IEICE Transactions on Information and Systems,
vol.E92-D, no.7, pp.1338–1353, 2009.

Recent Advances and Trends in Large-scale Kernel Methods

Hisashi Kashima (`hkashima@jp.ibm.com`)
IBM Research, Tokyo Research Laboratory

Tsuyoshi Idé (`goodidea@jp.ibm.com`)
IBM Research, Tokyo Research Laboratory

Tsuyoshi Kato (`kato-tsuyoshi@aist.go.jp`)
Center for Informational Biology, Ochanomizu University

Masashi Sugiyama (`sugi@cs.titech.ac.jp`)
Department of Computer Science, Tokyo Institute of Technology

Abstract

Kernel methods such as the support vector machine are one of the most successful algorithms in modern machine learning. Their advantage is that linear algorithms are extended to non-linear scenarios in a straightforward way by the use of the kernel trick. However, naive use of kernel methods is computationally expensive since the computational complexity typically scales cubically with respect to the number of training samples. In this article, we review recent advances in the kernel methods, with emphasis on scalability for massive problems.

Keywords

Kernel methods, Support vector machines, Kernel trick, Low-rank approximation, Optimization, Structured data

1 Introduction

Kernel methods are a family of machine learning algorithms that can handle non-linear models as if they are linear models. The key technique is the use of similarity functions called *kernel functions*. Thanks to this, the kernel methods possess favorable characteristics, and in particular, their computational complexity is independent of the dimensionality of the feature space.

One of the most famous kernel methods is the *support vector machine* (SVM) [1]. Since Vapnik's seminal work, many linear supervised and unsupervised learning algorithms have been kernelized [2, 3], including ridge regression, perceptrons, Fisher discriminant analysis, principal component analysis (PCA), k-means clustering, and independent component analysis (ICA). These kernelized algorithms have been shown to perform very well in many real-world problems. However, naive use of these kernel methods is computationally expensive since the computational complexity typically scales cubically with respect to the number of training samples.

Recently, considerable effort has been devoted to improving the computational efficiency of the kernel methods [4]. The purpose of this paper is to review various techniques for accelerating the kernel methods. After briefly reviewing the basic ideas of kernel methods in Section 2, we review recent advances in the kernel methods for large scale problems. One of the computational bottlenecks of the kernel methods is the computation of the kernel matrix. In Section 3, we review how the kernel matrix can be approximated by low-rank matrices, considering incomplete Cholesky decomposition, the Lanczos approximation, the Nyström method, and the fast Gaussian transform. In Section 4, we review supervised kernel methods. For regression problems, we show how the low-rank approximation techniques can be used to speed up the kernel algorithms such as ridge regression, partial least-squares, and Gaussian processes. For classification problems, we show how the optimization problem involved in the support vector algorithm can be solved efficiently by using the cutting-plane or dual coordinate descent techniques. In Section 5, we focus on unsupervised methods and show how the dimensionality of *sparse* data can be efficiently reduced by the Laplacian eigenmap. This can be employed naturally in the spectral clustering algorithm. In Section 6, we address how the kernel functions themselves can be computed efficiently when dealing with structured data. Section 6 also considers how to learn a kernel from multiple kernels. Finally we conclude in Section 7.

2 Basics of Kernel Methods

In order to briefly explain how linear algorithms can be non-linearized by using kernel functions, let us consider the linear parametric model

$$f(\boldsymbol{x};\boldsymbol{w}) \equiv \langle \boldsymbol{w}, \boldsymbol{x} \rangle, \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^d$ is an input variable, $\boldsymbol{w} \in \mathbb{R}^d$ is a parameter vector, and $\langle \cdot, \cdot \rangle$ denotes the inner product. Given input-output training data $\{(\boldsymbol{x}_i, y_i) \,|\, \boldsymbol{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}\}_{i=1}^{\ell}$, the parameter $\boldsymbol{w}$ is determined so that the linear sum of an empirical risk term (or a "goodness-of-fit" term for the training data) and a regularization term is minimized:

$$J(\boldsymbol{w}) \equiv R_{\text{emp}}(\boldsymbol{w}) + \lambda \Omega(\boldsymbol{w}),$$

where $\lambda > 0$ is the regularization parameter that controls the balance between goodness-of-fit and regularization. For ridge regression, the squared loss function is used for measuring the risk and the squared regularization function is adopted as the regularizer:

$$R_{\mathrm{emp}}(\boldsymbol{w}) \equiv \sum_{i=1}^{\ell} \left(y_i - f(\boldsymbol{x}_i;\boldsymbol{w})\right)^2, \tag{2}$$

$$\Omega(\boldsymbol{w}) \equiv \|\boldsymbol{w}\|_2^2, \tag{3}$$

where $\|\cdot\|_2$ denotes the 2-norm. We denote the matrix consisting of all input samples by

$$\boldsymbol{X} = (\boldsymbol{x}_1|\boldsymbol{x}_2|\cdots|\boldsymbol{x}_\ell).$$

Then the ridge regression solution is given analytically as

$$\widehat{\boldsymbol{w}} = (\boldsymbol{X}^\top\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^\top\boldsymbol{y}, \tag{4}$$

where $\boldsymbol{I}$ denotes the identity matrix, $^\top$ indicates the transpose, and

$$\boldsymbol{y} \equiv (y_1, y_2, \ldots, y_\ell)^\top.$$

This implies that the computational complexity of computing the ridge regression solution is $O(d^3)$ and that it explicitly depends on the input dimensionality.

Suppose the parameter $\boldsymbol{w}$ is written as the linear combination of the training samples $\{\boldsymbol{x}_i\}_{i=1}^{\ell}$ as

$$\boldsymbol{w} \equiv \sum_{i=1}^{\ell} \alpha_i \boldsymbol{x}_i = \boldsymbol{X}\boldsymbol{\alpha}, \tag{5}$$

where

$$\boldsymbol{\alpha} \equiv (\alpha_1, \alpha_2, \ldots, \alpha_\ell)^\top$$

is an ℓ-dimensional vector of the parameters. Then Eq.(1) can be expressed using the inner product as

$$f(\boldsymbol{x};\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i \langle \boldsymbol{x}_i, \boldsymbol{x}\rangle.$$

Now let us define the *kernel function* as

$$k(\boldsymbol{x},\boldsymbol{x}') \equiv \langle \boldsymbol{x}, \boldsymbol{x}'\rangle.$$

Then the parametric model is expressed as

$$f(\boldsymbol{x}) = \sum_{i=1}^{\ell} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}). \tag{6}$$

With this kernelized model, the training objective function of ridge regression can be rewritten as

$$J(\boldsymbol{\alpha}) = \|\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}\|^2 + \lambda\boldsymbol{\alpha}^\top\boldsymbol{K}\boldsymbol{\alpha},$$

where $\boldsymbol{K}$ is the $\ell \times \ell$ matrix called the *kernel matrix* defined as

$$K_{i,j} \equiv k(\boldsymbol{x}_i, \boldsymbol{x}_j).$$

The solution $\widehat{\boldsymbol{\alpha}}$ can be obtained as

$$\widehat{\boldsymbol{\alpha}} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{y}. \tag{7}$$

This implies that the computational complexity is $O(d\ell^3)$, where the factor d comes from the computation of the kernel function values and the cubic factor ℓ^3 is the computational complexity of inverting the kernel matrix. Therefore, now the dependency of the computational complexity on the number ℓ of training samples is more significant than the input dimensionality d. Note that in this kernel formulation, the data samples are accessed only *through* the kernel functions both in the training and test phases; the input vectors are not directly handled.

The above formulation is based on the fact that the parameter vector $\boldsymbol{w}$ is expressed by a linear combination of training samples (see Eq.(5)). This can be justified by the *representer theorem* [5]. In order to highlight an advantage of the kernel formulation, let us consider the transformation

$$\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^{d'}.$$

We assume $d' \gg d$ and learning is carried out with the transformed samples $\{\boldsymbol{\phi}(x_i)\}_{i=1}^{\ell}$. In the primal formulation (4), computing the solution for $\{\boldsymbol{\phi}(x_i)\}_{i=1}^{\ell}$ may be intractable due to high dimensionality d'. In contrast, in the kernel formulation (7), the input samples are dealt with only through the kernel function evaluation

$$k(\boldsymbol{x}, \boldsymbol{x}') \equiv \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle.$$

Denoting by t the computational complexity for computing the kernel function value, we can compute the solution with computational complexity $O(t\ell^3)$. Thus if the kernel function can be computed efficiently (which means, independently of d'), the kernel formulation allows us to efficiently compute the solution even when d' is large. This computational trick is called the *kernel trick*.

Conversely, if there exists a kernel function which corresponds to the inner product between $\boldsymbol{\phi}(\boldsymbol{x})$ and $\boldsymbol{\phi}(\boldsymbol{x}')$, then the use of the kernel formulation would be beneficial. The existence of such an inner product is guaranteed when the kernel function is *positive semi-definite*. Such a kernel function is called a *Mercer kernel* or a *reproducing kernel* [6, 7]. There are many kernel functions whose computational complexity is independent of d'. A typical example would be the *polynomial kernel* (of order c):

$$k(\boldsymbol{x}, \boldsymbol{x}') = \left(\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + 1\right)^c.$$

The dimensionality d' of the polynomial feature $\boldsymbol{\phi}(\boldsymbol{x})$ will be very large if c is large. However, the value of the polynomial kernel can still be computed with computational complexity $O(d)$, which is independent of d'. A more striking example is the *Gaussian kernel* (with width σ),

$$k(\boldsymbol{x}, \boldsymbol{x}') = \mathrm{e}^{-\|\boldsymbol{x} - \boldsymbol{x}'\|^2/\sigma^2}.$$

For the Gaussian kernel, the dimensionality d' is actually infinite. Therefore, computing the solution within the primal formulation of Eq.(4) is no longer possible. In contrast, the kernel formulation of Eq.(7) still allows us to compute the solution.

Through the kernel trick, non-linear learning can be performed without regard to the input dimensionality. However, it still greatly depends on the number of training samples. The ridge regression described above requires the computational complexity of $O(\ell^3)$. Thus, when the number of training samples is large, the kernel methods are still computationally expensive.

3 Kernel Matrix Approximation

The learning algorithms of kernel machines are generally designed so that they do not depend on the dimensionality of the feature space. Given an efficient way to compute the kernel function itself, most of the kernel methods require $O(\ell^3)$ cost in naive implementations. However, this can be prohibitive when ℓ is large. This difficulty leads us to the idea of first performing feature extraction to produce a relatively small number of features based on the kernel matrix and then tackling the learning task using the extracted features. If the extracted features contain rich information on the nonlinearities of the data, the simple linear learning algorithms will be useful for the learning tasks.

In this section, we review various approaches to building a low-dimension feature space through feature extraction.

3.1 Spectral Decomposition

Perhaps the most direct method for nonlinear feature extraction is spectral decomposition. Any real symmetric matrix $\boldsymbol{A} \in \mathbb{S}^\ell$ can be expanded with eigenvectors as

$$\boldsymbol{A} = \sum_{i=1}^{\ell} \mu^{(i)} \boldsymbol{u}^{(i)} \boldsymbol{u}^{(i)\top},$$

where $\mu^{(i)}$ and $\boldsymbol{u}^{(i)}$ are the i-th eigenvalue and the i-th eigenvector of $\boldsymbol{A}$, respectively. We assume that the eigenvalues are always sorted in descending order. This type of expansion is called the *spectral expansion* [8]. If $\boldsymbol{A}$ is positive definite, it is easy to see that this expansion can be written as

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{U}^\top,$$

where

$$\boldsymbol{U} \equiv (\sqrt{\mu^{(1)}}\boldsymbol{u}^{(1)} | \sqrt{\mu^{(2)}}\boldsymbol{u}^{(2)} | \ldots | \sqrt{\mu^{(\ell)}}\boldsymbol{u}^{(\ell)}).$$

If we regard $\boldsymbol{A}$ as the kernel matrix $\boldsymbol{K}$, by Mercer's Theorem there exists a basis function $\boldsymbol{\phi}$ such that $K_{i,j} = \langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j) \rangle$. In matrix notation, we have

$$\boldsymbol{K} = \boldsymbol{\Phi}^\top \boldsymbol{\Phi}, \tag{8}$$

where $\mathbf{\Phi} \equiv (\boldsymbol{\phi}(\boldsymbol{x}_1)|\boldsymbol{\phi}(\boldsymbol{x}_2)|\ldots|\boldsymbol{\phi}(\boldsymbol{x}_\ell))$. This means that each column vector of $\boldsymbol{U}^\top$ corresponds to a feature vector.

If we use the kernel PCA [9], the small eigenvalues will correspond to noise, and the larger eigenvalues will dominate the distribution of the data. This leads us to an approximation using the first r terms of the expansion. In this case, the feature vectors are represented with $\boldsymbol{u}^{(1)}, \ldots, \boldsymbol{u}^{(r)}$. If one uses an iterative method such as the power method [10], the computational cost of this approximation is $O(r\ell^2)$ if the power iteration is terminated witha finite number of repetitions.

3.2 Incomplete Cholesky Decomposition

Another useful method for extracting feature vectors in the feature space is *incomplete Cholesky decomposition*. Let us look at the original Cholesky decomposition first. Cholesky decomposition factorizes a positive definite matrix into the product of lower triangular matrices. For a kernel matrix $\boldsymbol{K}$, the decomposition is written as

$$\boldsymbol{K} = \boldsymbol{L}\boldsymbol{L}^\top, \tag{9}$$

where $\boldsymbol{L}$ is a lower triangular matrix in which $L_{i,j} = 0$ for $i < j$. Note that Cholesky factorization amounts to implicitly finding the basis functions of the kernel. Comparing Eqs. (8) and (9), we see that $\boldsymbol{L}$ corresponds to the data matrix $\mathbf{\Phi}^\top$ up to an arbitrary orthogonal matrix. This means that Cholesky factorization essentially finds the basis function apart from the arbitrary orthogonal matrix. Cholesky factorization is quite useful not only for numerical efficiency but also for theoretical interpretation.

The numerical computation of a Cholesky factorization is very easy. Direct element-wise comparison between both sides of Eq.(9) leads to

$$L_{i,i} = \sqrt{K_{i,i} - \sum_{k=1}^{i-1} L_{i,k}^2},$$

$$L_{i,j} = \frac{1}{L_{i,i}} \left[K_{j,i} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k} \right]$$

for $j = i+1, i+2, \ldots, \ell$, from which the entries of $\boldsymbol{L}$ are computed. Note that $\boldsymbol{L}^{-1}$ and thus $\boldsymbol{K}^{-1}$ can be easily computed once $\boldsymbol{L}$ has been obtained [10]. As suggested by these simple equations, Cholesky factorization is numerically very stable, although it requires $\ell^3/6$ operations.

The incomplete Cholesky decomposition [11, 12] truncates this procedure at a pre-determined number $\tilde{d} < \ell$, which amounts to explicitly generating a $\tilde{d}$-dimensional feature space. At each step of incomplete Cholesky decomposition, one column of the matrix is chosen and added so that the truncation error is minimized for the current number of columns. It is important to note that we do not need an $\ell \times \ell$ memory space to store the Gram matrix, and that only the diagonal elements are needed to compute the off-diagonal elements. The computational cost is only $O(\tilde{d}^2\ell)$, which means that the computation can be carried out without accessing all of the elements. Although incomplete Cholesky factorization is less popular than the original Cholesky factorization, good performance has been reported in an SVM classification task [11] and kernel ICA [12].

3.3 Tridiagonalization

In this subsection, we consider a feature extraction method via tridiagonalization. *Tridiagonalization* is an orthogonal transformation of a symmetric matrix $\boldsymbol{A} \in \mathbb{S}^\ell$ such that

$$\boldsymbol{Q}^\top \boldsymbol{A}\boldsymbol{Q} = \boldsymbol{T}, \tag{10}$$

where $\boldsymbol{Q}$ and $\boldsymbol{T}$ are orthogonal and tridiagonal matrices, respectively. A symmetric tridiagonal matrix is a matrix whose entries are all zero except for diagonal and subdiagonal elements, i.e.,

$$\boldsymbol{T} = \begin{pmatrix} \gamma_1 & \beta_1 & & & \Large 0 \\ \beta_1 & \gamma_2 & \beta_2 & & \\ & \beta_2 & \ddots & \ddots & \\ & & \ddots & \gamma_{\ell-1} & \beta_{\ell-1} \\ \Large 0 & & & \beta_{\ell-1} & \gamma_\ell \end{pmatrix}.$$

By setting the number of columns in $\boldsymbol{Q}$ to be smaller than ℓ, this transformation can be viewed as a low-rank approximation of $\boldsymbol{A}$. Note that tridiagonalization is not unique. The key fact is that there exists an algorithm which allows us to find an optimal $\boldsymbol{Q}$ in that $\boldsymbol{T}$ reproduces extremal eigenvalues such as the largest and smallest eigenvalues with high precision. This is in contrast to the spectral low-rank approximation, where smaller eigenvalues are simply truncated. This subsection focuses on a useful tridiagonalization algorithm called the Lanczos algorithm, and briefly discusses its connection to the well-known *conjugate gradient (CG)* algorithm.

The derivation of the Lanczos algorithm is very simple. Consider a tridiagonal transformation as in Eq.(10). Let $\gamma_1, \dots, \gamma_\ell$ and $\beta_1, \dots, \beta_{\ell-1}$ be the diagonal and subdiagonal elements of $\boldsymbol{T}$, respectively. If we directly record the s-th column of the equation $\boldsymbol{A}\boldsymbol{Q} = \boldsymbol{Q}\boldsymbol{T}$, we have

$$\boldsymbol{A}\boldsymbol{q}_s = \gamma_s \boldsymbol{q}_s + \beta_{s-1}\boldsymbol{q}_{s-1} + \beta_s \boldsymbol{q}_{s+1},$$

where $\boldsymbol{q}_s$ is the s-th column vector of $\boldsymbol{Q}$. Using the orthogonal relation $\boldsymbol{q}_i^\top \boldsymbol{q}_j = \delta_{i,j}$, we immediately have $\gamma_s = \boldsymbol{q}_s{}^\top \boldsymbol{A}\boldsymbol{q}_s$. In this way, it is easy to construct an algorithm to find γ_s and β_s sequentially from this recurrent equation:

```
1: Input $\boldsymbol{r}_0$.
2: $\beta_0 \leftarrow \|\boldsymbol{r}_0\|$;
3: repeat
4:     $\boldsymbol{q}_{s+1} \leftarrow \boldsymbol{r}_s/\beta_s$;
5:     $s \leftarrow s + 1$;
```

```
 6:     $\gamma_s \leftarrow \boldsymbol{q}_s{}^\top \boldsymbol{A}\boldsymbol{q}_s$;
 7:     $\boldsymbol{r}_s \leftarrow \boldsymbol{A}\boldsymbol{q}_s - \gamma_s\boldsymbol{q}_s - \beta_{s-1}\boldsymbol{q}_{s-1}$;
 8:     $\beta_s \leftarrow \|\boldsymbol{r}_s\|$;
 9: **until** $s = \ell$;
10: return $\{\gamma_i\}$, $\{\beta_i\}$, and $\{\boldsymbol{q}_i\}$.
```

This procedure needs an initial vector $\boldsymbol{r}_0$ and its norm $\beta_0 = \|\boldsymbol{r}_0\|$. This tridiagonalization procedure is called the *Lanczos algorithm* [13].

By running this procedure up to $r < \ell$, we obtain an $r \times r$ tridiagonalized matrix $\boldsymbol{T}_r$ with calculated $\{\gamma_i\}$ and $\{\beta_i\}$. This matrix can be thought of as a low-rank approximation of the original matrix, but its nature is quite different from the spectral low-rank approximations in that the spectrum of $\boldsymbol{T}_r$ reproduces the extremal eigenvalues to a very high accuracy [13].

One useful application of the Lanczos algorithm is (kernel) PCA. As shown in [14], the eigenspace projection of a given vector can be computed directly from the eigenvectors of the truncated tridiagonalized matrix $\boldsymbol{T}_r$. The computational cost to diagonalize $\boldsymbol{T}_r$ is only $O(r)$ when $r \ll \ell$. A similar idea is used in [15] to accelerate a Markov decision process.

The Lanczos algorithm has a close relationship with the CG method for solving linear equations. Consider a linear equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, for $\boldsymbol{A} \in \mathbb{S}_+^\ell$. In the CG algorithm, we first initialize a residual vector $\boldsymbol{r}_0$ as $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$ using an initial estimate of the solution $\boldsymbol{x}_0$. Then we repeat the following procedure until the residual $\boldsymbol{r}_{s+1}$ is small enough.

```
 1: Input: $\boldsymbol{x}_0$, $\epsilon$
 2: $\boldsymbol{r}_0 \leftarrow \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$;
 3: **repeat**
 4:     $\gamma_s \leftarrow \|\boldsymbol{r}_s\|^2/(\boldsymbol{p}_s^\top \boldsymbol{A}\boldsymbol{p}_s)$;
 5:     $\boldsymbol{x}_{s+1} \leftarrow \boldsymbol{x}_s + \gamma_s\boldsymbol{p}_s$;
 6:     $\boldsymbol{r}_{s+1} \leftarrow \boldsymbol{r}_s - \gamma_s\boldsymbol{A}\boldsymbol{p}_s$;
 7:     $\beta_s \leftarrow \|\boldsymbol{r}_{s+1}\|^2/\|\boldsymbol{r}_s\|^2$;
 8:     $\boldsymbol{p}_{s+1} \leftarrow \boldsymbol{r}_{s+1} - \beta_s\boldsymbol{p}_s$;
 9:     $s \leftarrow s + 1$;
10: **until** $\|\boldsymbol{r}_{s+1}\| < \epsilon$;
11: return the solution $\boldsymbol{x}_{s+1}$.
```

This algorithm solves $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ by iteratively minimizing

$$\frac{1}{2}\boldsymbol{x}^\top\boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^\top\boldsymbol{b}.$$

One major advantage of CG is that it gives a very accurate approximate solution with the number of iterations far smaller than the size of the matrix. It can be shown that the residual vectors generated in the CG procedure are essentially the same as the Lanczos vectors $\{\boldsymbol{q}_i\}$ (for a proof, see Section 10.2 in [13]), which means that the excellent convergence properties of CG are due to the low-rank approximation by Lanczos tridiagonalization.

3.4 The Nyström Method

When ℓ is very large, directly handling the kernel matrix $\boldsymbol{K}$ is inconvenient. The *Nyström method* allows us to approximate $\boldsymbol{K}$ and its eigenvectors with a smaller kernel matrix of a subset of the samples.

Suppose we have a data set $\mathcal{D} = \{\boldsymbol{x}_i | i = 1, \ldots, \ell\}$ with a very large ℓ, and let the corresponding kernel matrix be $\boldsymbol{K} \in \mathbb{S}_+^\ell$. Without loss of generality, let a subset of $\mathcal{D}$ be $\tilde{\mathcal{D}} = \{\boldsymbol{x}_i | i = 1, \ldots, m\}$ with $m < \ell$. Corresponding to this, we partition $\boldsymbol{K}$ as

$$\boldsymbol{K} = \begin{pmatrix} \tilde{\boldsymbol{K}} & \boldsymbol{L}^\top \\ \boldsymbol{L} & * \end{pmatrix}, \quad \boldsymbol{C} \equiv \begin{pmatrix} \tilde{\boldsymbol{K}} \\ \boldsymbol{L} \end{pmatrix}, \tag{11}$$

where $\tilde{\boldsymbol{K}} \in \mathbb{S}_+^m$, and we also define a matrix $\boldsymbol{C} \in \mathbb{R}^{\ell \times m}$.

The Nyström approximation relates the eigenvalues and eigenvectors of $\boldsymbol{K}$ with those of $\tilde{\boldsymbol{K}}$ as (for a proof, see Section 8 in [16])

$$\mu^{(i)} \approx \frac{\ell}{m}\tilde{\mu}^{(i)} \text{ and } \boldsymbol{u}^{(i)} \approx \sqrt{\frac{m}{\ell}}\frac{1}{\tilde{\mu}^{(i)}}\boldsymbol{C}\tilde{\boldsymbol{u}}^{(i)}, \tag{12}$$

where $(\mu^{(i)}, \boldsymbol{u}^{(i)})$ are the i-th eigenvalue and eigenvector of $\boldsymbol{K}$, and $(\tilde{\mu}^{(i)}, \tilde{\boldsymbol{u}}^{(i)})$ are those of $\tilde{\boldsymbol{K}}$. Since $\tilde{\boldsymbol{K}}$ is an $m \times m$ matrix, i runs from 1 through m.

These expressions lead to a rank-m approximation of the kernel matrix $\boldsymbol{K}$ as

$$\boldsymbol{K} \approx \sum_{i=1}^{m} \mu^{(i)}\boldsymbol{u}^{(i)}{\boldsymbol{u}^{(i)}}^\top = \boldsymbol{C}\tilde{\boldsymbol{K}}^{-1}\boldsymbol{C}^\top,$$

where the final equality is easily verified by plugging in Eq.(12). This type of approximation was used in Gaussian process regression for large data sets [17]. However, applying the Nyström method to real-world problems is sometimes tricky, since it is extremely sensitive to the choice of the parameters (how to choose $\tilde{\mathcal{D}}$) [18].

3.5 The Fast Gauss Transform

In iterative algorithms such as CG, the computational cost is bounded by the cost of the matrix-vector products such as $\boldsymbol{K}\boldsymbol{\alpha}$, which scales as $O(\ell^2)$ in a naive implementation. If the kernel is Gaussian, the product is reduced to computing the Gaussian transform as

$$G(\boldsymbol{x}) \equiv \sum_{n=1}^{\ell} \alpha_n \mathrm{e}^{-\|\boldsymbol{x}_n - \boldsymbol{x}\|^2/\sigma^2}.$$

In this case, an algorithm called the *Fast Gauss Transform (FGT)* significantly reduces the computational costs [19]. The idea of the original FGT, which is a special case of multi-pole expansion, a standard technique in physics, is to utilize a truncated form (up to the first p terms) of the Hermite expansion of the Gaussian. When $d = 1$, the truncated Hermite expansion reads

$$\mathrm{e}^{-\|x-x_n\|^2/\sigma^2} \approx \sum_{s=0}^{p-1} \frac{1}{s!} \left(\frac{x_n - x_*}{\sigma} \right)^s h_s \left(\frac{x - x_*}{\sigma} \right),$$

where h_s is the Hermite polynomial of order s [20]. Notice that x and x_n are separated in each term by introducing the expansion center x_*, which is treated as a parameter determined from the data. By inserting this expansion, $G(x)$ can be written as

$$G(x) = \sum_{s=0}^{p-1} C_s h_s \left(\frac{x - x_*}{\sigma} \right),$$

where

$$C_s = \sum_{n=1}^{\ell} \frac{\alpha_n}{s!} \left(\frac{x_n - x_*}{\sigma} \right)^s.$$

Since C_s is independent of x, the computational cost for estimating $G(x)$ over the ℓ different locations is just $(\ell + \ell)p$, which is linear with respect to ℓ. When $d > 1$, the cost scales as $O(p^d \ell)$. The FGT is further accelerated by using the Taylor expansion and spatial partitioning [20, 21].

4 Supervised Methods

In this section, we briefly address computational issues of the supervised kernel methods.

4.1 Regression

We begin with the regression methods including kernel ridge regression [22], partial least squares [23], the Lasso [24], Gaussian process regression [24], relevance vector machines [25], and support vector regression [2].

4.1.1 Kernel Ridge Regression

Kernel Ridge Regression (KRR) (e.g., Section 2.2 in [22]) is a widely used regression technique that is useful even when $\boldsymbol{K}$ is numerically rank deficient. Solving KRR amounts to solving the following linear equation (cf. Eq.(7)):

$$(\boldsymbol{K} + \lambda \boldsymbol{I}_\ell)\boldsymbol{\alpha} = \boldsymbol{y},$$

where $\boldsymbol{I}_\ell$ denotes the ℓ-dimensional identity matrix. Since $\boldsymbol{K}$ is a dense matrix in general, a common approach to solving this equation is Cholesky factorization followed by forward-backward substitutions [10], which costs $O(\ell^3)$. If the CG method is used to solve this equation, the cost becomes $O(r\ell^2)$, where r is the number of CG iterations. Although the numerical stability of CG is not necessarily guaranteed for dense matrices due to the nature of the Krylov subspace [13], it generally works well in many practical cases. If the kernel function is Gaussian, the computational cost of CG can be reduced to $O(\ell)$, as mentioned in the previous section.

To determine the regularization parameter λ, a theoretically valid approach would be leave-one-out (LOO) cross validation (CV). In KRR, a closed-form solution of the LOOCV score is available as

$$g_{\mathrm{LOO}}(\lambda) = \|\boldsymbol{H}^{-1}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha})\|^2,$$

where

$$\boldsymbol{H} \equiv \boldsymbol{I}_N - \mathrm{diag}(\boldsymbol{K}(\boldsymbol{K} + \lambda\boldsymbol{I}_\ell)^{-1}),$$

and $\boldsymbol{\alpha}$ is the solution of KRR under λ in $\boldsymbol{H}$. The symbol 'diag' is an operator for setting all of the off-diagonal elements to zero. For a proof, see [5, 26] and use the Woodbury identity [2] to get a dual expression for $\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{X}^\top + \lambda\boldsymbol{I}_d)^{-1}\boldsymbol{X}$. To compute this, however, the inverse of $(\boldsymbol{K} + \lambda\boldsymbol{I}_\ell)$ is needed, resulting in $O(\ell^3)$ computational costs.

4.1.2 Partial Least Squares

In chemometrics, a class of heuristic iterative algorithms called *partial least squares (PLS)* is a popular regression technique. In recent years, many attempts have been made in the machine learning community to investigate and improve the PLS procedures, e.g., kernelized PLS [23]. For a general description on PLS, see, e.g., Section 6.7 in [3].

Although PLS is usually introduced as an iterative procedure without explicit objective functions, it was shown [27] that kernel PLS minimizes the same objective function as ordinary least squares, but the solution is restricted within a subspace spanned by $\boldsymbol{K}\boldsymbol{y}, \boldsymbol{K}^2\boldsymbol{y}, \ldots, \boldsymbol{K}^r\boldsymbol{y}$, where r is a given integer representing the number of PLS components. This type of subspace is called the Krylov subspace. It can be shown that the solution is found by solving

$$\boldsymbol{K}^2\boldsymbol{\alpha} = \boldsymbol{K}\boldsymbol{y}$$

using CG with the initial estimate of $\boldsymbol{\alpha} = \boldsymbol{0}$. To determine r, see [27] for a comparison among various criteria. In all kernelized PLS algorithms known so far, the computational cost is dominated by the matrix-vector products. While naive implementations require $O(\ell^2)$ computational cost, FGT reduces the cost to linear time when the kernel is Gaussian (see Section 3.5). The Lanczos approximation can also be used for speeding up kernel PLS [28].

Since kernel PLS gives a dense solution in general, great efforts have been made to sparsify PLS. For example, an ε-insensitive loss function is adopted in the objective function [29], a smaller Gram matrix via a random selection of samples is used [30], and a fixed threshold on $\boldsymbol{\alpha}$ is introduced to force many components to have the zero values in a graph regression task [31].

4.1.3 Lasso

The *least absolute shrinkage and selection operator (Lasso)* [24] is a shrinkage and selection method for linear regression. It minimizes the sum of a squared loss function (Eq.(2)) with the ℓ_1 regularizer

$$J_{\text{lasso}}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{\ell}(y_i - \langle \boldsymbol{x}_i, \boldsymbol{w}\rangle)^2 + \lambda\|\boldsymbol{w}\|_1, \tag{13}$$

where $\|\boldsymbol{w}\|_1 = \sum_{i=1}^{d}|w_i|$. The ℓ_1 regularizer allows us to automatically prune irrelevant features, so the Lasso solution tends to be sparse.

The lasso optimization problem can be reformulated as a linear program by doubling the number of parameters, so the solution can be obtained using a standard optimization software. Using the fact that the solution path of Lasso is piecewise linear with respect to λ, least angle regression (LARS) [32] allows us to obtain the Lasso solution for all λ (the technique is called *parametric optimization* [33]). However, when computing the solution for a single λ, a simple coordinate descent algorithm is much faster [34].

Let us briefly look at how the algorithm coordinate descent proposed in [34] works. Consider minimizing the convex objective function using Eq.(13) with respect to $\boldsymbol{w}$ using the subgradient method. Differentiating the objective function, we have

$$\begin{aligned}\frac{\partial g}{\partial w_i} &= \sum_m W_{i,m}w_m - s_i + \lambda\,\text{sign}(w_i)\\ &= W_{i,i}w_i + \sum_{m\neq i} W_{i,m}w_m - s_i + \lambda\,\text{sign}(w_i),\end{aligned}$$

where $\boldsymbol{W} \equiv \boldsymbol{X}\boldsymbol{X}^\top$ and $\boldsymbol{s} \equiv \boldsymbol{X}\boldsymbol{y}$. By considering the equation $\partial g/\partial w_i = 0$ separately for $w_i > 0$ and $w_i < 0$, it is straightforward to derive the following update rule:

$$w_i \leftarrow \begin{cases} (A_i - \lambda)/W_{i,i} & \text{for } A_i > \lambda,\\ 0 & \text{for } -\lambda < A_i < \lambda,\\ (A_i + \lambda)/W_{i,i} & \text{for } A_i < -\lambda,\end{cases} \tag{14}$$

for each $i = 1, \ldots, d$. Here we define

$$A_i \equiv s_i - \sum_{m\neq i} W_{i,m}w_m. \tag{15}$$

This algorithm is in particularly useful in structure learning since structure learning requires to solve a series of regression problems [35]. For recent work, see [36, 37].

Although the Lasso is a linear method, it can be kernelized in a straightforward way in [38, 39]. Some authors have focused on how to compute the regularization path of nonlinear versions of the Lasso [40].

4.1.4 Gaussian Process Regression

Gaussian Process Regression (GPR) is a popular Bayesian regression method due to its good generalization ability and simple closed-form equations to give the global solutions. The predictive mean (m) and variance (s^2) for a test input $\boldsymbol{x}$ are given as follows [16]:

$$\begin{aligned} m &= \boldsymbol{k}^\top \boldsymbol{C}^{-1}\boldsymbol{y}, \\ s^2 &= \sigma^2 + k(\boldsymbol{x},\boldsymbol{x}) - \boldsymbol{k}^\top \boldsymbol{C}^{-1}\boldsymbol{k}, \end{aligned} \tag{16}$$

where $\boldsymbol{k}$ and $\boldsymbol{C}$ are defined as

$$\begin{aligned} \boldsymbol{k} &= (k(\boldsymbol{x}_1,\boldsymbol{x}),\ldots,k(\boldsymbol{x}_\ell,\boldsymbol{x}))^\top, \\ \boldsymbol{C} &= \boldsymbol{K} + \sigma^2 \boldsymbol{I}_\ell. \end{aligned}$$

σ^2 denotes the variance of the observation noise. From Eq.(16), we see that the computational cost to calculate m is the same as that for KRR: $O(r\ell^2)$ to compute $\boldsymbol{C}^{-1}\boldsymbol{y}$ with the use of CG [41]. For computing s^2, one needs to compute $\boldsymbol{C}^{-1}\boldsymbol{k}$ in making predictions for each input $\boldsymbol{x}$, which costs $O(\ell^2)$ with the use of CG at every trial of a prediction. This can be problematic for repeated predictions, making the precomputation of $\boldsymbol{C}^{-1}$ via (incomplete) Cholesky factorization a useful approach in practice.

For choosing hyperparameter values such as the noise variance σ^2 and the kernel parameters, a common approach from a Bayesian perspective is evidence maximization. More specifically, the hyperparameters are chosen so that the marginalized log-likelihood

$$-\frac{1}{2}\ln\det(\boldsymbol{C}) - \frac{1}{2}\boldsymbol{y}^\top \boldsymbol{C}^{-1}\boldsymbol{y}$$

is maximized (for a derivation, see [16]). Unlike KRR, no closed-form solution is known for σ, and currently, gradient-based methods are the only practical methods to learn the hyperparameters. At each update step of the search direction, $O(\ell^3)$ computations are needed to compute the inverse of $\boldsymbol{C}$. This can be prohibitive for large datasets.

Speeding up GPR is an active research area, and many approaches have been proposed to date. One popular approach is to reduce the size of the problem in some way. The Nyström method [17] was a popular approach until it was shown to be extremely sensitive to the parameter choice and thus impractical [18]. A recent research trend is to use pseudo-inputs, rather than to use a subset of samples [42, 43]. However, these methods are not mature enough for use in critical applications since most of them require solving a complicated non-convex optimization problems.

4.1.5 Relevance Vector Machines

The *Relevance Vector Machine (RVM)* [25] is a Bayesian regression method that is capable of producing sparse solutions. While the Lasso uses ℓ_1 regularization for sparsity, RVM utilizes the automatic relevance determination mechanism to obtain sparse solutions. RVM assumes the linear regression model as in Eq.(6) (for centered data), but treats the coefficient $\boldsymbol{\alpha}$ as a random variable with a prior

$$p(\boldsymbol{\alpha}|\boldsymbol{\theta}) \equiv \prod_{i=1}^{N} \mathcal{N}(\alpha_i|0, \theta_i^{-1}),$$

where $\mathcal{N}(\alpha|0, \theta_i^{-1})$ denotes a Gaussian density with mean 0 and variance θ_i^{-1}. If we also assume Gaussian observation noise with mean 0 and variance σ^2, we have a Gaussian posterior distribution for $\boldsymbol{\alpha}$ with mean and covariance given by

$$\begin{aligned}\bar{\boldsymbol{\alpha}} &= \sigma^{-2}\boldsymbol{\Sigma}\boldsymbol{K}\boldsymbol{y},\\ \boldsymbol{\Sigma} &= \left(\boldsymbol{\Theta} + \sigma^{-2}\boldsymbol{K}^2\right)^{-1},\end{aligned}$$

respectively, where $\boldsymbol{\Theta}$ is the matrix whose diagonal elements are $\{\theta_i\}_{i=1}^N$ and all the off-diagonal elements are zero.

To determine $\{\theta_i\}_{i=1}^N$ and σ, RVM uses evidence approximation to obtain the following simple updating rules [25]:

$$\begin{aligned}\theta_i^{\text{new}} &\leftarrow \gamma_i/\bar{\alpha}_i^2,\\ \sigma_{\text{new}}^2 &\leftarrow \frac{\|\boldsymbol{y} - \boldsymbol{K}\bar{\boldsymbol{\alpha}}\|^2}{\ell - \sum_i \gamma_i},\end{aligned}$$

where γ_i is defined as $1 - \theta_i\Sigma_{ii}$. While updating the parameters by the above equations require $O(\ell^3)$ computational costs in naive implementations, an efficient algorithm is known [44]—this requires only $O(M^3)$, where M is the number of relevance vectors included at an update round. For detailed discussions on the RVM, see [2].

4.1.6 Support Vector Regression

Support vector regression (SVR) [2] is formulated as a minimization problem of the following objective function:

$$J_{\text{SVR}}(\boldsymbol{w}) = \frac{C}{\ell}\sum_{i=1}^{\ell} l_\epsilon\left(y_i - \langle \boldsymbol{x}_i, \boldsymbol{w}\rangle\right) + \frac{1}{2}\|\boldsymbol{w}\|_2^2, \tag{17}$$

where l_ϵ is called the ϵ-insensitive loss function defined as

$$l_\epsilon(\eta) \equiv \begin{cases} 0 & \text{if } |\eta| < \epsilon,\\ |\eta| - \epsilon & \text{otherwise.}\end{cases}$$

A standard approach to obtaining the SVR solution is to solve a quadratic programming problem derived as the dual problem of Eq.(17). Recently, it was shown that linear SVR can be efficiently solved with the bundle methods for regularized empirical risk minimization [45]. For this approach, see Section 4.2.1.

4.2 Classification

In this subsection, we turn our focus to classification by support vector machines (SVMs). More specifically, we review recent approaches to SVM learning from huge datasets, in which both the number of examples and the number of dimensions are extremely large. Such massive problems are found, e.g., in document or compound classification and linear SVMs are often preferred to nonlinear SVMs in this context. There are mainly two reasons for this: One is that linear SVMs performs as accurate as non-linear SVM in these extremely high-dimensional problems; and the other reason is that the feature vectors are typically sparse, which can be utilized for developing computationally efficient training algorithms. Here we review two state-of-the-art approaches: a cutting plane algorithm [45] and a coordinate descent algorithm [46].

Note that these methods are specialized to linear SVMs. One might consider employing these approaches in non-linear kernel methods by extracting feature vectors from a kernel matrix using eigenvalue decomposition. However, the feature vectors extracted as such are typically dense and therefore advantages of these approaches cannot be enjoyed.

4.2.1 Cutting Plane Algorithm

We begin with a cutting plane algorithm called the *bundle methods for regularized risk minimization* (BMRM) [45]. BMRM is applicable to general linear learning machines.

As seen in Section 2, many learning machines are trained by minimizing the objective function consisting of the empirical risk $R_{\text{emp}}(\boldsymbol{w})$ and a regularization function $\Omega(\boldsymbol{w})$:

$$J(\boldsymbol{w}) \equiv \Omega(\boldsymbol{w}) + C R_{\text{emp}}(\boldsymbol{w}), \tag{18}$$

where $\boldsymbol{w} \in \mathbb{R}^d$ is the model parameters and $C \in \mathbb{R}$ is a positive constant. For simplicity, we here focus on binary classification, and we assume to be given ℓ labeled training examples $(\boldsymbol{x}_i, y_i) \in \mathbb{R}^d \times \{\pm 1\} \quad (i = 1, \ldots, \ell)$. The empirical risk is expressed as

$$R_{\text{emp}}(\boldsymbol{w}) \equiv \frac{1}{\ell} \sum_{i=1}^{\ell} l(\boldsymbol{x}_i, y_i; \boldsymbol{w}),$$

where $l(\cdot, \cdot; \cdot)$ is a loss function; we assume the loss function is nonnegative and convex.

The linear SVMs use the ℓ_2-norm of $\boldsymbol{w}$ as the regularization function $\Omega(\cdot)$ and the hinge loss for $l(\cdot, \cdot; \cdot)$. Namely,

$$\begin{aligned} \Omega(\boldsymbol{w}) &\equiv \frac{1}{2}\|\boldsymbol{w}\|^2, \\ l(\boldsymbol{x}, y; \boldsymbol{w}) &\equiv \max(0, 1 - y \langle \boldsymbol{w}, \boldsymbol{x} \rangle). \end{aligned}$$

Note that the bias term is not used here (i.e., we set $b = 0$ in the linear scoring function $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$).

Solving the learning problem directly is often computationally intractable for large datasets. BMRM tackles this issue by using the cutting plane algorithm as follows. For any set of the model parameters $\mathcal{W} \subseteq \mathbb{R}^d$, the empirical risk can be bounded from below as

$$R_{\text{emp}}(\boldsymbol{w}) \geq \langle \boldsymbol{a}(\bar{\boldsymbol{w}}), \boldsymbol{w} \rangle + b(\bar{\boldsymbol{w}}), \qquad \forall \boldsymbol{w} \in \mathcal{W}, \tag{19}$$

where

$$\begin{aligned} \boldsymbol{a}(\bar{\boldsymbol{w}}) &\equiv \partial_{\boldsymbol{w}} R_{\text{emp}}(\bar{\boldsymbol{w}}), \\ b(\bar{\boldsymbol{w}}) &\equiv R_{\text{emp}}(\bar{\boldsymbol{w}}) - \langle \boldsymbol{a}(\bar{\boldsymbol{w}}), \bar{\boldsymbol{w}} \rangle . \end{aligned}$$

The plane $\langle \boldsymbol{a}(\boldsymbol{w}), \boldsymbol{x} \rangle + b(\boldsymbol{w}) = 0$ is called the cutting plane. For example, in the case of the hinge loss, the parameters of the cutting plane is expressed as

$$\boldsymbol{a}(\bar{\boldsymbol{w}}) = -\frac{1}{\ell} \sum_{i=1}^{\ell} c_i y_i \boldsymbol{x}_i, \qquad b(\bar{\boldsymbol{w}}) \equiv \frac{1}{\ell} \sum_{i=1}^{\ell} c_i,$$

where

$$c_i \equiv \begin{cases} 1 & \text{if } y_i(\bar{\boldsymbol{w}}^\top \boldsymbol{x}_i) < 1, \\ 0 & \text{otherwise.} \end{cases}$$

We can easily see that the parameters of the cutting plane can be computed with the order of the number of non-zero values in the whole training set. This bound can be utilized to develop the BMRM algorithm [45]:

$$R(\boldsymbol{w}; \mathcal{W}) \equiv \max_{\bar{\boldsymbol{w}} \in \mathcal{W}} \left(\langle \boldsymbol{a}(\bar{\boldsymbol{w}}), \boldsymbol{w} \rangle + b(\bar{\boldsymbol{w}}) \right),$$

BMRM is summarized as below:

```
1: Input: ${\boldsymbol{x}_i, y_i}_{i=1}^{\ell}$.
2: $t \leftarrow 0$;   $\boldsymbol{w}_t \leftarrow \mathbf{0}$;   $\mathcal{W} \leftarrow \emptyset$;
3: repeat
4:    $\mathcal{W} \leftarrow \mathcal{W} \cup \{\boldsymbol{w}_t\}$;
5:    Compute the parameters of the cutting plane, $\boldsymbol{a}(\boldsymbol{w}_t)$ and $b(\boldsymbol{w}_t)$ to compose
      $R(\bar{\boldsymbol{w}}; \mathcal{W})$;
6:    $\boldsymbol{w}_{t+1} \leftarrow \operatorname{argmin}_{\bar{\boldsymbol{w}}} \Omega(\bar{\boldsymbol{w}}) + CR(\bar{\boldsymbol{w}}; \mathcal{W})$;
7:    $t \leftarrow t + 1$;
8: until converged;
9: return $\boldsymbol{w}$.
```

The algorithm is guaranteed to converge to the optimum solution [45]. In the case of linear SVM, it is shown [47] that the algorithm terminates at the optimum after a limited number of iterations; thus the time complexity for training is a linear function of the number of non-zero values in the whole training set.

The following two modifications have been proposed to further speed-up BMRM [48]:

- A new step is inserted after Step 6. This step searches for a new solution $\boldsymbol{w}_t^{\mathrm{b}}$ that minimizes the original objective function in Eq.(18) by the following line search:

$$\boldsymbol{w}_t^{\mathrm{b}} \leftarrow \min_{\beta \geq 0} J((1-\beta)\boldsymbol{w}_{t-1}^{\mathrm{b}} + \beta \boldsymbol{w}_t).$$

- The cutting plane is constructed from interpolation between $\boldsymbol{w}_t$ and $\boldsymbol{w}_t^{\mathrm{b}}$.

The modified algorithm is called the *optimized cutting plane algorithm* (OCAS) [48], and it was shown that the number of iterations needed for convergence is the same order as that in [47]. On the other hand, OCAS is reported to be much faster in numerical experiments than the algorithm proposed in [47].

4.2.2 Dual Coordinate Descent Algorithm

Another type of popular techniques for training linear SVMs is the dual coordinate descent algorithm (DCDA) [46]. While the cutting plane methods [47, 48] reviewed above solved the primal problems DCDA solves the dual problem:

Problem 1.

$$\textit{min}\ \frac{1}{2}\boldsymbol{\alpha}^\top \boldsymbol{Q}\boldsymbol{\alpha} - \|\boldsymbol{\alpha}\|_1 \quad \textit{wrt}\ \boldsymbol{\alpha} \in \mathbb{R}_+^\ell \quad \textit{subj to}\ \boldsymbol{\alpha} \leq \frac{C}{\ell}\mathbf{1}_\ell,$$

The matrix $\boldsymbol{Q} \in \mathbb{S}_+^\ell$ has elements

$$Q_{ij} \equiv y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle .$$

Below, we describe the detail of DCDA. Let us denote the dual objective function by $J_{\mathrm{dual}}(\boldsymbol{\alpha})$. Note that Problem 1 has no equality constraint, which comes from the absence of the bias term in the primal problem (i.e., $b = 0$ in $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$). DCDA repeats the outer loop until $\boldsymbol{\alpha}$ becomes optimal. Each outer iteration has ℓ inner iterations. The ith inner iteration optimizes α_i fixing the other variables. Namely, the inner iteration minimizes the following quadratic function with respect to a scalar δ:

$$J_{\mathrm{dual}}(\boldsymbol{\alpha}^{\mathrm{old}} + \delta \boldsymbol{e}_i) \equiv \frac{1}{2} Q_{ii}\delta^2 + \partial_{\alpha_i} J_{\mathrm{dual}}(\boldsymbol{\alpha}^{\mathrm{old}})\delta + \mathrm{const.},$$

subject to the following box constraint:

$$0 \leq \alpha_i + \delta \leq \frac{C}{\ell}.$$

The solution can be obtained analytically. If $Q_{ii} > 0$, the new value of α_i is given by

$$\alpha_i^{\mathrm{new}} \leftarrow \min\left(\max\left(\alpha_i^{\mathrm{old}} - \frac{\partial_{\alpha_i} J_{\mathrm{dual}}(\boldsymbol{\alpha}^{\mathrm{old}})}{Q_{ii}}, 0\right), \frac{C}{\ell}\right),$$

where $\boldsymbol{\alpha}^{\text{old}}$ is the old value of $\boldsymbol{\alpha}$. The current value of the model parameters can be written as

$$\boldsymbol{w}^{\text{old}} \equiv \sum_{i=1}^{\ell} y_i \alpha_i^{\text{old}} \boldsymbol{x}_i.$$

This allows us to express the gradient as

$$\partial_{\alpha_i} J_{\text{dual}}(\boldsymbol{\alpha}^{\text{old}}) = y_i \left\langle \boldsymbol{w}^{\text{old}}, \boldsymbol{x}_i \right\rangle - 1.$$

This can be computed with $O(s)$, where s is the average number of non-zero elements. To compute the gradient in each iteration, DCDA also updates the model parameters by

$$\boldsymbol{w}^{\text{new}} \leftarrow \boldsymbol{w}^{\text{old}} + (\alpha_i^{\text{new}} - \alpha_i^{\text{old}}) y_i \boldsymbol{x}_i.$$

This update takes $O(s)$. Totally, the computational cost of the inner loop is $O(s)$. It was shown [46] that the series of the solutions generated by DCDA converges to the optimum solution.

Sequential minimal optimization (SMO) [49, 50], is also well-known as an efficient algorithm for training SVM. SMO breaks the original quadratic programming (QP) problem into a series of smallest possible QP problems (i.e., with two parameters). These small QP problems can be actually solved analytically and therefore we can avoid employing a numerical QP optimization as an inner loop. SMO is more general than DCDA in that linear/non-linear SVMs with the bias term can be dealt with, while DCDA would be faster than SMO but is specialized to linear SVMs without the bias term.

5 Unsupervised Methods

In this section, we give a review of unsupervised methods that could be applied to large-scale datasets.

5.1 Dimensionality Reduction

The goal of dimensionality reduction is to reduce the dimensionality of samples while most of the 'intrinsic' information contained in the data is kept. Let $\{\boldsymbol{x}_i\}_{i=1}^{\ell}$ be the original samples of d-dimension and we would like to reduce the dimensionality of these samples to r $(1 \leq r \leq d)$; let $\{\boldsymbol{z}_i\}_{i=1}^{\ell}$ be the dimension-reduced expressions of $\{\boldsymbol{x}_i\}_{i=1}^{\ell}$. In the case of linear dimensionality reduction, the dimension-reduced sample $\boldsymbol{z}_i$ is given by using a transformation matrix $\boldsymbol{T}$ $(\in \mathbb{R}^{r \times d})$ as

$$\boldsymbol{z}_i \equiv \boldsymbol{T} \boldsymbol{x}_i.$$

Otherwise $\boldsymbol{z}_i$ is obtained by a non-linear transformation of $\boldsymbol{x}_i$.

Principal component analysis (PCA), which finds the subspace that retains the maximum variance of the data, would be one of the most fundamental methods of linear unsupervised dimensionality reduction [51]. A PCA solution can be computed through eigendecomposition of the sample-covariance matrix, which is a $d \times d$ matrix. Thus even when ℓ is huge, PCA may still be computationally tractable as long as d is not too large.

A non-linear variant of PCA has been investigated in the context of neural network learning [52]. However, since neural network learning involves non-convex optimization, it is hard to obtain a good solution in a systematic way. After emergence of support vector machines [53], the kernel trick has been applied in PCA and a non-linear variant called kernel PCA has been developed [9]. However, for computing a kernel PCA solution, the $\ell \times \ell$ kernel matrix needs to be eigendecomposed which requires $O(\ell^3)$ computational costs; this may be infeasible when ℓ is huge.

In the last decade, various types of non-linear unsupervised dimensionality reduction methods have been proposed and their properties have been studied [54, 55, 56, 57, 58, 59]. Among them, we briefly review a scalable method called the *Laplacian eigenmap* [56] below.

First, we give a brief review of a linear version of the Laplacian eigenmap called the *locality preserving projection* (LPP) [60]. The basic idea of LPP is to seek a transformation matrix $\boldsymbol{T}$ such that nearby data pairs in the original space $\mathbb{R}^d$ are kept close in the embedding space $\mathbb{R}^r$. Thus, LPP tends to preserve local structure of the data.

Let $\boldsymbol{A}$ be an *affinity matrix*, i.e., the $\ell \times \ell$ matrix with $A_{i,j}$ being the affinity between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. We assume that $A_{i,j} \in [0,1]$; $A_{i,j}$ is large if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are 'close' and $A_{i,j}$ is small if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are 'far apart'. There are several different manners of defining $\boldsymbol{A}$. Among them, we adopt the nearest neighbor method here, i.e., $A_{i,j} = 1$ if $\boldsymbol{x}_i$ is a k-nearest neighbor of $\boldsymbol{x}_j$ or vice versa; otherwise $A_{i,j} = 0$. This definition is advantageous in that the affinity matrix $\boldsymbol{A}$ becomes sparse if k is not so large. Let $\boldsymbol{D}$ be the $\ell \times \ell$ diagonal matrix with

$$D_{i,i} \equiv \sum_{j=1}^{\ell} A_{i,j},$$

and let $\boldsymbol{\mathcal{L}}$ be the $\ell \times \ell$ matrix called the *Laplacian matrix* [61] defined by

$$\boldsymbol{\mathcal{L}} \equiv \boldsymbol{D} - \boldsymbol{A}. \tag{20}$$

Note that $\boldsymbol{\mathcal{L}}$ is sparse if $\boldsymbol{A}$ is sparse. Then the LPP transformation matrix $\boldsymbol{T}_{\mathrm{LPP}}$ is defined as

$$\boldsymbol{T}_{\mathrm{LPP}} \equiv \mathop{\mathrm{argmin}}_{\boldsymbol{T} \in \mathbb{R}^{r \times d}} \mathrm{tr}(\boldsymbol{T}\boldsymbol{X}\boldsymbol{\mathcal{L}}\boldsymbol{X}^\top\boldsymbol{T}^\top(\boldsymbol{T}\boldsymbol{X}\boldsymbol{D}\boldsymbol{X}^\top\boldsymbol{T}^\top)^{-1}).$$

Let $\{\boldsymbol{\varphi}_t\}_{t=1}^d$ be the generalized eigenvectors associated with the generalized eigenvalues $\{\mu_t\}_{t=1}^d$ of the following generalized eigenvalue problem:

$$\boldsymbol{X}\boldsymbol{\mathcal{L}}\boldsymbol{X}^\top\boldsymbol{\varphi} = \mu\boldsymbol{X}\boldsymbol{D}\boldsymbol{X}^\top\boldsymbol{\varphi}.$$

We assume that the generalized eigenvalues are sorted as

$$\mu_1 \leq \cdots \leq \mu_d,$$

and the generalized eigenvectors are normalized as

$$\boldsymbol{\varphi}_t^\top \boldsymbol{X}\boldsymbol{D}\boldsymbol{X}^\top \boldsymbol{\varphi}_t = 1 \quad \text{for } t = 1, \ldots, d.$$

Note that this normalization is often automatically carried out by an eigensolver. Then a solution $\boldsymbol{T}_{\text{LPP}}$ is given analytically as

$$\boldsymbol{T}_{\text{LPP}} = (\boldsymbol{\varphi}_1 | \cdots | \boldsymbol{\varphi}_r)^\top.$$

$\boldsymbol{X}\boldsymbol{\mathcal{L}}\boldsymbol{X}^\top$ and $\boldsymbol{X}\boldsymbol{D}\boldsymbol{X}^\top$ are $d \times d$ matrices and $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{D}$ are sparse; so even when ℓ is huge, LPP may still be computationally tractable as long as d is not too large.

By an application of the kernel trick, LPP can be non-linearized in a straightforward manner. That is, the above generalized eigenvalue problem can be kernelized as

$$\boldsymbol{K}\boldsymbol{\mathcal{L}}\boldsymbol{K}\boldsymbol{\alpha} = \mu \boldsymbol{K}\boldsymbol{D}\boldsymbol{K}\boldsymbol{\alpha},$$

where $\boldsymbol{K}\boldsymbol{\alpha}$ corresponds to $\boldsymbol{X}^\top \boldsymbol{\varphi}$. Since $\boldsymbol{K}\boldsymbol{\mathcal{L}}\boldsymbol{K}$ and $\boldsymbol{K}\boldsymbol{D}\boldsymbol{K}$ are $\ell \times \ell$ matrices, this generalized eigenvalue problem may not be computationally tractable when ℓ is large. However, if we only reduce the dimensionality of training samples $\{\boldsymbol{x}_i\}_{i=1}^\ell$, the solution of the above generalized eigenvalue problem can be obtained by solving the following sparse generalized eigenvalue problem.

$$\boldsymbol{\mathcal{L}}\boldsymbol{\alpha} = \mu \boldsymbol{D}\boldsymbol{\alpha}.$$

More specifically, a solution is given by

$$(\boldsymbol{z}_1 | \cdots | \boldsymbol{z}_\ell) = (\boldsymbol{\alpha}_2 | \cdots | \boldsymbol{\alpha}_{r+1})^\top,$$

where $\{\boldsymbol{\alpha}_t\}_{t=1}^\ell$ are the generalized eigenvectors of the above generalized eigenvalue problem associated with the generalized eigenvalues $\mu_1 \leq \cdots \leq \mu_\ell$. Note that μ_1 is always zero so $\boldsymbol{\alpha}_1$ is discarded. As long as $\boldsymbol{A}$ is sparse, the above generalized eigenvalue problem may be solved efficiently even when ℓ is huge.

5.2 Spectral Clustering

The goal of data clustering is to group a given set of samples so that samples in the same group are 'similar' to each other. Let $\{\boldsymbol{x}_i\}_{i=1}^\ell$ be the original samples of d-dimension and we would like to group the samples into k disjoint groups. We assume that the number of clusters k is fixed in advance.

The k-means clustering algorithm would be one of the most fundamental clustering algorithms. Let $\mathcal{C}_t$ be the set of samples assigned to the cluster t. Every sample belongs to one of the clusters without overlap, i.e., $\{\mathcal{C}_t\}_{t=1}^k$ satisfies

$$\bigcup_{t=1}^k \mathcal{C}_t = \{\boldsymbol{x}_i\}_{i=1}^\ell \quad \text{and} \quad \mathcal{C}_t \cap \mathcal{C}_{t'} = \emptyset \text{ for } t \neq t'.$$

If the similarity of samples within the same class is measured by the variance, the optimal solution may be given as the minimizer of the following optimization problem:

$$\min_{\{\mathcal{C}_t\}_{t=1}^k} \sum_{t=1}^{k} \sum_{\boldsymbol{x} \in \mathcal{C}_t} \|\boldsymbol{x} - \boldsymbol{\mu}_t\|^2,$$

where $\boldsymbol{\mu}_t$ is the 'center' of the cluster t defined by

$$\boldsymbol{\mu}_t = \frac{1}{|\mathcal{C}_t|} \sum_{\boldsymbol{x}' \in \mathcal{C}_t} \boldsymbol{x}'.$$

However, this discrete minimization problem is known to be NP-hard and thus may not be exactly solved in practice. A local optimal solution may be obtained by the following algorithm called *k-means clustering*:

1. Randomly initialize the cluster partition $\{\mathcal{C}_t\}_{t=1}^k$.

2. For $i = 1, \ldots, \ell$, assign $\boldsymbol{x}_i$ to the cluster which has the closest center, i.e, to the cluster $\mathcal{C}_{\widehat{t}}$ where
$$\widehat{t} = \operatorname*{argmin}_t \|\boldsymbol{x}_i - \boldsymbol{\mu}_t\|.$$

3. Repeat this until convergence.

The k-means clustering algorithm corresponds to maximum likelihood estimation of a Gaussian mixture model, so the 'shape' of clusters is limited to be convex. This limitation could be mitigated by the use of the kernel trick, which results in *kernel k-means* [62]: the sample $\boldsymbol{x}_i$ is assigned to the cluster $\mathcal{C}_{\widehat{t}}$, where

$$\widehat{t} = \operatorname*{argmax}_t \sum_{\boldsymbol{x}' \in \mathcal{C}_t} k(\boldsymbol{x}, \boldsymbol{x}') - \frac{2}{|\mathcal{C}_t|} \sum_{\boldsymbol{x}', \boldsymbol{x}'' \in \mathcal{C}_t} k(\boldsymbol{x}', \boldsymbol{x}'').$$

However, the kernel k-means algorithm tends to suffer from more serious local optimum problems than the plain k-means algorithm due to non-linearity introduced by the kernel function $k(\cdot, \cdot)$.

Another idea for clustering is to group the data samples so that the sum of the similarity values among different clusters is minimized [63]:

$$\operatorname*{argmin}_{\{\mathcal{C}_t\}_{t=1}^k} \sum_{t=1}^{k} \frac{\sum_{\boldsymbol{x} \in \mathcal{C}_t} \sum_{\boldsymbol{x}' \notin \mathcal{C}_t} A(\boldsymbol{x}, \boldsymbol{x}')}{\sum_{\boldsymbol{x}'' \in \mathcal{C}_t} \sum_{i=1}^{\ell} A(\boldsymbol{x}'', \boldsymbol{x}_i)},$$

where $A(\boldsymbol{x}, \boldsymbol{x}')$ denotes the affinity value between $\boldsymbol{x}$ and $\boldsymbol{x}'$. Note that this is equivalent to finding the *minimum normalized-cut* in graph theory if the affinity matrix $\boldsymbol{A}$ is regarded as the adjacency matrix of a graph. It was shown that the above normalized-cut criterion is equivalent to a weighted variant of the kernel k-means criterion [62]. Thus a slightly modified kernel k-means algorithm could be used for obtaining a local optimal solution of the normalized-cut criterion. However, the problem of frequently trapped by a local optimum still remains.

Instead of learning the cluster assignment $\{\mathcal{C}_t\}_{t=1}^k$, let us learn the cluster indicator matrix $\boldsymbol{W}$, which is the $k \times \ell$ matrix defined by

$$W_{t,i} = \begin{cases} 1 & \text{if } \boldsymbol{x}_i \in \mathcal{C}_t, \\ 0 & \text{otherwise.} \end{cases}$$

Then, it is known that the solution of the following optimization problem agrees with normalized-cut clustering [62]:

$$\operatorname*{argmin}_{\boldsymbol{A} \in \mathcal{B}} \operatorname{tr}(\boldsymbol{A}\boldsymbol{\mathcal{L}}\boldsymbol{A}^\top) \text{ subject to } \boldsymbol{A}\boldsymbol{D}\boldsymbol{A}^\top = \boldsymbol{I},$$

where $\mathcal{B}$ is the set of all $k \times \ell$ matrices such that one of the elements in each column takes one and others are all zero. This problem is again NP-hard so we may not be able to obtain the optimal solution. A relaxation approach is to allow $\boldsymbol{A}$ to take real values, i.e., $\boldsymbol{A} \in \mathbb{R}^{k \times \ell}$. Then the solution of the relaxed problem is shown to agree with the solution of the Laplacian eigenmap [62]. This implies that the embedded samples by the Laplacian eigenmap has 'soft' cluster structure.

Based on this finding, a clustering algorithm called *spectral clustering* has been developed [64]:

1. Apply the Laplacian eigenmap algorithm to the original d-dimensional samples $\{\boldsymbol{x}_i\}_{i=1}^\ell$ and obtain their $(k-1)$-dimensional expressions $\{\boldsymbol{z}_i\}_{i=1}^\ell$.
2. Apply the plain k-means clustering algorithm to $\{\boldsymbol{z}_i\}_{i=1}^\ell$ and obtain the clustering result $\{\mathcal{C}_t\}_{t=1}^k$.

Thanks to the soft clustering property of the Laplacian eigenmap, the local optimum problem tends to be mitigated in the above spectral clustering algorithm.

6 Kernel Functions

In this section, we consider computational issues for the kernel functions themselves.

The first part of this section is on designing kernel functions for complex data structures such as sequences, trees, and graphs. The second part of this section is on the automatic selection of informative kernels when we have multiple kernel functions for different information sources.

6.1 Kernels for Structured Data

In Section 2, we have seen the Gaussian kernel and the polynomial kernel as examples of kernel functions where their computational costs are much smaller than the dimension of the feature space.

Here, we consider more complex data structures, such as sequences, trees, or more general graph-structured data, and efficient kernel methods for handling such data. When we consider *structure* in data, we have two types of structures, *external* structures and *internal* structures. Graph structures formed by relationships among data are called external structures, meaning that the feature spaces are given as graph structures. On the other hand, graph structures observed inside each data sample are called internal structure.

6.1.1 Externally Structured Data

When we consider an external structure, our main concern is how to define a kernel function between arbitrary two nodes (representing two data instances) in a given graph $G = (E, V)$ representing the feature space. The *diffusion kernel* [65] defines similarity among nodes by a diffusion process over the graph. Let us denote the Laplacian matrix of the given graph by $\boldsymbol{\mathcal{L}}$. Then the kernel matrix for the diffusion kernel is defined as

$$\boldsymbol{K} \equiv \exp(-t\boldsymbol{\mathcal{L}}) = \boldsymbol{I} + (-t\boldsymbol{\mathcal{L}}) + \frac{(-t\boldsymbol{\mathcal{L}})^2}{2!} + \frac{(-t\boldsymbol{\mathcal{L}})^3}{3!} + \cdots . \tag{21}$$

The naive computation of the diffusion kernel is time-consuming, but diagonalization of $\boldsymbol{\mathcal{L}}$ makes the computation efficient. Let us consider a diagonalization of $\boldsymbol{\mathcal{L}}$ as $\boldsymbol{\mathcal{L}} = \boldsymbol{U}^{-1}\boldsymbol{D}\boldsymbol{U}$, where $\boldsymbol{D}$ is a diagonal matrix and $\boldsymbol{U}$ is a non-singular matrix. Then Eq.(21) becomes

$$\boldsymbol{K} = \boldsymbol{U}^{-1}\exp(t\boldsymbol{D})\boldsymbol{U}.$$

Since $\boldsymbol{D}$ is a diagonal matrix, $\exp(t\boldsymbol{D})$ is also a diagonal matrix whose i-th diagonal element is $\exp(tD_{ii})$.

In contrast to the diffusion kernel where the task is to predict the characteristics of the data in a network-structured feature space, we can consider the different problem of predicting the external structure (meaning the network structure) from the given feature spaces. In that case, the task is to predict pair-wise relationships (such as the existence of links) among data. The *pair-wise kernel* [66, 67, 68] is can be used for such purposes. Let us denote by $\tilde{\boldsymbol{K}}$ a kernel matrix defined by a given feature space. Then the pair-wise kernel $\boldsymbol{K}$ is defined as

$$\boldsymbol{K} \equiv \tilde{\boldsymbol{K}} \otimes \tilde{\boldsymbol{K}}, \tag{22}$$

where $\otimes$ is the Kronecker product of two matrices. The pair-wise kernels are understood as inner products in the product space of the two feature spaces of the given (element-wise) kernels. Since $\boldsymbol{K}$ is an $\ell^2 \times \ell^2$ dense matrix, it is not realistic to store the whole matrix in memory (especially when we want to consider relationships that are more complicated than pairs) and sub-sampling or sequential learning is used. Instead of Eq.(22), The *Kronecker sum pair-wise* kernel has been proposed [69, 70]:

$$\boldsymbol{K} \equiv \tilde{\boldsymbol{K}} \oplus \tilde{\boldsymbol{K}} = \tilde{\boldsymbol{K}} \otimes \boldsymbol{I} + \boldsymbol{I} \otimes \tilde{\boldsymbol{K}},$$

where $\oplus$ indicates the Kronecker sum. This is much sparser than the (Kronecker product) pair-wise kernel and easier to compute for large-scale data sets.

Note that, in the case of external structure with bipartite graphs between two distinct sets, the pair-wise kernel matrix is defined as $\boldsymbol{K} \equiv \tilde{\boldsymbol{K}}_1 \otimes \tilde{\boldsymbol{K}}_2$, where $\tilde{\boldsymbol{K}}_1$ and $\tilde{\boldsymbol{K}}_2$ are the kernel matrices for the two sets.

6.1.2 Internally Structured Data

Now we turn to designing kernel functions for internally structured data. It seems natural to consider that the characteristics of internally structured data can be represented by *substructures* such as subsequences for sequences, subtrees for trees, and subgraphs for graphs. The idea of the *convolution kernels* [71] is to construct feature spaces by using the substructures. The challenge is to efficiently compute the inner products in a feature space defined by the substructures since there exist a enormous number of possible substructures and explicit construction of feature vectors is prohibitively expensive.

For sequential data where elements are connected by linear structures, a sequence kernel has been constructed based on subsequences [72]. The algorithm is based on dynamic programming, and can be computed in quadratic time. A simpler but more efficient kernel called the *spectrum kernel* uses (contiguous) substrings and can be computed in linear time [73, 74]. The spectrum kernel with gaps [75] was proposed as an intermediate variation of these two string kernels.

For tree-structured data, dynamic programming-based kernels were extended for trees by using subtrees as features [76, 77]; a spectrum kernel for trees which can be computed in almost linear time has also been proposed [78].

For graph-structured data, graph kernels for labeled graphs have been proposed [79, 80]. Since subgraph-based graph kernels are still prohibitively expensive, paths generated by random walks were used as features. A recursive formulation similar to those of the other dynamic programming-based kernels results in a system of linear equations, and allows us to efficiently compute the graph kernels. Several approaches to accelerating the computation are discussed in [81].

6.2 Multiple Kernels

Recently, multiple kernel learning has received considerable attention in the field of machine learning. Multiple kernel learning involves training a learning machine from multiple kernels instead of selecting a single kernel used for learning. In the early years, the multiple kernels were simply integrated as sums or averages [82], but the research focus has shifted to automatic selection of the optimal combinations of kernels [83, 84]. A formulation for multiple kernel learning based on *semidefinite programming* (SDP) has been presented in [83]. This section begins by reviewing this formulation.

Given ℓ labeled training examples $(\boldsymbol{x}_i, y_i) \in \mathbb{R}^d \times \{\pm 1\} \quad (i = 1, \ldots, \ell)$, the regularized risk of an SVM is given by

$$\begin{aligned} J &\equiv \min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{\ell}\sum_{i=1}^{\ell} \max(0, 1 - y_i(\langle \boldsymbol{w}, \phi(\boldsymbol{x}_i)\rangle + b)) \\ &= \max_{\boldsymbol{\alpha}\in\mathcal{S}} \|\boldsymbol{\alpha}\|_1 - \frac{1}{2}\boldsymbol{\alpha}^\top \boldsymbol{Q}\boldsymbol{\alpha}, \end{aligned}$$

where the matrix $\boldsymbol{Q} \in \mathbb{S}_+^{\ell}$ has elements

$$Q_{ij} \equiv y_i y_j \left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle .$$

The set $\mathcal{S}$ is defined as

$$\mathcal{S} \equiv \{\boldsymbol{\alpha} \in \mathbb{R}_+^{\ell} \,|\, \boldsymbol{y}^\top \boldsymbol{\alpha} = 0,\, \boldsymbol{\alpha} \leq \frac{C}{\ell}\mathbf{1}_{\ell}\}.$$

ϕ is a map from an input space to an embedding space on which the learning machine operates. Learning the map ϕ is equivalent to learning the kernel matrix $\boldsymbol{K}^{\text{int}} \in \mathbb{S}_+^{\ell}$ in which $K_{ij}^{\text{int}} = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j)\rangle$. Learning the optimal kernel matrix from a subset of a semidefinite cone $\mathcal{K} \subset \mathbb{S}_+^{\ell}$ has been considered in [83]. Typical multiple kernel learning algorithms assume that the subset $\mathcal{K}$ is the set of weighted averages of p given kernel matrices $\boldsymbol{K}_k \in \mathbb{S}_+^{\ell}, (k = 1, \ldots, p)$. When we use multiple kernel learning, we often normalize each kernel matrix so that every diagonal element is one. We here impose this assumption for simplicity. Then we can write the problem for learning the kernel matrix which minimizes the regularized risk as

Problem 2.

$$\begin{aligned} &\mathit{min} && \max_{\boldsymbol{\alpha}\in\mathcal{S}} J(\boldsymbol{\alpha}; \boldsymbol{K}^{int}) \qquad \mathit{wrt} \quad \boldsymbol{\beta} \in \Delta_p \\ &\mathit{subj\ to} && \boldsymbol{K}^{int} = \sum_{k=1}^{p} \beta_k \boldsymbol{K}_k, \\ &\mathit{where} && J(\boldsymbol{\alpha}\,;\, \boldsymbol{K}) \equiv \|\boldsymbol{\alpha}\|_1 - \frac{1}{2}\boldsymbol{\alpha}^\top \operatorname{diag}(\boldsymbol{y})\boldsymbol{K}\operatorname{diag}(\boldsymbol{y})\boldsymbol{\alpha}, \\ &&& \Delta_p \equiv \{\boldsymbol{\beta} \in \mathbb{R}_+^p \mid \|\boldsymbol{\beta}\|_1 = 1\}. \end{aligned}$$

It was shown [83] that Problem 2 is an SDP problem and can be reduced to *second-order cone programming* (SOCP), which can be solved more efficiently than SDP. However, the problem is still intractable when the number of training examples is large. A smoothed version of the problem so that SMO [50] is applicable has been introduced in [85].

Another efficient algorithm for solving Problem 2 without modification has been presented in [86]. This algorithm is based on the following linear programming problem:

Problem 3.

$$\begin{aligned} &\mathit{min} && \theta \qquad \mathit{wrt} \quad \theta \in \mathbb{R}, \quad \boldsymbol{\beta} \in \boldsymbol{\Delta}_p \\ &\mathit{subj\ to} && \forall \boldsymbol{\alpha} \in \mathcal{S}' : \sum_{k=1}^{p} \beta_k J(\boldsymbol{\alpha}\,;\, \boldsymbol{K}_k) \leq \theta, \\ &\mathit{where} && \mathcal{S}' \textit{ is a subset of } \mathcal{S}. \end{aligned}$$

The optimal kernel weights coincide with the solution of Problem 2 when $\mathcal{S}' = \mathcal{S}$. However, since $\mathcal{S}$ is an infinite set, the problem cannot be solved by general-purpose linear programming tools. In [86], this is solved iteratively as follows. Initially set $\mathcal{S}' = \emptyset$ and $\boldsymbol{\beta} = \mathbf{1}/p$; then repeat the following two steps until convergence:

1: Perform an SVM learning with the kernel matrix integrated with the current kernel weights $\boldsymbol{\beta}$ and add the obtained $\boldsymbol{\alpha} \in \mathcal{S}$ to $\mathcal{S}'$;
2: Solve Problem 3 to get the new kernel weights.

Another algorithm for multiple kernel learning has been proposed in [87]. This algorithm basically repeats the following two steps until convergence:

1: Train SVM using the kernel matrix combined with the current kernel weights $\boldsymbol{\beta}$ and obtain $\boldsymbol{\alpha} \in \mathcal{S}$.
2: Update the kernel weights $\boldsymbol{\beta}$ by using line search from the current weights in the reduced descent direction of $J(\boldsymbol{\alpha}\,;\, \sum_k \beta_k \boldsymbol{K}_k)$.

So far we have focused on binary classification tasks and used the hinge loss for multiple kernel learning. Multiple kernel learning methods for other tasks such as multi-class classification, regression, and one-class classification has been discussed in [86, 88]. However, the computational cost for multiple kernel learning is still too large for learning from large datasets. To cope with this problem, recent work replaces each kernel matrix with an undirected graph [89, 90]. An efficient algorithm for learning from multiple graphs in binary classification developed in [89] has been reported to be approximately 30 times faster than the algorithm proposed in [86].

7 Conclusions

Optimization theory has been extensively studied in recent decades and various new techniques have been explored to exploit the explosive increases of computational power. The development of optimization methodologies has also had strong impact on the machine learning community, allowing it to tackle large-scale real-world problems. At the same time, beyond standard optimization paradigms, machine learning algorithms have highly intricate structures and great efforts have been devoted to specific problems of machine learning, such as improving overall computation times including model selection by tracking the entire solution path, reducing the computational costs in the test phases by sparcifying solutions, and improving the computation of complicated kernel functions by using special data structures. Thus, for further improvements to the computational efficiency of machine learning algorithms, it is important to have interdisciplinary collaboration between fundamental mathematical areas such as optimization theory, combinatorics, and statistics and various application areas such as bioinformatics, computational chemistry, robotics, natural language processing, speech analysis, and image processing.

The kernel methods we have dealt with in this article are essentially converting linear algorithms to non-linear domains by using the kernel trick. Another movement in the kernel machine community, which we could not cover in the current article due to the space limitation, is to use the kernel trick in the context of *statistical tests*, including the independence test and the two-sample test. These ideas have been applied in various domains such as independent component analysis [12], dimensionality reduction [91], feature selection [92], and non-stationarity adaptation [93]. The computational issues in this new topic do not seem to have been extensively explored yet. This may be a challenging new research topic for kernel method researchers.

A popular classification approach apart from the support vector machines is *boosting* [94, 95, 96], which sequentially combines weak learners to make predictions. Although boosting and the support vector machines are very different in spirit, recent work has shown that both are improving the *margin distribution* in some sense [97, 98]. Thus, going beyond computational issues, it would be important for the kernel method community to systematically merge the ideas of these two different approaches and develop a more sophisticated approach.

References

[1] V. Vapnik, The Nature of Statistical Learning Theory, Springer, 2000.

[2] C. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[3] J. Shawe-Taylor and N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge University Press New York, NY, USA, 2004.

[4] L. Bottou, O. Chapelle, D. Decoste, and J. Weston, eds., Large-scale Kernel Machines, The MIT Press, Cambridge, MA, 2007.

[5] G. Wahba, "Spline models for observational data," volume 59 of CBMS-NSF Regional Conference Series in Applied Mathematics, SIAM, 1987.

[6] J. Mercer, "Functions of positive and negative type, and their connection with the theory of integral equations," Proceedings of the Royal Society of London. Series A, vol.83, no.559, pp.69–70, 1909.

[7] N. Aronszajn, "Theory of reproducing kernels," Transactions of the American Mathematical Society, vol.68, pp.337–404, 1950.

[8] G. Strang, Linear Algebra and its Applications, Academic Press, 1976.

[9] B. Schölkopf, A. Smola, and K.R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," Neural Computation, vol.10, no.5, pp.1299–1319, 1998.

[10] H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, Numerical Recipes in C, 2nd. ed., Cambridge University Press, 1992.

[11] S. Fine and K. Scheinberg, "Efficient SVM training using low-rank kernel representations," The Journal of Machine Learning Research, vol.2, pp.243–264, 2002.

[12] F. Bach and M. Jordan, “Kernel independent component analysis,” The Journal of Machine Learning Research, vol.3, pp.1–48, 2003.

[13] G.H. Golub and C.F.V. Loan, Matrix Computations (3rd Edition), Johns Hopkins University Press, Baltimore, MD, 1996.

[14] T. Idé and K. Tsuda, “Change-point detection using Krylov subspace learning,” Proceedings of 2007 SIAM International Conference on Data Mining, pp.515–520, 2007.

[15] S. Mahadevan, “Fast spectral learning using Lanczos eigenspace projections,” Proc. AAAI Conference on Artificial Intelligence, pp.1472–1475, 2008.

[16] C.E. Rasmussen and C. Williams, Gaussian Processes for Machine Learning, MIT Press, 2006.

[17] C.K.I. Williams and M. Seeger, “Using the Nyström method to speed up kernel machines,” Advances in Neural Information Processing Systems 13, pp.682–688, 2001.

[18] C.K.I. Williams, C.E. Rasmussen, A. Schwaighofer, and V. Tresp, “Observations on the Nyström method for Gaussian process prediction,” tech. rep., 2002.

[19] N.D. Freitas, Y. Wang, M. Mahdaviani, and D. Lang, “Fast Krylov methods for N-body learning,” in Advances in Neural Information Processing Systems 18, pp.251–258, 2006.

[20] C. Yang, R. Duraiswami, and L. Davis, “Efficient kernel machines using the improved fast Gauss transform,” Advances in Neural Information Processing Systems 17, pp.1561–1568, 2005.

[21] D. Lee, A.G. Gray, and A.W. Moore:, “Dual-tree fast Gauss transforms,” Advances in Neural Information Processing Systems 18, pp.1561–1568, 2006.

[22] J. Shawe-Taylor and N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge University Press, 2004.

[23] R. Rosipal and L.J. Trejo, “Kernel partial least squares regression in reproducing kernel Hilbert space,” Journal of Machine Learning Research, vol.2, pp.97–123, 2001.

[24] R. Tibshirani, “Regression shrinkage and selection via the lasso,” Journal of the Royal Statistical Society, B, vol.58, no.1, pp.267–288, 1996.

[25] M.E. Tipping, “Sparse Bayesian learning and the relevance vector machine,” Journal of Machine Learning Research, vol.1, pp.211–244, 2001.

[26] G.C. Cawley and N.L. Talbot, “Fast exact leave-one-out cross-validation of sparse least-squares support vector machines,” Neural Networks, vol.17, no.10, pp.1467–1475, 2004.

[27] N. Krämer and M.L. Braun, “Kernelizing PLS, degrees of freedom, and efficient model selection,” in Proceedings of International Conference on Machine Learning, pp.441–448, 2007.

[28] N. Krämer, M. Sugiyama, and M. Braun, “Lanczos approximations for the speedup of kernel partial least squares regression,” Proceedings of Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS2009), 2009.

[29] M. Momma, “Efficient computations via scalable sparse kernel partial least squares and boosted latent features,” Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.654–659, 2005.

[30] J. Arenas-Garcia, K.B. Petersen, and L.K. Hansen, “Sparse kernel orthogonalized PLS for feature extraction in large data sets,” in Advances in Neural Information Processing Systems 18, pp.33–40, 2006.

[31] H. Saigo, N. Krämer, and K. Tsuda, “Partial least squares regression for graph mining,” Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.578–586, 2008.

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

[33] M.J. Best, “An algorithm for the solution of the parametric quadratic programming problem,” CORR Report 82-24, Faculty of Mathematics, University of Waterloo, 1982.

[34] J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani, “Pathwise coordinate optimization,” Annals of Applied Statistics, vol.1, no.2, pp.302–332, 2007.

[35] N. Meinshausen and P. Bühlmann, “High-dimensional graphs and variable selection with the lasso,” Annals of Statistics, vol.34, no.3, pp.1436–1462, 2006.

[36] J. Friedman, T. Hastie, and R. Tibshirani, “Sparse inverse covariance estimation with the graphical lasso,” Biostatistics, vol.9, no.3, pp.432–441, 2008.

[37] T. Idé, A.C. Lozano, N. Abe, and Y. Liu, “Proximity-based anomaly detection using sparse structure learning,” Proceedings of 2009 SIAM International Conference on Data Mining, 2009.

[38] V. Roth, “Sparse kernel regressors,” Proceedings of International Conference on Artificial Neural Networks, pp.339–346, 2001.

[39] V. Roth, “The generalized lasso,” IEEE Transactions on Neural Networks, vol.15, no.1, pp.16–28, 2004.

[40] G. Wang, D.Y. Yeung, and F.H. Lochovsky, "The kernel path in kernelized lasso," Proc. the Eleventh International Workshop on Artificial Intelligence and Statistics, pp.580–587, 2007.

[41] M.N. Gibbs, Bayesian Gaussian Processes for Regression and Classification, Ph.D. thesis, Department of Physics, University of Cambridge, 1997.

[42] J. Quiñonero-Candela and C.E. Rasmussen, "A unifying view of sparse approximate Gaussian process regression," Journal of Machine Learning Research, vol.6, pp.1939–1959, 2005.

[43] E. Snelson and Z. Ghahramani, "Sparse Gaussian processes using pseudo-inputs," Advances in Neural Information Processing Systems 18, ed. Y. Weiss, B. Schölkopf, and J. Platt, Cambridge, MA, pp.1257–1264, MIT Press, 2006.

[44] M.E. Tipping and A.C. Faul, "Fast marginal likelihood maximisation for sparse Bayesian models," Proc. the Ninth International Workshop on Artificial Intelligence and Statistics, pp.3–6, 2003.

[45] C.H. Teo, Q. Le, A.J. Smola, and S.V.N. Vishwanathan, "A scalable modular convex solver for regularized risk minimization," Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, pp.727–736, ACM, 2007.

[46] C.J. Hsieh, K.W. Chang, C.J. Lin, S.S. Keerthi, and S. Sundararajan, "A dual coordinate descent method for large-scale linear SVM," Proceedings of the 25nd International Conference on Machine Learning, New York, NY, pp.408–415, ACM, 2008.

[47] T. Joachims, "Training linear SVMs in linear time," Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, pp.217–226, ACM, 2006.

[48] V. Franc and S. Sonnenburg, "Optimized cutting plane algorithm for support vector machines," Proceedings of the 25nd International Conference on Machine Learning, New York, NY, pp.320–327, ACM, 2008.

[49] J.C. Platt, "Fast training of support vector machines using sequential minimal optimization," in Advances in Kernel Methods - Support Vector Learning, ed. B. Schölkopf, C. Burges, and A. Smola, pp.185–208, MIT Press, Cambridge, MA, 1999.

[50] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy, "Improvements to Platt's SMO algorithm for SVM classier design," Neural Computation, vol.13, pp.637–649, March 2001.

[51] I.T. Jolliffe, Principal Component Analysis, Springer-Verlag, New York, 1986.

[52] E. Oja, "A simplified neuron model as a principal component analyzer," Journal of Mathematical Biology, vol.15, pp.267–273, 1982.

[53] V.N. Vapnik, Statistical Learning Theory, Wiley, New York, 1998.

[54] J.B. Tenenbaum, V. de Silva, and J.C. Langford, "A global geometric framework for nonlinear dimensionality reduction," Science, vol.290, no.5500, pp.2319–2323, 2000.

[55] S. Roweis and L. Saul, "Nonlinear dimensionality reduction by locally linear embedding," Science, vol.290, no.5500, pp.2323–2326, 2000.

[56] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," Neural Computation, vol.15, no.6, pp.1373–1396, 2003.

[57] D.L. Donoho and C.E. Grimes, "Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data," Proceedings of the National Academy of Arts and Sciences, pp.5591–5596, 2003.

[58] J. Ham, D.D. Lee, S. Mika, and B. Schölkopf, "A kernel view of the dimensionality reduction of manifolds," Proceedings of the Twenty-First International Conference on Machine Learning, New York, NY, ACM Press, 2004.

[59] K.Q. Weinberger and L.K. Saul, "Unsupervised learning of image manifolds by semidefinite programming," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2004.

[60] X. He and P. Niyogi, "Locality preserving projections," in Advances in Neural Information Processing Systems 16, ed. S. Thrun, L. Saul, and B. Schölkopf, MIT Press, Cambridge, MA, 2004.

[61] F.R.K. Chung, Spectral Graph Theory, American Mathematical Society, Providence, R.I., 1997.

[62] I.S. Dhillon, Y. Guan, and B. Kulis, "Kernel k-means, spectral clustering and normalized cuts," Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, pp.551–556, ACM Press, 2004.

[63] J. Shi and J. Malik, "Normalized cuts and image segmentation," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.22, no.8, pp.888–905, 2000.

[64] A.Y. Ng, M.I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," Advances in Neural Information Processing Systems 14, ed. T.G. Dietterich, S. Becker, and Z. Ghahramani, Cambridge, MA, MIT Press, 2002.

[65] R.I. Kondor and J. Lafferty, "Diffusion kernels on graphs and other discrete input spaces," Proceedings of the 19th International Conference on Machine Learning, pp.315–322, 2002.

[66] S. Oyama and C.D. Manning, "Using feature conjunctions across examples for learning pairwise classifiers," Proceedings of the 15th European Conference on Machine Learning, pp.322–333, 2004.

[67] J. Basilico and T. Hofmann, "Unifying collaborative and content-based filtering," Proceedings of the 21st International Conference on Machine Learning, 2004.

[68] A. Ben-Hur and W.S. Noble, "Kernel methods for predicting protein-protein interactions," Bioinformatics, vol.21, no.Suppl. 1, pp.i38–i46, 2005.

[69] H. Kashima, S. Oyama, Y. Yamanishi, and K. Tsuda, "On pairwise kernels: An efficient alternative and generalization analysis," Proceedings of the 13th Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2009.

[70] H. Kashima, T. Kato, Y. Yamanishi, M. Sugiyama, and K. Tsuda, "Link propagation: A fast semi-supervised algorithm for link prediction," Proceedings of the 2009 SIAM International Conference on Data Mining, 2009.

[71] D. Haussler, "Convolution kernels on discrete structures," Tech. Rep. UCSC-CRL-99-10, University of California in Santa Cruz, 1999.

[72] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins, "Text classification using string kernels," Journal of Machine Learning Research, vol.2, pp.419–444, 2002.

[73] C. Leslie, E. Eskin, and W. Noble, "The spectrum kernel: A string kernel for SVM protein classification," Proceedings of the Pacific Symposium on Biocomputing, ed. R.B. Altman, A.K. Dunker, L. Hunter, K. Lauerdale, and T.E. Klein, pp.566–575, World Scientific, 2002.

[74] S. Vishwanathan and A. Smola, "Fast kernels for string and tree matching," Advances in Neural Information Processing Systems 15, ed. S. Becker, S. Thrun, and K. Obermayer, Cambridge, MA, MIT Press, 2003.

[75] C. Leslie, E. Eskin, J. Weston, and W. Noble, "Mismatch string kernels for svm protein classification," Advances in Neural Information Processing Systems 15, ed. S. Becker, S. Thrun, and K. Obermayer, Cambridge, MA, MIT Press, 2003.

[76] M. Collins and N. Duffy, "Convolution kernels for natural language," Advances in Neural Information Processing Systems 14, Cambridge, MA, MIT Press, 2002.

[77] H. Kashima and T. Koyanagi, "Kernels for semi-structured date," Proceedings of the 19th International Conference on Machine Learning, San Francisco, CA, pp.291–298, Morgan Kaufmann, 2002.

[78] H. Kuboyama, K. Hirata, K.F. Aoki-Kinoshita, H. Kashima, and H. Yasuda, "A gram distribution kernel applied to glycan classification and motif extraction," Proceedings of the 17th International Conference on Genome Informatics, 2006.

[79] H. Kashima, K. Tsuda, and A. Inokuchi, "Marginalized kernels between labeled graphs," Proceedings of the 20th International Conference on Machine Learning, San Francisco, CA, Morgan Kaufmann, 2003.

[80] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," Proceedings of the 16th Annual Conference on Computational Learning Theory, 2003.

[81] S.V.N. Vishwanathan, K. Borgwardt, and N. Schraudolph, "Fast computation of graph kernels," Advances in Neural Information Processing Systems 19, 2007.

[82] P. Pavlidis, J. Weston, J. Cai, and W.S. Noble, "Learning gene functional classifications from multiple data types," Journal of Computational Biology, vol.9, no.2, pp.401–411, 2002.

[83] G.R.G. Lanckriet, N. Cristianini, P. Bartlett, L.E. Ghaoui, and M.I. Jordan, "Learning the kernel matrix with semidefinite programming," Journal of Machine Learning Research, vol.5, pp.27–72, 2004.

[84] T. Kato, K. Tsuda, and K. Asai, "Selective integration of multiple biological data for supervised network inference," Bioinformatics, vol.21, pp.2488–2495, 2005.

[85] F.R. Bach, G.R.G. Lanckriet, and M.I. Jordan, "Multiple kernel learning, conic duality, and the SMO algorithm," Proceedings of the 21st International Conference on Machine Learning, 2004.

[86] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf, "Large scale multiple kernel learning," Journal of Machine Learning Research, vol.7, pp.1531–1565, Jul 2006.

[87] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet, "Simplemkl," Journal of Machine Learning Research, vol.9, pp.2491–2521, 2008.

[88] A. Zien and C.S. Ong, "Multiclass multiple kernel learning," Proceedings of the 24th International Conference on Machine Learning, New York, NY, pp.1191–1198, ACM, 2007.

[89] T. Kato, H. Kashima, and M. Sugiyama, "Robust label propagation on multiple networks," IEEE Transactions on Neural Networks.

[90] T. Kato, Y. Murata, K. Miura, K. Asai, P.B. Horton, K. Tsuda, and W. Fujibuchi, "Network-based de-noising improves prediction from microarray data," BMC Bioinformatics, vol.7(Suppl 1), March 2006.

[91] K. Fukumizu, F.R. Bach, and M.I. Jordan, "Dimensionality reduction for supervised learning with reproducing kernel Hilbert spaces," Journal of Machine Learning Research, vol.5, no.Jan, pp.73–99, 2004.

[92] L. Song, A. Smola, A. Gretton, K.M. Borgwardt, and J. Bedo, "Supervised feature selection via dependence estimation," Proceedings of the 24th International Conference on Machine learning, pp.823–830, 2007.

[93] J. Huang, A. Smola, A. Gretton, K.M. Borgwardt, and B. Schölkopf, "Correcting sample selection bias by unlabeled data," in Advances in Neural Information Processing Systems 19, pp.601–608, 2007.

[94] Y. Freund and R.E. Schapire, "Experiments with a new boosting algorithm," International Conference on Machine Learning, 1996.

[95] Y. Freund and R.E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," Journal of Computer and System Sciences, vol.55, pp.119–139, 1997.

[96] R. Schapire, Y. Freund, P. Bartlett, and W. Lee, "Boosting the margin: A new explanation for the effectiveness of voting methods," Annals of Statistics, vol.26, pp.1651–1686, 1998.

[97] A. Takeda and M. Sugiyama, "Nu-support vector machine as conditional value-at-risk minimization," Proceedings of 25th Annual International Conference on Machine Learning, ed. A. McCallum and S. Roweis, pp.1056–1063, 2008.

[98] L. Wang, M. Sugiyama, C. Yang, Z.H. Zhou, and J. Feng, "On the margin explanation of boosting algorithms," Proceedings of 21st International Conference on Learning Theory, ed. R. Servedio and T. Zhang, pp.479–490, 2008.