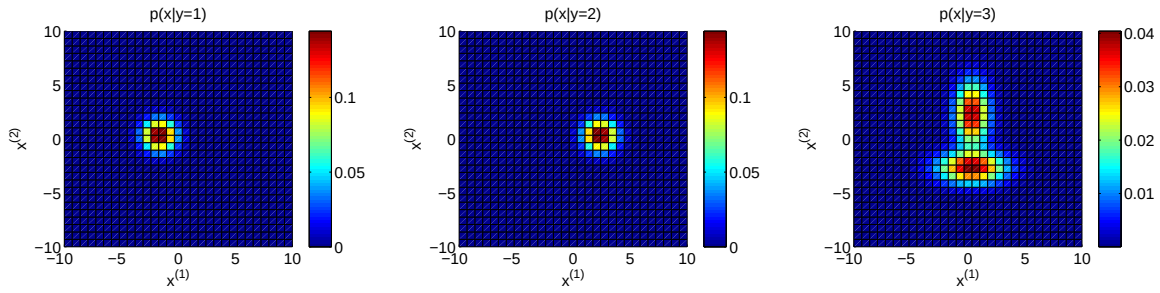


Figure 4: Illustrative examples. Class-conditional sample densities $p(\boldsymbol{x}|y)$.

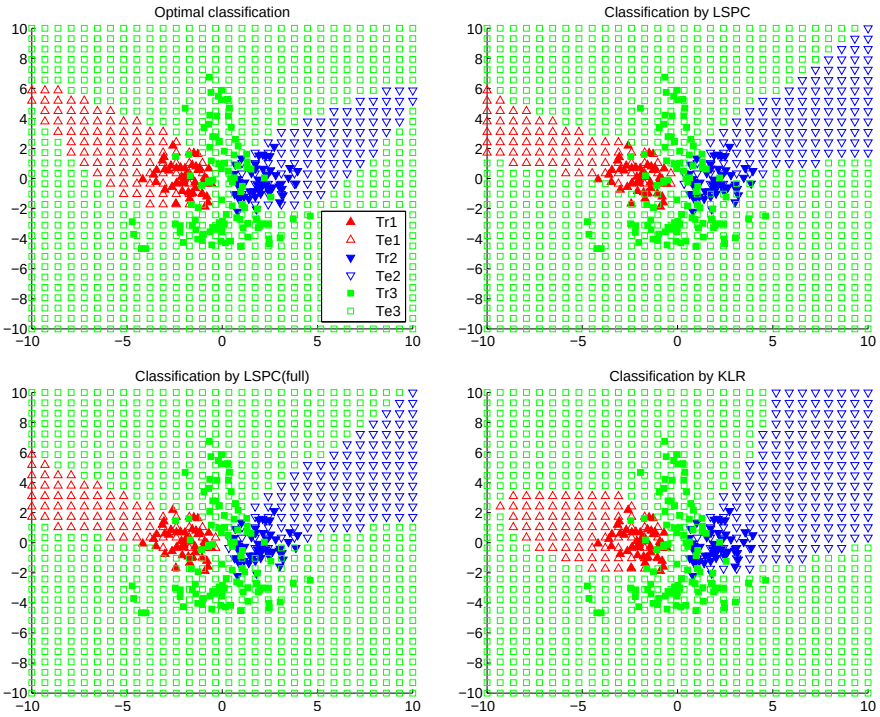


Figure 5: Illustrative examples. Training samples are plotted with filled symbols. Unfilled symbols denote the classification results based on the true class-posterior probabilities and their estimates obtained by LSPC, LSPC(full), and KLR.

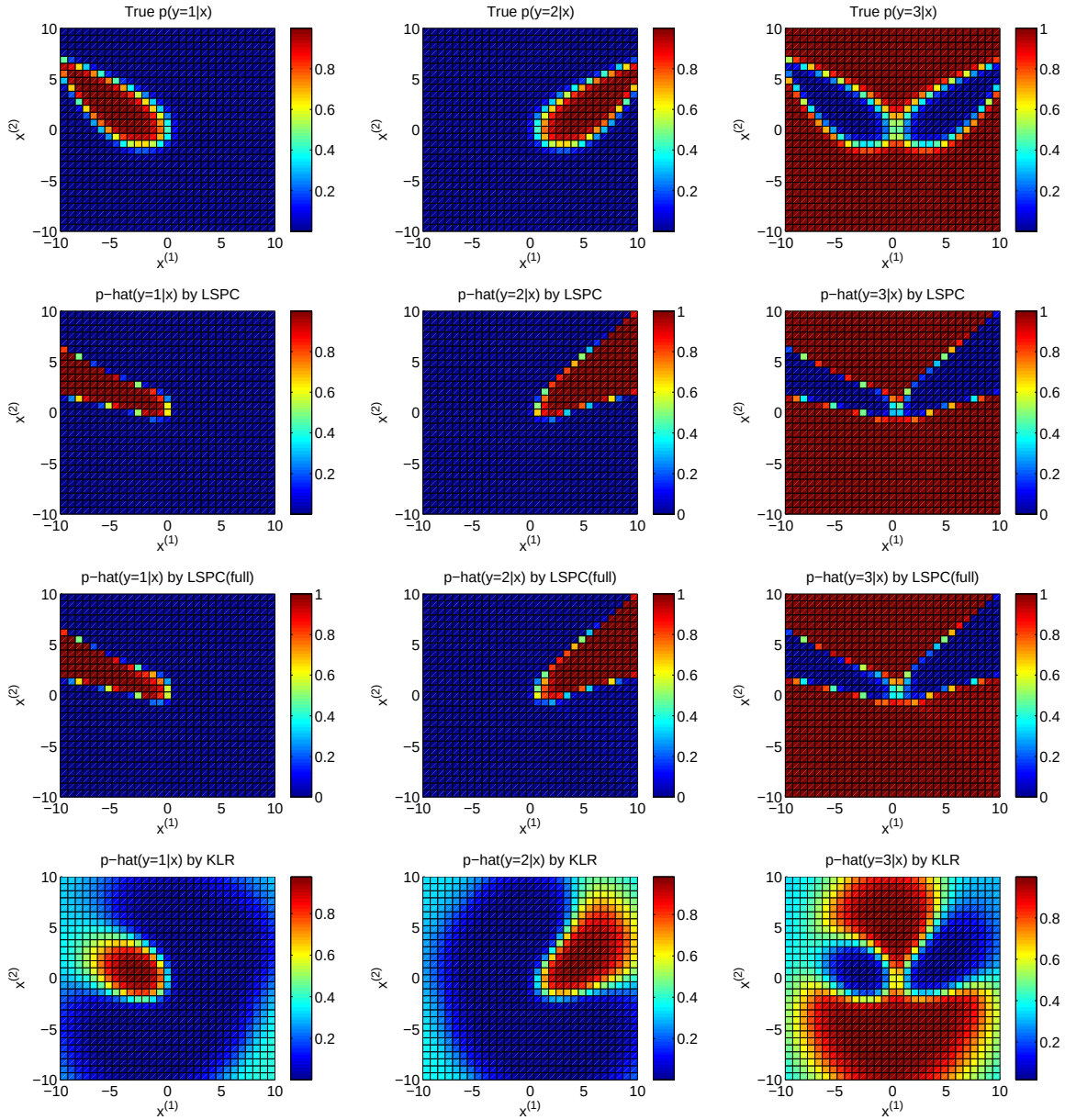


Figure 6: Illustrative examples. The plots show the true class-posterior probabilities $p(y|\boldsymbol{x})$, their estimates by LSPC, LSPC(full), and KLR from top to bottom, and $y = 1, 2, 3$ from left to right.

hand, KLR outputs values close to one outside the training region since $\exp(0) = 1$; then they are normalized and thus are reduced to $1/c$.

The classification results based on the true class-posterior probabilities and their estimates obtained by LSPC, LSPC(full), and KLR are plotted in Figure 5. This shows that all the method gave reasonable classification results.

3.2 Performance Comparison

Next, we evaluate the classification accuracy and computation time of each method using the following multi-class classification datasets taken from the *LIBSVM* web page [5]:

- **mnist**: Input dimensionality is 717 and the number of classes is 10.
- **usps**: Input dimensionality is 256 and the number of classes is 10.
- **satimage**: Input dimensionality is 36 and the number of classes is 6.
- **letter**: Input dimensionality is 16 and the number of classes is 26.

We investigated the classification accuracy and computation time of LSPC, LSPC(full), and KLR. For given n and c, we randomly chose $n_y = \lfloor n/c \rfloor$ training samples from each class y, where $\lfloor t \rfloor$ is the largest integer not greater than t. In the first set of experiments, we fixed the number of classes c to the original number shown above, and changed the number of training samples as $n = 100, 200, 500, 1000, 2000$. In the second set of experiments, we fixed the number of training samples to $n = 1000$, and changed the number of classes c—samples only in the first c classes in the dataset are used. The classification accuracy is evaluated using 100 test samples randomly chosen from each class. The computation time is measured by the CPU computation time required for training each classifier when the Gaussian width and the regularization parameter chosen by cross-validation were used.

The experimental results are summarized in Figure 7 and Figure 8. The left column in Figure 7 shows that when n is increased, the misclassification error for all the methods tends to decrease, and LSPC, LSPC(full), and KLR performed similarly well. The right column in Figure 7 shows that when n is increased, the computation time tends to grow for all the methods. LSPC is faster than KLR by two orders of magnitude. The left column in Figure 8 shows that when c is increased, the misclassification error tends to increase for all the methods, and LSPC, LSPC(full), and KLR behaved similarly well. The right column in Figure 8 shows that when c is increased, the computation time of KLR tends to grow, while that of LSPC is kept constant or even it tends to slightly decrease. This happened because the number of samples in each class decreases when c is increased, and the computation time of LSPC is governed by the number of samples in *each* class, not by the *total* number of samples (see Section 2.3).

Overall, the computation of LSPC was shown to be faster than that of KLR by orders of magnitude, while LSPC and KLR were shown to be comparable to each other in terms of the classification accuracy. LSPC and LSPC(full) were shown to possess similar

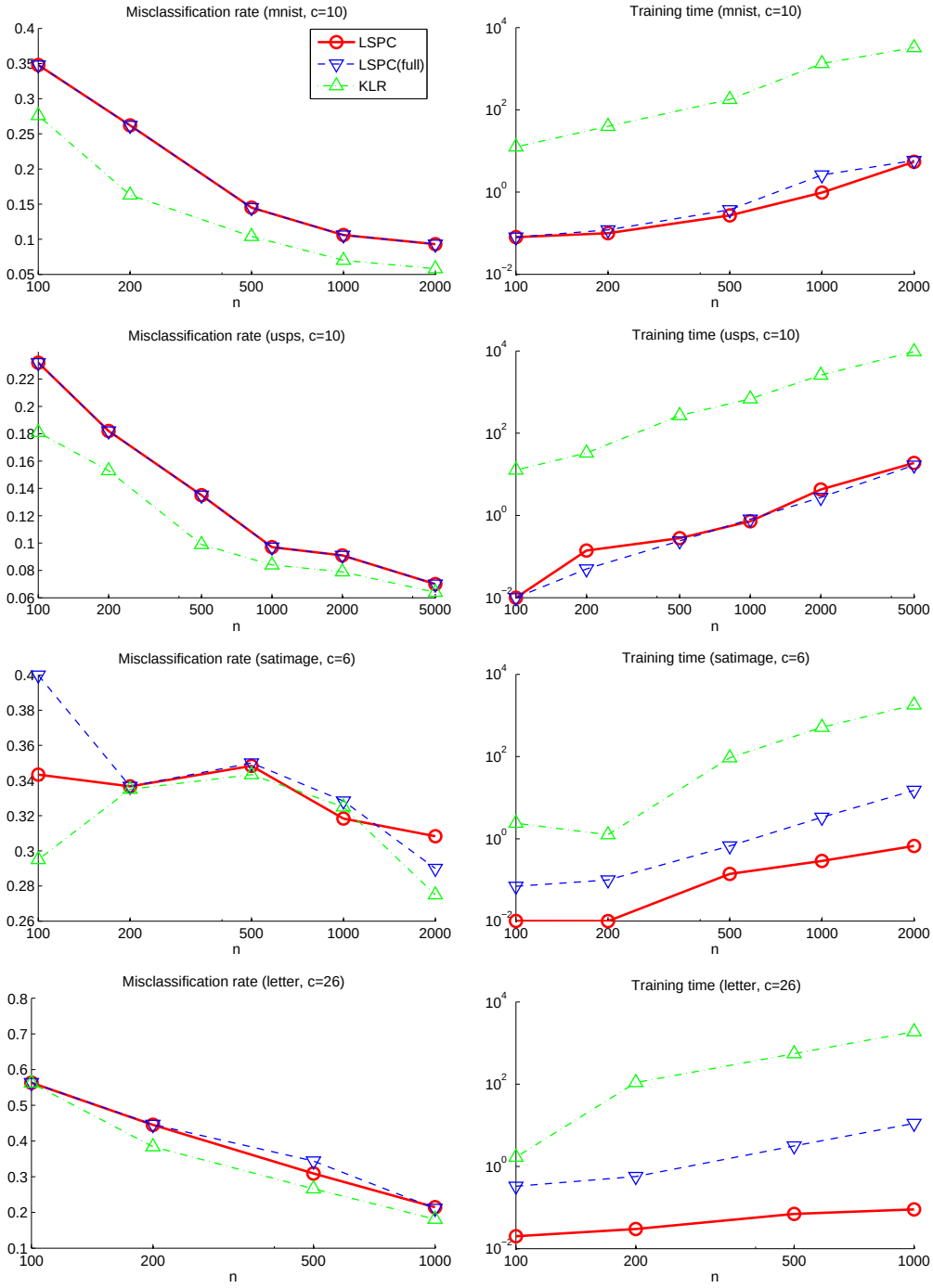


Figure 7: Misclassification rate (in percent, left) and computation time (in second, right) as functions of the number of training samples n. From top to bottom, the graphs correspond to the 'mnist', 'usps', 'satimage', and 'letter' datasets.

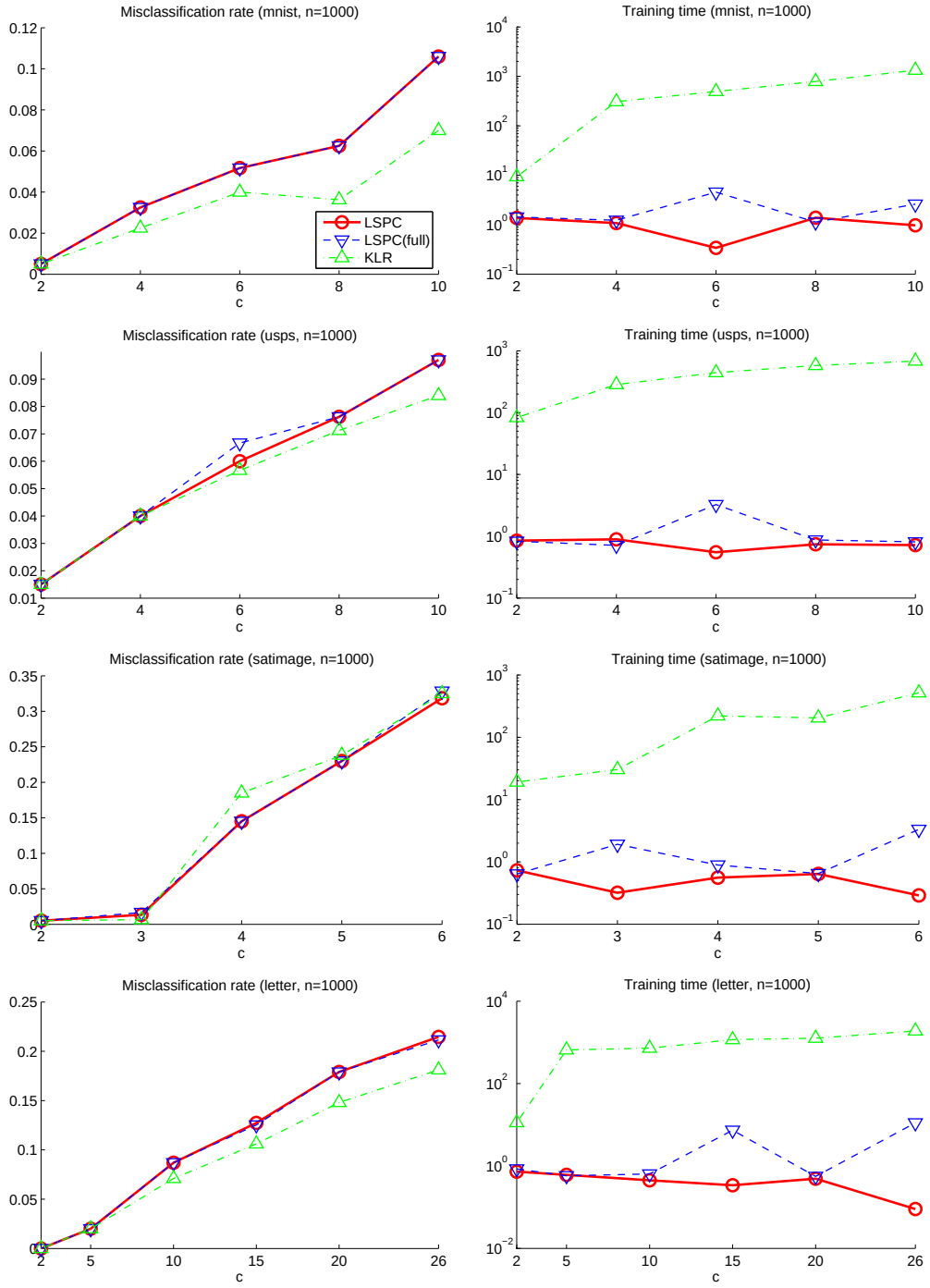


Figure 8: Misclassification rate (in percent, left) and computation time (in second, right) as functions of the number of classes c. From top to bottom, the graphs correspond to the 'mnist', 'usps', 'satimage', and 'letter' datasets.

classification performance, and thus a computationally efficient version, LSPC, would be more preferable in practice.

4 Discussion and Conclusion

Recently, various efficient algorithms for computing the solution of logistic regression have been developed for high-dimensional sparse data [22, 10]. However, for dense data, using standard non-linear optimization techniques such as Newton's method or quasi-Newton methods seem to be a common choice [15, 23]. The performance of these general-purpose non-linear optimizers has been improved in the last decade, but computing the solution of logistic regression for a large number of dense training samples is still a challenge problem.

In this paper, we proposed a simple probabilistic classification algorithm called *Least-Squares Probabilistic Classifier (LSPC)*. LSPC employs a linear combination of Gaussian kernels centered at training points for modeling the class-posterior probability and the parameters are learned by least-squares. Notable advantages of LSPC are that its solution can be computed *analytically* just by solving a system of linear equations and training can be carried out separately in a class-wise manner. In experiments, we showed that LSPC is faster than kernel logistic regression (KLR) in computation time by two orders of magnitude, with comparable accuracy.

The computational efficiency of LSPC was brought by the combination of appropriate model choice and loss function. More specifically, KLR uses a log-linear combination of kernel functions and its parameters are learned by regularized maximum likelihood. In this log-linear maximum likelihood formulation, normalization of the model is essential to avoid the likelihood diverging to infinity. Thus the likelihood function tends to be complicated and numerically solving the optimization problem may be unavoidable. On the other hand, in LSPC, we chose a linear combination of Gaussian kernel functions for modeling the class-posterior probability and its parameters are learned by regularized least-squares. This combination allowed us to obtain the solution analytically. When Newton's method (more specifically, *iteratively reweighted least-squares*) is used for learning the KLR model, a system of linear equations needs to be solved in *every* iteration until convergence [15]. On the other hand, LSPC requires to solve a system of linear equations only once.

We chose to separate the kernel for inputs and outputs, and adopted the delta kernel for outputs (see Eq.(5)). This allowed us to perform the training of LSPC in a class-wise manner. We showed that this contributes to reducing the training time particularly in multi-class classification problems. We note that this model choice is essentially the same as that of KLR[2].

We further proposed to reduce the number of kernels when "localized" kernels such as the Gaussian kernel (6) is used. Through the experimental evaluation in Section 3, we found that this heuristic model simplification does not degrade the classification accuracy,

[2]The number of parameters in LSPC with model (5) is cn, while the number of parameters in KLR is $(c-1)n$ since the normalization ('sum-to-one') constraint is incorporated in the training phase.

but reduces the computation time.

It is straightforward to show that solutions for *all* regularization parameter values (i.e., the *regularization path*, see [9, 14]) can be computed efficiently in LSPC. Let us consider the eigendecomposition of the matrix $\widehat{\boldsymbol{H}}^{(y)}$ (see Eq.(9)):

$$\widehat{\boldsymbol{H}}^{(y)} = \sum_{\ell=1}^{n_y} \gamma_\ell \boldsymbol{\psi}_\ell \boldsymbol{\psi}_\ell^\top,$$

where $\{\boldsymbol{\psi}_\ell\}_{\ell=1}^{n_y}$ are the eigenvectors of $\widehat{\boldsymbol{H}}^{(y)}$ associated with the eigenvalues $\{\gamma_\ell\}_{\ell=1}^{n_y}$. Then, the solution $\widehat{\boldsymbol{\alpha}}^{(y)}$ can be expressed as

$$\widehat{\boldsymbol{\alpha}}^{(y)} = (\widehat{\boldsymbol{H}}^{(y)} + \lambda \boldsymbol{I}_{n_y})^{-1} \widehat{\boldsymbol{h}}^{(y)} = \sum_{\ell=1}^{n_y} \frac{\widehat{\boldsymbol{h}}^\top \boldsymbol{\psi}_\ell}{\gamma_\ell + \lambda} \boldsymbol{\psi}_\ell.$$

Since $(\widehat{\boldsymbol{h}}^\top \boldsymbol{\psi}_\ell)\boldsymbol{\psi}_\ell$ is common to all λ, we can compute the solution $\widehat{\boldsymbol{\alpha}}^{(y)}$ for all λ efficiently by eigendecomposing the matrix $\widehat{\boldsymbol{H}}^{(y)}$ *once* in advance. Although eigendecomposition of $\widehat{\boldsymbol{H}}^{(y)}$ may be computationally slightly more demanding than solving a system of linear equations of the same size, this approach would be useful, e.g., when computing the solutions for various values of λ in the cross-validation procedure.

When n_y is large, we may further reduce the computational cost and memory space by using only a subset of kernels.

$$\begin{aligned} q(y|\boldsymbol{x};\boldsymbol{\alpha}) &= \sum_{\ell=1}^{b_y} \alpha_\ell^{(y)} K(\boldsymbol{x}, \boldsymbol{c}_\ell^{(y)}), \\ \widehat{H}_{\ell,\ell'}^{(y)} &= \frac{1}{n} \sum_{i=1}^{n} K(\boldsymbol{x}_i, \boldsymbol{c}_\ell^{(y)}) K(\boldsymbol{x}_i, \boldsymbol{c}_{\ell'}^{(y)}), \\ \widehat{h}_\ell^{(y)} &= \frac{1}{n} \sum_{i=1}^{n_y} K(\boldsymbol{x}_i^{(y)}, \boldsymbol{c}_\ell^{(y)}), \end{aligned}$$

where b_y is a constant chosen to be smaller than n_y and $\{\boldsymbol{c}_\ell^{(y)}\}_{\ell=1}^{b_y}$ is a subset of $\{\boldsymbol{x}_\ell^{(y)}\}_{\ell=1}^{n_y}$. This would be a useful heuristic when a huge number of samples are used for training.

Another option for reducing the computation time when the number of samples is very large would be the *stochastic gradient descent* method [1]. That is, starting from some initial parameter value, gradient descent is carried out only for a randomly chosen single sample in each iteration. Since our optimization problem is convex, convergence to the global solution is guaranteed (in a probabilistic sense) by stochastic gradient descent.

We focused on using the delta kernel for class labels (see Section 2.3). We expect that designing appropriate kernel functions for class labels would be useful for improving the classification performance, e.g., in the context of *multi-task learning* [4, 2, 21]. We will pursue this direction in our future work.

Acknowledgments

The author thanks Dr. Ryota Tomioka, Mr. Jaak Simm, and Dr. Hirotaka Hachiya for their valuable comments. This work was supported by AOARD, SCAT, and the JST PRESTO program.

References

[1] S. Amari, "Theory of adaptive pattern classifiers," IEEE Transactions on Electronic Computers, vol.EC-16, no.3, pp.299–307, 1967.

[2] B. Bakker and T. Heskes, "Task clustering and gating for Bayesian multitask learning," Journal of Machine Learning Research, vol.4, pp.83–99, 2003.

[3] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, New York, NY, USA, 2006.

[4] R. Caruana, L. Pratt, and S. Thrun, "Multitask learning," Machine Learning, vol.28, pp.41–75, 1997.

[5] C.C. Chang and C.J. Lin, "LIBSVM: A library for support vector machines," tech. rep., Department of Computer Science, National Taiwan University, 2001. `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

[6] R. Collobert and S. Bengio., "SVMTorch: Support vector machines for large-scale regression problems," Journal of Machine Learning Research, vol.1, pp.143–160, 2001.

[7] C. Cortes and V. Vapnik, "Support vector networks," Machine Learning, vol.20, pp.273–297, 1995.

[8] R.O. Duda, P.E. Hart, and D.G. Stor, Pattern Classification, Wiley, New York, 2001.

[9] B. Efron, T. Hastie, R. Tibshirani, and I. Johnstone, "Least angle regression," The Annals of Statistics, vol.32, no.2, pp.407–499, 2004.

[10] R.E. Fan, K.W. Chang, C.J. Hsieh, X.R. Wang, and C.J. Lin, "LIBLINEAR: A library for large linear classification," Journal of Machine Learning Research, vol.9, pp.1871–1874, 2008.

[11] R.E. Fan, P.H. Chen, and C.J. Lin, "Working set selection using second order information for training SVM," Journal of Machine Learning Research, vol.6, pp.1889–1918, 2005.

[12] V. Franc and S. Sonnenburg, "Optimized cutting plane algorithm for large-scale risk minimization," Journal of Machine Learning Research, vol.10, pp.2157–2192, 2009.

[13] G.M. Fung and O.L. Mangasarian, "Multicategory proximal support vector machine classifiers," Machine Learning, vol.59, no.1–2, pp.77–97, 2005.

[14] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu, "The entire regularization path for the support vector machine," Journal of Machine Learning Research, vol.5, pp.1391–1415, 2004.

[15] T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer, New York, 2001.

[16] T. Joachims, "Making large-scale SVM learning practical," in Advances in Kernel Methods—Support Vector Learning, ed. B. Schölkopf, C.J.C. Burges, and A.J. Smola, pp.169–184, The MIT Press, Cambridge, MA, 1999.

[17] T. Joachims, "Training linear SVMs in linear time," ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2006), pp.217–226, 2006.

[18] T. Kanamori, S. Hido, and M. Sugiyama, "A least-squares approach to direct importance estimation," Journal of Machine Learning Research, vol.10, pp.1391–1445, Jul. 2009.

[19] T. Kanamori, T. Suzuki, and M. Sugiyama, "Condition number analysis of kernel-based density ratio estimation," tech. rep., arXiv, 2009.

[20] T. Kanamori, T. Suzuki, and M. Sugiyama, "Theoretical analysis of density ratio estimation," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol.E93-A, no.4, pp.787–798, 2010.

[21] T. Kato, H. Kashima, M. Sugiyama, and K. Asai, "Conic programming for multi-task learning," IEEE Transactions on Knowledge and Data Engineering, vol.22, no.7, pp.957–968, 2010.

[22] K. Koh, S.J. Kim, and S.P. Boyd, "An interior-point method for large-scale l_1-regularized logistic regression," Journal of Machine Learning Research, vol.8, pp.1519–1555, 2007.

[23] T.P. Minka, "A comparison of numerical optimizers for logistic regression," tech. rep., Microsoft Research, 2007.

[24] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in Advances in Kernel Methods—Support Vector Learning, ed. B. Schölkopf, C.J.C. Burges, and A.J. Smola, pp.169–184, The MIT Press, Cambridge, MA, 1999.

[25] J. Platt, "Probabilities for SV machines," in Advances in Large Margin Classifiers, ed. A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, The MIT Press, Cambridge, MA, 2000.

[26] R. Rifkin, G. Yeo, and T. Poggio, "Regularized least-squares classification," Advances in Learning Theory: Methods, Models and Applications, ed. J.A.K. Suykens, G. Horvath, S. Basu, C. Micchelli, and J. Vandewalle, NATO Science Series III: Computer & Systems Sciences, vol.190, Amsterdam, the Netherlands, pp.131–154, IOS Press, 2003.

[27] M. Schmidt, minFunc, 2005. `http://people.cs.ubc.ca/~schmidtm/Software/minFunc.html`.

[28] B. Schölkopf and A.J. Smola, Learning with Kernels, MIT Press, Cambridge, MA, 2002.

[29] J.A.K. Suykens, T.V. Gestel, J.D. Brabanter, B.D. Moor, and J. Vandewalle, Least Squares Support Vector Machines, World Scientific Pub. Co., Singapore, 2002.

[30] Y. Tang and H.H. Zhang, "Multiclass proximal support vector machines," Journal of Computational and Graphical Statistics, vol.15, no.2, pp.339–355, 2006.

[31] C.H. Teo, Q. Le, A. Smola, and S.V.N. Vishwanathan, "A scalable modular convex solver for regularized risk minimization," ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2007), pp.727–736, 2007.

[32] I. Tsang, J. Kwok, and P.M. Cheung, "Core vector machines: Fast SVM training on very large data sets," Journal of Machine Learning Research, vol.6, pp.363–392, 2005.

[33] V.N. Vapnik, Statistical Learning Theory, Wiley, New York, NY, USA, 1998.

[34] T.F. Wu, C.J. Lin, and R.C. Weng, "Probability estimates for multi-class classification by pairwise coupling," Journal of Machine Learning Research, vol.5, pp.975–1005, 2004.

[35] M. Yamada, M. Sugiyama, G. Wichern, and J. Simm, "Improving the accuracy of least-squares probabilistic classifiers," IEICE Transactions on Information and Systems, 2011. submitted.