CMPUT 551 Fall 2011 Machine Learning Lecture Notes

Junfeng Wen
Department of Computing Science
University of Alberta

Last Update: February 5, 2017

Preface

This document is my lecture note taken from CMPUT 551 Fall 2011, Machine Learning at University of Alberta, given by Prof. Dale Schuurmans.

This course mainly attempted to organize existing algorithms from an optimization perspective. In order to understand these notes thoroughly, you should be familiar with basic linear algebra as well as fundamental optimization. (They will come back as we proceed, hopefully.)

In this note, there are two kinds of blocks that provide more information on the idea or concept being discussed. The first kind introduces examples:

An example

This is an example.

The second kind provides more detailed information:

Further information

Some further details.

Readers may want to skip some of these contents.

Notations

- Bold letter (e.g., $\boldsymbol{a}$) as vector[1], normal letter (e.g., a) as scalar.
- $\boldsymbol{a} \succeq \boldsymbol{b}$ means each component of $\boldsymbol{a}$ is greater than or equal to the corresponding component of $\boldsymbol{b}$. Specifically, $\boldsymbol{a} \succeq \mathbf{0}$ means each component of $\boldsymbol{a}$ is non-negative.
- $\boldsymbol{a}^\top$ means the transpose of $\boldsymbol{a}$.
- A_{ij} means the entry of matrix A that is in the ith row and jth column. $A_{i:}$ means the ith row of A (as a row vector) and $A_{:j}$ means the jth column of A (as a column vector). $A \succeq 0$ means matrix A is positive semi-definite unless specified otherwise.

[1]In this note, by default we assume all vectors are column vectors unless specified otherwise.

Contents

1 Linear Framework **7**
1.1 General Framework 7
1.2 Data Representation and Error Function 7
1.3 Solving Optimization 8
1.4 Geometry Interpretation 8
1.5 Convexity/Robustness Tradeoff 8
1.6 Beyond Linear Model 10

2 Regularization and Kernel **13**
2.1 Learning with Regularization 13
2.2 The Representer Theorem 13
2.3 Kernel 14
2.4 Operations that Preserve PSD 14
2.5 Feature Selection: L_1 Regularization 15

3 Output Transformation **17**
3.1 Transform Function 17
3.2 Matching Loss 17
3.3 Surrogate Loss for Classification 19

4 Support Vector Machine **21**
4.1 Large Margin Classification 21
4.2 Duality 23
4.3 Application of Duality: SVM 25

5 Large Output **27**
5.1 Multivariate Prediction 27
5.1.1 Feature Expansion 28
5.1.2 Kernelization 28
5.1.3 Feature Selection: Sparsity 28
5.1.4 Output Transformation and Matching Loss 30
5.1.5 Surrogate Loss for Classification 31
5.2 Structured Output 31

6 Unsupervised Learning **35**
6.1 Dimensionality Reduction 35
6.2 Hard Clustering 36
6.3 Kernelization 37
6.3.1 Kernel Principal Component Analysis 37
6.3.2 Kernelized Clustering 38
6.3.3 Normalized Kernel Methods 39

Chapter 1

Linear Framework

1.1 General Framework

Most machine learning algorithms attempt to learn a function (mapping) from given examples:

$$\begin{tikzcd}
\begin{matrix}(x_1,y_1)\\ \vdots \\ (x_t,y_t)\end{matrix} \arrow[r] & \boxed{\text{Learner}} \arrow[r] & h : \mathcal{X} \mapsto \mathcal{Y}
\end{tikzcd}$$

Figure 1.1: Learning procedure.

where x_i comes from a domain $\mathcal{X}$ and y_i lies in a range $\mathcal{Y}$ for all $i \in \{1, 2, \cdots, t\}$. For example, our input x_i can be an image, and the output $y_i \in \{0, 1\}$ can be an indicator to determine whether there is a dog in the image. In most cases, we define an error function $\text{err}(\widehat{y}, y)$ to measure the difference between the true output y and our prediction $\widehat{y} = h(x)$ for some prediction function $h(\cdot)$. A generic learning algorithm usually find the function $h(\cdot)$ via empirical error minimization on the training set: given a hypothesis space $\mathcal{H}$ and t training examples (x_i, y_i), we do

$$\min_{h\in\mathcal{H}} \quad \sum_{i=1}^{t} \text{err}(h(x_i), y_i)$$

1.2 Data Representation and Error Function

As an example that we will further investigate in the following chapters, assume that $\boldsymbol{x}_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}$. For instance, a 16×16 gray-scale image can be vectorized into a 256-dimensional input. Accordingly, our hypothesis space can also be parametrized, that is

$$\mathcal{H} = \{h_{\boldsymbol{w}} | \boldsymbol{w} \in \mathbb{R}^n, h_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}\}.$$

Note that for this linear model, the offset (bias) is omitted. Instead of $h_{\boldsymbol{w},b}(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$, we simply augmented the input $\boldsymbol{x}_i$ by appending a 1 on each data point: $[\boldsymbol{x}_i; 1]$. The offset is then encoded in the last element of $\boldsymbol{w}$. Therefore, the offset is ignored without loss of generality hereafter. With the above assumptions, we can represent the input as a matrix $X \in \mathbb{R}^{t\times n}$, whose rows are data points, and output as a vector $\boldsymbol{y} \in \mathbb{R}^t$.

As for error function, the following are commonly used in the machine learning literature.

$$\begin{aligned}
&L_1 \text{ error: } \text{err}(\widehat{\boldsymbol{y}}, \boldsymbol{y}) = \|\widehat{\boldsymbol{y}} - \boldsymbol{y}\|_1 \\
&L_2 \text{ error: } \text{err}(\widehat{\boldsymbol{y}}, \boldsymbol{y}) = \|\widehat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2 \\
&L_p \text{ error: } \text{err}(\widehat{\boldsymbol{y}}, \boldsymbol{y}) = \|\widehat{\boldsymbol{y}} - \boldsymbol{y}\|_p^p \quad 0 \le p \le \infty
\end{aligned}$$

(Recall that the L_1 norm of a vector $\boldsymbol{a}$ is the sum of its absolute values $|a_j|$; general information about L_p norm can be found online[1].)

If $y \in [0, 1]$ is a probability, typically we can use the *cross entropy* to measure the error: cross entropy

$$\text{err}(\widehat{y}, y) = y \ln\left(\frac{y}{\widehat{y}}\right) + (1-y) \ln\left(\frac{1-y}{1-\widehat{y}}\right)$$

[1] `http://en.wikipedia.org/wiki/Lp_space`

1.3 Solving Optimization

convex

Now that we see the learning problem as an optimization, when is the training efficient? If the error function $\mathrm{err}(\widehat{\boldsymbol{y}}, \boldsymbol{y})$ is *convex* in $\widehat{\boldsymbol{y}}$, we can reliably identify the global minimizer h^*. Simply set the derivative to zero will lead to global optimal solution in most cases. If the error function is non-convex in $\widehat{\boldsymbol{y}}$, then we have trouble finding the best prediction function.

Now let's investigate two concrete examples. It is not so obvious how to solve them.

L_1 error minimization

$$\min_{\boldsymbol{w}} \quad \sum_{i=1}^{t} |\boldsymbol{w}^\top \boldsymbol{x}_i - y_i|$$

The above formulation tries to reduce the residual of true output y_i and predicted output $\widehat{y}_i = \boldsymbol{w}^\top \boldsymbol{x}_i$. We introduce a residual vector $\boldsymbol{\delta} \succeq \mathbf{0}$, whose component represents the absolute value of the difference between y_i and $\widehat{y}_i$. Here $\succeq$ means element-wise greater than or equal to. Then the minimization becomes

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{\delta}} \quad & \boldsymbol{\delta}^\top \mathbf{1} \\ \text{s.t.} \quad & \boldsymbol{\delta} \succeq \mathbf{0} \\ & X\boldsymbol{w} \preceq \boldsymbol{y} + \boldsymbol{\delta} \\ & X\boldsymbol{w} \succeq \boldsymbol{y} - \boldsymbol{\delta}. \end{aligned}$$

This new formulation is a linear programming[a] and can be solved efficiently.

[a] `http://en.wikipedia.org/wiki/Linear_programming`

L_∞ error minimization

$$\min_{\boldsymbol{w}} \max_{i} \quad |\boldsymbol{w}^\top \boldsymbol{x}_i - y_i|$$

Recall that the L_∞-norm of a vector is defined to be its component with maximum absolute value. Again, we can introduce a residual variable $\delta \geq 0$ and transform the original problem to

$$\begin{aligned} \min_{\boldsymbol{w},\delta} \quad & \delta \\ \text{s.t.} \quad & \delta \geq 0 \\ & \boldsymbol{w}^\top \boldsymbol{x}_i \leq y_i + \delta \qquad i = 1, \cdots, t \\ & \boldsymbol{w}^\top \boldsymbol{x}_i \geq y_i - \delta \qquad i = 1, \cdots, t \end{aligned}$$

Again, this is a linear programming.

1.4 Geometry Interpretation

The geometry interpretation of the linear model $\widehat{\boldsymbol{y}} = X\boldsymbol{w}$ is shown in Figure 1.2. Our prediction $\widehat{\boldsymbol{y}}$ lies in the column span of matrix X. Using the notation $X_{:i}$ to represent the ith column of X, $\widehat{\boldsymbol{y}}$ is in fact a linear combination of $X_{:i}$, with w_i as coefficients. The true output $\boldsymbol{y}$ can be factorized as $\boldsymbol{y} = \widehat{\boldsymbol{y}} + \boldsymbol{y}_\perp$. To minimize the L_2 squared error is equivalent to minimize the "length" of $\boldsymbol{y}_\perp$. Therefore, the best bet would be to choose the $\widehat{\boldsymbol{y}}$ such that $\boldsymbol{y}_\perp$ is perpendicular to the column space of X.

1.5 Convexity/Robustness Tradeoff

Figure 1.3 shows the L_p error as a function of our prediction $\widehat{y}$ with different p values. If $p \geq 1$, the error function is convex in $\widehat{y}$; if $p > 1$, the error function is smooth and differentiable. When p goes below 1, the error function is no longer convex, but grows gradually slower as we make prediction further from the true output y. Theoretically, when p reaches 0, the error function will be zero only when the prediction matches the output perfectly, otherwise the error is 1.

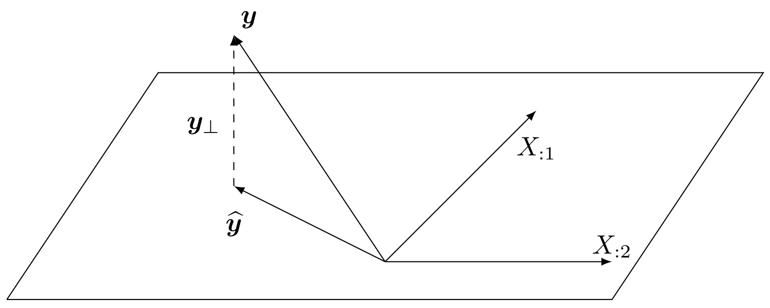
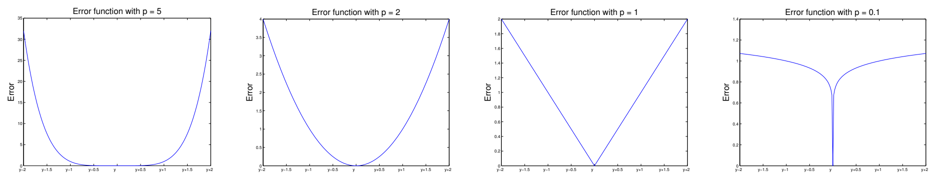


Figure 1.2: Learning procedure.



(a) $p = 5$ (b) $p = 2$ (c) $p = 1$ (d) $p = 0.1$

Figure 1.3: Error function with different p values.

Error function with smaller p is more robust. To see this, Figure 1.4 shows some training points generated from $y = x \cdot 0 + \epsilon, \epsilon \sim \mathcal{N}(0, 1^2)$ and one single outlier, which is far away from regularity. Here $\mathcal{N}(\mu, \sigma^2)$ is the *Normal distribution* with mean μ and variance σ^2. We fit three linear model $\widehat{y} = w \cdot x$ with L_1, L_2 and L_∞ error functions. Obviously, the L_∞ line chases the outlier significantly, followed by the L_2 line, then the L_1 line, which basically ignores the outlier.

Normal distribution

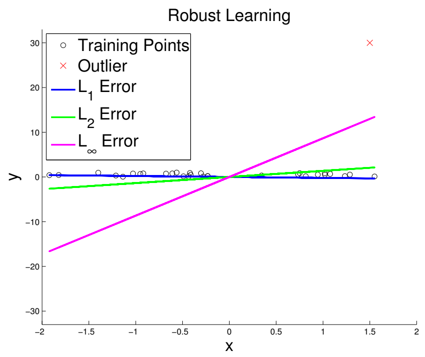


Figure 1.4: Learning with outlier, different error functions.

Let's discuss the phenomenon in Figure 1.4. Large p value in L_p indicates that the error grows rapidly as the prediction $\widehat{y}$ getting further away from the true output y, which makes the learned model more likely to chase the outlier since it creates huge error. On the other hand, small p value indicates that the error of one single outlier is insignificant (even though the its prediction is far away from its truth, see Figure 1.3(d)). As a result, the model can robustly ignore the outlier.

If robustness is so helpful and prevent fitting noise, why not just focus on error function with small p value? There is no such thing as a free lunch. Error function with p smaller than 1 is not convex and difficult to minimize. Therefore, we need to trade-off between convexity and robustness. Table 1.1 summarizes the pros and cons of convex and robust errors.

Table 1.1: Pros and cons of convex versus robust errors.

Error type	Pros	Cons
Convex error	Efficient optimization	Sensitive to noise
Robust error	Slow growth, robust	Difficult to optimize

Robust error functions

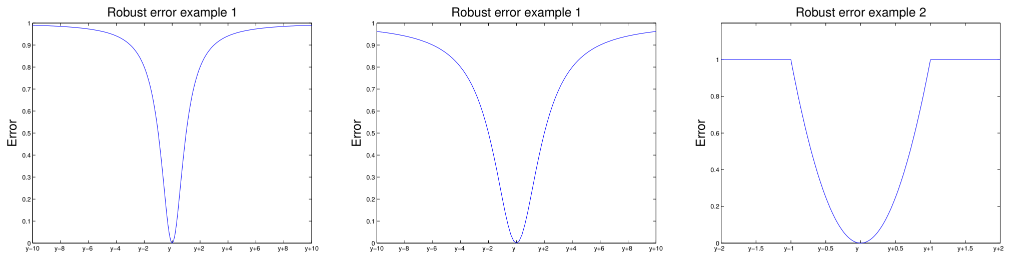
Example 1:

$$\mathrm{err}_\sigma(\widehat{y}, y) = \frac{(\widehat{y}-y)^2}{\sigma^2 + (\widehat{y}-y)^2} \qquad \sigma > 0$$

Example 2:

$$\mathrm{err}(\widehat{y}, y) = \min\left(1, (\widehat{y}-y)^2\right)$$

Figure 1.5 illustrates how they behave as $\widehat{y}$ getting further away from y.



(a) Example 1, $\sigma = 1$. (b) Example 1, $\sigma = 2$. (c) Example 2.

Figure 1.5: Robust error functions.

1.6 Beyond Linear Model

One may think why we focus exclusively on linear model $\widehat{y} = \boldsymbol{w}^\top \boldsymbol{x}$ rather than more complicated ones. In fact, linear model can be extended to non-linear one by expanding the input $\boldsymbol{x}$. That is, we can construct some functions $\varphi_i(\boldsymbol{x})$ to expand the input. The input matrix X is transformed into Φ, an feature-expanded representation, whose ith column is the feature constructed from $\varphi_i(\cdot)$. We predict on $\boldsymbol{x}$ by $\widehat{y} = \sum_i w_i \varphi_i(\boldsymbol{x})$ and we optimize

$$\boldsymbol{w} = \operatorname*{argmin}_{\boldsymbol{w}} \quad \|\Phi \boldsymbol{w} - \boldsymbol{y}\| \tag{1.1}$$

The following examples show how to use feature expansion to construct non-linear model.

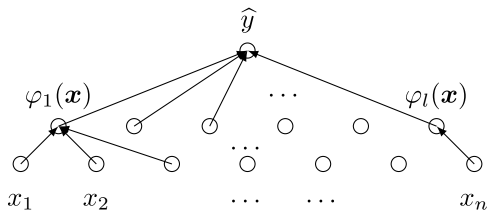
Polynomial curve fitting

Given $\{(x_1, y_1), (x_2, y_2), \cdots, (x_t, y_t)\}$, where both x_i and y_i are in $\mathbb{R}$, we construct the expanded feature representation as

$$\Phi = \begin{bmatrix} 1 & x_1 & \cdots & x_1^d \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_t & \cdots & x_t^d \end{bmatrix} \in \mathbb{R}^{t \times (d+1)},$$

where d is the degree of the polynomial curve we are fitting. With $d = t-1$ and (1.1), we can find the coefficients w_i of the perfect curve, that passes through all t points precisely.

Neural network (feed forward perceptrons)



As shown in the above figure, the neural network consists of one input layer, whose node represents one single feature of the input $\boldsymbol{x} \in \mathbb{R}^n$; one hidden layer, whose node represents an expanded (hidden) feature $\varphi_j(\boldsymbol{x})$ for the input; and one output layer which is a linear combination of hidden features $\widehat{y} = \sum_{j=1}^{l} w_j \varphi_j(\boldsymbol{x})$. Typically, a neural network learning procedure constructs the weight w_j and hidden feature $\varphi_j(\boldsymbol{x})$ simultaneously.

How to construct such features, or basis functions? One intuitive idea is that the new representation should reflect the similarity between inputs: similar inputs should have similar features. Or, we can simply choose an alternative "similarity function". The following are two examples of such similarity functions.

Gaussian similarity (a radial basis function)

$$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right)$$

1. If we use this similarity function as feature function and choose some fixed instances $\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_l \in \mathcal{X}$, the new representation would be
$$\boldsymbol{x} \mapsto \boldsymbol{\varphi}(\boldsymbol{x}) = [\kappa(\boldsymbol{x}, \boldsymbol{\mu}_1), \cdots, \kappa(\boldsymbol{x}, \boldsymbol{\mu}_l)]^\top .$$
2. If we place the centres $\boldsymbol{\mu}_j$ on the inputs $\boldsymbol{x}_i$ themselves, then the new data matrix Φ would be: $\Phi_{i,j} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Note that this Φ is symmetric since $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \kappa(\boldsymbol{x}_j, \boldsymbol{x}_i)$. Moreover, if Φ is invertible, the solution to the learning problem (1.1) with L_2 error will be $\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} \|\Phi \boldsymbol{w} - \boldsymbol{y}\|_2^2 = \Phi^{-1}\boldsymbol{y}$.

***k*-nearest neighbor similarity**

$$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \begin{cases} 1 & \text{if } \boldsymbol{x}_i \in N_k(\boldsymbol{x}_j) \\ 0 & \text{otherwise} \end{cases}$$

where $N_k(\boldsymbol{x}_j)$ is the set of points that are the top (closest) k nearest neighbors of $\boldsymbol{x}_j$. Note that this similarity function is not symmetric.

Chapter 2

Regularization and Kernel

2.1 Learning with Regularization

As one may notice, too many features may lead to over-fitting, while too few features may lead to under-fitting. For example, given t distinct 2-dimensional data points (x_i, y_i) with different x values, we can always find a polynomial curve of degree $t-1$ to perfectly fit all the points, at a price of highly quirky curve and poor generalization. To avoid such situation, we have two strategies:

- **Regularization** – to smooth the weights $\boldsymbol{w}$. Concretely, we add a regularizer $R(\boldsymbol{w})$ to our learning problem
$$\min_{\boldsymbol{w}} \quad \|\Phi\boldsymbol{w} - \boldsymbol{y}\| + \beta R(\boldsymbol{w})$$
where $\beta \geq 0$ is a parameter to control the degree of regularization. $R(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$ and $R(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$ are widely used in machine learning literature.
- **Feature selection** – select a subset of the expanded features for training. We will see later that feature selection can be achieved by $R(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$ regularization.

L_2 loss with L_2 regularization.

$$\min_{\boldsymbol{w}} \quad \|X\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \beta\|\boldsymbol{w}\|_2^2$$

We can solve the above optimization by setting derivative w.r.t. $\boldsymbol{w}$ to zero:

$$\begin{aligned} &X^\top(X\boldsymbol{w} - \boldsymbol{y}) + \beta\boldsymbol{w} = 0 \\ \Longrightarrow &(X^\top X + \beta I_t)\boldsymbol{w} = X^\top\boldsymbol{y} \end{aligned}$$

where I_t is the identity matrix of size $t \times t$, with ones on the diagonal and zeros everywhere else. Note that $(X^\top X + \beta I_t)$ is invertible[a] as long as $\beta > 0$. Therefore, the solution is

$$\boldsymbol{w}^* = (X^\top X + \beta I_t)^{-1} X^\top \boldsymbol{y}$$

[a]Real symmetric matrix $X^\top X$ can be eigen-decomposed as $X^\top X = Q^\top \Lambda Q$, where Q is orthogonal matrix, consisting of eigenvectors of $X^\top X$, and Λ is diagonal, with the corresponding eigenvalues (greater than or equal to 0) on the diagonal. Then $(X^\top X + \beta I_t) = Q^\top(\Lambda + \beta I_t)Q$. All elements on the diagonal of $(\Lambda + \beta I_t)$ is now greater than 0 as long as $\beta > 0$. Thus $(X^\top X + \beta I_t)^{-1} = Q^\top(\Lambda + \beta I_t)^{-1}Q$ is always invertible.

2.2 The Representer Theorem

The representer theorem[1] is one of the most important theorems in machine learning. It states an important property for L_2 regularization.

Theorem 1. *For any loss function L ($> -\infty$, i.e., bounded from below), the solution*

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}} \quad L(X\boldsymbol{w}, \boldsymbol{y}) + \beta\|\boldsymbol{w}\|_2^2 \tag{2.1}$$

always satisfies $\boldsymbol{w}^* = X^\top\boldsymbol{\alpha}$, for some $\boldsymbol{\alpha} \in \mathbb{R}^t$.

[1]The representer theorem here is simplified for the ease of mathematical burden. Visit `http://en.wikipedia.org/wiki/Representer_theorem` for a complete and formal statement.

Proof. Any $\boldsymbol{w}$ can be factorized as $\boldsymbol{w} = \boldsymbol{w}_0 + \boldsymbol{w}_1$, where $\boldsymbol{w}_1 \in \mathit{RowSpace}(X)$, that is, $\boldsymbol{w}_1 = X^\top \boldsymbol{\alpha}$ for some $\boldsymbol{\alpha} \in \mathbb{R}^t$, and $\boldsymbol{w}_0 \perp \mathit{RowSpace}(X)$ such that $X\boldsymbol{w}_0 = \boldsymbol{0}$. First, because $X\boldsymbol{w} = X(\boldsymbol{w}_0 + \boldsymbol{w}_1) = X\boldsymbol{w}_1$, the $\boldsymbol{w}_0$ component cannot affect the loss function L. Second, since $\|\boldsymbol{w}\|_2^2 = \|\boldsymbol{w}_0\|_2^2 + \|\boldsymbol{w}_1\|_2^2$, to minimize $\beta\|\boldsymbol{w}\|_2^2$, we simply set $\boldsymbol{w}_0$ to zero. Therefore, the optimal $\boldsymbol{w}^*$ should be $\boldsymbol{w}_1^*$, which is equal to $X^\top \boldsymbol{\alpha}$ for some $\boldsymbol{\alpha} \in \mathbb{R}^t$. $\square$

With the representer theorem, we obtain an equivalent formulation of the learning problem:

$$\min_{\boldsymbol{\alpha}} \quad L(XX^\top \boldsymbol{\alpha}, \boldsymbol{y}) + \beta \boldsymbol{\alpha}^\top XX^\top \boldsymbol{\alpha}$$

To predict the output of a new point $\boldsymbol{x}_0$, instead of $\boldsymbol{w}^\top \boldsymbol{x}_0$, we compute the its prediction as $\widehat{y} = \boldsymbol{\alpha}^\top X\boldsymbol{x}_0$. Furthermore, it turns out that we do not care about the input, and what really matters is the inner product of the inputs $\boldsymbol{x}_i^\top \boldsymbol{x}_j$. With the notation $K = XX^\top$ and $\boldsymbol{k}_0 = X\boldsymbol{x}_0$, the learning problem (2.1) becomes

$$\min_{\boldsymbol{\alpha}} \quad L(K\boldsymbol{\alpha}, \boldsymbol{y}) + \beta \boldsymbol{\alpha}^\top K \boldsymbol{\alpha}$$

and during test, $\widehat{y} = \boldsymbol{k}_0^\top \boldsymbol{\alpha}$. Recall that in previous chapter, we use a new feature vector $\boldsymbol{\varphi}(\boldsymbol{x})$ for training. By the representer theorem, we no longer need the feature vector explicitly if we have access to their inner product $\boldsymbol{\varphi}^\top(\boldsymbol{x}_i)\boldsymbol{\varphi}(\boldsymbol{x}_j)$ directly. This technique of replacing inner product is sometimes called the *kernel trick*.

kernel trick

2.3 Kernel

Now let us take a side step and learn the definitions of kernel operator and kernel matrix.

Definition 1. *A **kernel operator** $k(\cdot,\cdot)$ is a symmetric function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ such that for any finite subset $x_1, x_2, \cdots, x_t \in \mathcal{X}$,*

$$K = \begin{bmatrix} k(x_1,x_1) & \cdots & k(x_1,x_t) \\ \vdots & \ddots & \vdots \\ k(x_t,x_1) & \cdots & k(x_t,x_t) \end{bmatrix}$$

is positive semidefinite (PSD). Or equivalently, $k(x_i,x_j) = \langle \varphi(x_i), \varphi(x_j) \rangle$, the inner product of $\varphi(x_i)$ and $\varphi(x_j)$ for some feature representation $\varphi(\cdot)$.

Definition 2. *A **kernel matrix** K is a matrix such that $K = \Phi\Phi^\top$ for some matrix Φ.*

Theorem 2. *The following statements are equivalent:*

1. *K is a kernel matrix;*
2. *K is positive semidefinite;*
3. *Eigenvalues of K are non-negative.*

Proof. $1 \Rightarrow 2$: $\forall \boldsymbol{a}$,

$$\begin{aligned} \boldsymbol{a}^\top K \boldsymbol{a} &= \boldsymbol{a}^\top \Phi\Phi^\top \boldsymbol{a} \\ &= \boldsymbol{v}^\top \boldsymbol{v} \quad \text{with} \quad \boldsymbol{v} = \Phi^\top \boldsymbol{a} \\ &\geq 0 \end{aligned}$$

$2 \Rightarrow 3$: Trivial, by definition.

$3 \Rightarrow 1$: Let $K = Q\Lambda Q^\top$ be the eigen-decomposition of K. Since all the eigenvalues of K are non-negative, $\Phi = Q\Lambda^{\frac{1}{2}}$ exists and $K = \Phi\Phi^\top$. $\square$

2.4 Operations that Preserve PSD

Given the expanded feature function $\varphi(\cdot)$, we can always construct a kernel matrix by $K = \Phi\Phi^\top$. However, the converse is not that easy: $\varphi(\cdot)$ is not always easily accessible given a kernel matrix K. We can find $\varphi(\cdot)$ easily if it is transformed by PSD-preserving operations listed in Table 2.1.

Table 2.1: Operations that preserve PSD, given $x, y \in \mathcal{X}$.

Kernel modification	Corresponding feature modification
$\widetilde{k}(x,y) = k(x,y) + c, c \geq 0$	$\widetilde{\varphi}(x) = [\varphi(x); \sqrt{c}]$
$\widetilde{k}(x,y) = \frac{k(x,y)}{\sqrt{\langle k(x,x), k(y,y)\rangle}}$	$\widetilde{\varphi}(x) = \frac{\varphi(x)}{\sqrt{\langle \varphi(x), \varphi(x)\rangle}}$
$\widetilde{k}(x,y) = k_1(x,y) + k_2(x,y)$	$\widetilde{\varphi}(x) = [\varphi_1(x); \varphi_2(x)]$
$\widetilde{k}(x,y) = ak(x,y), a > 0$	$\widetilde{\varphi}(x) = \sqrt{a}\varphi(x)$
$\widetilde{k}(x,y) = k_1(x,y)k_2(x,y)$	$\widetilde{\varphi}(x) = [\cdots, \varphi_{1i}(x)\varphi_{2j}(x), \cdots]$ (feature cross product)
$\widetilde{k}(x,y) = k_2(\varphi_1(x), \varphi_1(y)), a > 0$	$\widetilde{\varphi}(x) = \varphi_2(\varphi_1(x))$

2.5 Feature Selection: L_1 Regularization

So far we have seen the benefits of the L_2 regularization ($R(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$ leads to the kernel trick). Now let us talk about the L_1 regularization ($R(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$).

Consider the problem of feature selection: given l features $\varphi_i(\boldsymbol{x}), i = 1, \cdots, l$, choose a subset of features to minimize the loss. This problem is difficult because we have 2^l subsets of features in total. It is intractable to enumerate all the possible subsets. Generally, finding a subset of bounded size that minimizes training loss is NP-hard.

The problem of feature selection can be formulated using the L_0 regularizer:

$$\min_{\boldsymbol{w}} \quad \sum_i L(\boldsymbol{x}_i^\top \boldsymbol{w}, y_i) + \beta\|\boldsymbol{w}\|_0, \ \beta \geq 0,$$

where $\|\boldsymbol{w}\|_0$ is the number of non-zero elements in $\boldsymbol{w}$. Again, this is intractable, so we tried to use a convex relaxation: L_1 regularization. That is,

$$\min_{\boldsymbol{w}} \quad \sum_i L(\boldsymbol{x}_i^\top \boldsymbol{w}, y_i) + \beta\|\boldsymbol{w}\|_1, \ \beta \geq 0. \tag{2.2}$$

Recall that the L_1 norm of $\boldsymbol{w}$ is convex. Moreover, as long as the loss function is convex in $\boldsymbol{w}$, the objective function remains convex in $\boldsymbol{w}$, and thus solvable using subgradient descent.

Subgradient

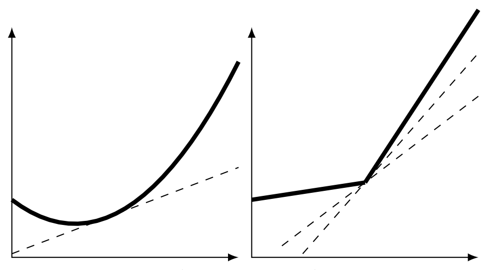
Suppose $f(\cdot)$ is a convex function. If it is differentiable, we have

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^\top \nabla f(\boldsymbol{x}_0),$$

where $\nabla f(\boldsymbol{x}_0)$ is the **gradient** of f at $\boldsymbol{x}_0$.
A **subgradient** at $\boldsymbol{x}_0$ is any $\boldsymbol{d}_0$ such that

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^\top \boldsymbol{d}_0.$$

If $f(\cdot)$ is differentiable at $\boldsymbol{x}_0$, then $\boldsymbol{d}_0$ is unique and $\boldsymbol{d}_0 = \nabla f(\boldsymbol{x}_0)$. However, if $f(\cdot)$ is not differentiable at $\boldsymbol{x}_0$, then $\boldsymbol{d}_0$ is not unique (see the figure below).



Gradient and (non-unique) Subgradient.

Let $F(\boldsymbol{w})$ be our objective:

$$F(\boldsymbol{w}) = L(\boldsymbol{w}) + \beta\|\boldsymbol{w}\|_1 = \sum_i L\left(\boldsymbol{x}_i^\top \boldsymbol{w}, y_i\right) + \beta\|\boldsymbol{w}\|_1.$$

Consider a descent step from $\boldsymbol{w}^{(k)}$, the parameters in the kth iteration. The partial derivative w.r.t. $w_j^{(k)}$ is[1]

$$\begin{aligned}\frac{\partial F}{\partial w_j^{(k)}} &= \frac{\partial L}{\partial w_j^{(k)}} + \beta \cdot \operatorname{sign}\left(w_j^{(k)}\right) \\ &= \sum_i L'\left(\boldsymbol{x}_i^\top \boldsymbol{w}^{(k)}, y_i\right) X_{ij} + \beta \cdot \operatorname{sign}\left(w_j^{(k)}\right), \quad \text{if } w_j^{(k)} \neq 0.\end{aligned} \tag{2.3}$$

Then to iterate:

$$w_j^{(k+1)} = w_j^{(k)} - \eta \frac{\partial F}{\partial w_j^{(k)}}$$

with $\eta > 0$ as step size.

What if $w_j^{(k)} = 0$? Let's suppose $w_j^{(k)} = 0$ and look at Eq.(2.3). If $\left|\partial L/\partial w_j^{(k)}\right| = \left|\sum_i L'(\boldsymbol{x}_i^\top \boldsymbol{w}, y_i) X_{ij}\right| < \beta$, then w_j remains at 0, otherwise we can reduce the objective by moving w_j away from 0, to the direction of $\left(-\partial L/\partial w_j^{(k)}\right)$. To see this, we assume $w_j^{(k)} = 0$ and consider the difference of two consecutive objective values *when w_j moves away from 0*. For smooth loss (where the following approximation is valid in a small neighbourhood of $\boldsymbol{w}^{(k)}$),

$$\begin{aligned}L\left(\boldsymbol{w}^{(k+1)}\right) &\approx L\left(\boldsymbol{w}^{(k)}\right) + \left(\nabla L(\boldsymbol{w}^{(k)})\right)^\top \left(\boldsymbol{w}^{(k+1)} - \boldsymbol{w}^{(k)}\right) \\ &= L\left(\boldsymbol{w}^{(k)}\right) - \eta \cdot \nabla L\left(\boldsymbol{w}^{(k)}\right)^\top \nabla L\left(\boldsymbol{w}^{(k)}\right) \text{ when } \boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta \nabla L\left(\boldsymbol{w}^{(k)}\right) \\ &= L\left(\boldsymbol{w}^{(k)}\right) - \eta \left\|\nabla L\left(\boldsymbol{w}^{(k)}\right)\right\|_2^2\end{aligned}$$

We can see that contribution of the jth component to the difference in loss is (approximately) $-\eta\left(\partial L/\partial w_j^{(k)}\right)^2$. For the regularizer, the difference is $\beta\eta\left|\partial L/\partial w_j^{(k)}\right|$ (changing from $w_j^{(k)} = 0$ to $w_j^{(k+1)} = -\eta \partial L/\partial w_j^{(k)}$). So the total contribution to the difference in objectives is

$$-\eta\left(\frac{\partial L}{\partial w_j^{(k)}}\right)^2 + \beta\eta\left|\frac{\partial L}{\partial w_j^{(k)}}\right|.$$

- If $\left|\partial L/\partial w_j^{(k)}\right| < \beta$, this quantity is positive. w_j moving away from 0 will increase the objective.
- If $\left|\partial L/\partial w_j^{(k)}\right| > \beta$, this quantity is negative. w_j moving towards $\left(-\partial L/\partial w_j^{(k)}\right)$ will reduce the objective.

Therefore, larger β indicates more zeros in $\boldsymbol{w}$. This sparse solution implicitly select features since $w_j = 0$ means the jth feature is unused.

If the loss function $L(\cdot,\cdot)$ is convex in the first argument, then the L_1 regularized problem (2.2) can be solved efficiently. To solve the problem, we introduce a slack variable $\boldsymbol{\xi} \in \mathbb{R}^n$ and reformulate the problem as

$$\begin{aligned}\min_{\boldsymbol{w},\boldsymbol{\xi}} \quad & \sum_i L(\boldsymbol{x}_i^\top \boldsymbol{w}, y_i) + \beta \mathbf{1}_n^\top \boldsymbol{\xi} \\ \text{s.t.} \quad & \boldsymbol{\xi} \succeq \boldsymbol{w} \\ & \boldsymbol{\xi} \succeq -\boldsymbol{w}\end{aligned}$$

For example, if $L(\widehat{y}, y) = (\widehat{y} - y)^2$, the above optimization is a quadratic programming; if $L(\widehat{y}, y) = |\widehat{y} - y|$, it is a linear programming.

[1]We write $\left.\frac{\partial F}{\partial w_j}\right|_{w_j = w_j^{(k)}}$ as $\frac{\partial F}{\partial w_j^{(k)}}$

Chapter 3

Output Transformation

3.1 Transform Function

So far we have talked about regression task, where the output $y \in \mathbb{R}$ is a real number. In some cases, we need a special type of output. For instances, we may need non-negative output $y \geq 0$, probability output $y \in [0, 1]$, or class indicator $y \in \{-1, +1\}$. For these special outputs, we can choose a transform function $f : \mathbb{R} \mapsto \mathcal{Y}$, such that the range of $f(\cdot)$ is $\mathcal{Y}$, the desired output domain.

Formally speaking, let $\widehat{z} = \boldsymbol{w}^T \boldsymbol{x}$ be the *pre-prediction* and $\widehat{y} = f(\widehat{z})$ is the *post-prediction*.

- For $y \geq 0$, $f_1(\widehat{z}) = e^{\widehat{z}}$. That is, $\widehat{y} = e^{\boldsymbol{w}^T \boldsymbol{x}}$.
- For $y \in [0, 1]$, $f_2(\widehat{z}) = \sigma(\widehat{z}) = (1 + e^{-\widehat{z}})^{-1}$, where $\sigma(\cdot)$ is the so-called sigmoid function in the literature. So, $\widehat{y} = (1 + e^{-\boldsymbol{w}^T \boldsymbol{x}})^{-1}$.
- For $y \in \{-1, +1\}$, $f_3(\widehat{z}) = \text{sign}(\widehat{z})$, which is the sign function. So, $\widehat{y} = \text{sign}(\boldsymbol{w}^T \boldsymbol{x})$. Technically, when $\widehat{z} = 0$, $\widehat{y}$ becomes 0, so we need to specify this particular case as $\widehat{y} = -1$ or $\widehat{y} = +1$.

Figure 3.1 shows the above transform functions.

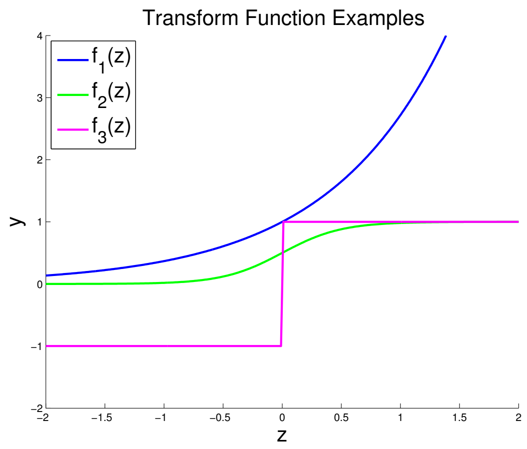


Figure 3.1: Transform functions examples.

3.2 Matching Loss

Once we set up the function to transform the pre-prediction to our desired domain, we can perform training as usual. However, we must adjust the loss function accordingly. If a specific $f(\cdot)$ is combined with arbitrary loss, exponentially many local minima may be created[1]. This means arbitrary combination will not produce convex formulation for optimization. To effectively solve the training problem, we need to specify a matching loss for the transform function $f(\cdot)$.

[1]Peter Auer, Mark Herbster, Manfred K Warmuth, et al. Exponentially many local minima for single neurons. Advances in neural information processing systems, pages 316–322, 1996.

Given any *differentiable* and *strictly increasing* function $f(\cdot)$, define the matching loss $L(\widehat{y}, y)$ as

$$\begin{aligned}L(\widehat{y}, y) &= L\left(f(\widehat{z}), f(z)\right)\\&= \int_z^{\widehat{z}} (f(u) - f(z))\,\mathrm{d}u\\&= F(u)|_z^{\widehat{z}} - f(z)u|_z^{\widehat{z}}\\&= F(\widehat{z}) - F(z) - f(z)(\widehat{z} - z)\end{aligned} \tag{3.1}$$

where $F'(\cdot) = f(\cdot)$. Note that since $F(\cdot)$ is strictly convex ($F''(u) = f'(u) > 0$ because $f(\cdot)$ is strictly increasing), $L(\widehat{y}, y)$ is strictly convex in $\widehat{z}$, and hence, in $\boldsymbol{w}$. Because of convexity,

$$F(\widehat{z}) \geq F(z) + f(z)(\widehat{z} - z) \Longrightarrow L(\widehat{y}, y) \geq 0.$$

Moreover,

$$L(f(\widehat{z}), f(z)) = 0 \iff \widehat{z} = z.$$

Bregman Divergence

Eq.(3.1) is also known as *Bregman Divergence* $D_F(\widehat{z}, z)$.

Now let us look at our previous examples.

- Identity transform: $y = f(z) = z, F(z) = \frac{1}{2}z^2$.

$$\begin{aligned}L(f(\widehat{z}), f(z)) &= F(\widehat{z}) - F(z) - f(z)(\widehat{z} - z)\\&= \frac{1}{2}\widehat{z}^2 - \frac{1}{2}z^2 - z(\widehat{z} - z)\\&= \frac{1}{2}\widehat{z}^2 - \widehat{z}z + \frac{1}{2}z^2\\&= \frac{1}{2}(\widehat{z} - z)^2\\&= \frac{1}{2}(\widehat{y} - y)^2 \quad \text{(squared loss, } L_2 \text{ loss).}\end{aligned}$$

- Exponent transform: $y = f_1(z) = e^z, F_1(z) = e^z$.

$$\begin{aligned}L(f(\widehat{z}), f(z)) &= F(\widehat{z}) - F(z) - f(z)(\widehat{z} - z)\\&= e^{\widehat{z}} - e^z - e^z(\widehat{z} - z)\\&= y \ln \frac{y}{\widehat{y}} + \widehat{y} - y \quad \text{(un-normalized entropy).}\end{aligned}$$

- Sigmoid transform: $y = f_2(z) = \sigma(z), f_2^{-1}(y) = \ln \frac{y}{1-y}, F_2(z) = \ln(e^z + 1)$.

$$\begin{aligned}L(f(\widehat{z}), f(z)) &= F(\widehat{z}) - F(z) - f(z)(\widehat{z} - z)\\&= \ln(e^{\widehat{z}} + 1) - \ln(e^z + 1) - \sigma(z)(\widehat{z} - z)\\&= y \ln \frac{y}{\widehat{y}} + (1 - y) \ln \frac{1 - y}{1 - \widehat{y}} \quad \text{(cross entropy).}\end{aligned}$$

Sadly, we do not have "the" matching loss for sign$(\cdot)$ transform function, because it is not differentiable, and not strictly increasing. One intuitive alternative is to count how many instances we misclassified, using an indicator loss function:

$$L(\widehat{y}, y) = \mathbb{1}(\widehat{y} \neq y) \tag{3.2}$$

where $\mathbb{1}(A)$ is one when A is true. Assume $\widehat{y} = \text{sign}(\boldsymbol{w}^T\boldsymbol{x} - b)$ (previously we ignore b by augmenting $\boldsymbol{x}$), then the problem becomes

$$\begin{aligned}\min_{\boldsymbol{w},b} \quad & \sum_{i=1}^{t} \mathbb{1}(\widehat{y}_i \neq y_i)\\ \text{s.t.} \quad & \widehat{\boldsymbol{y}} = \text{sign}(X\boldsymbol{w} - b\mathbf{1}_t)\end{aligned}$$

It is not surprising that the above problem is not tractable. However, if the dataset is linearly separable, we can run linear programming to solve it, by introducing slack variable $\delta \in \mathbb{R}$:

$$\begin{aligned}\max_{\boldsymbol{w},b,\delta} \quad & \delta\\ \text{s.t.} \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i - b) \geq \delta \quad \forall i\\ & \delta \leq 1.\end{aligned}$$

The last constraint ($\delta \leq 1$) is necessary because otherwise we can amplify $\boldsymbol{w}$, b and δ simultaneously and make the objective value go to infinity. Note that it is solvable even when the dataset is not linearly separable. For the attained solution, the dataset is

- linearly separable if and only if $\delta = 1$.
- not linearly separable if and only if $\delta < 0$.

Another linear programming formulation

Introduce slack variable $\boldsymbol{\xi} \in \mathbb{R}^t$,

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \boldsymbol{\xi}^T \mathbf{1}_t \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^T \boldsymbol{x}_i - b) \geq 1 - \xi_i \quad \forall i \\ & \boldsymbol{\xi} \preceq \mathbf{0}_t. \end{aligned}$$

After solving the problem, the dataset is

- linearly separable if and only if $\boldsymbol{\xi} = \mathbf{0}_t$, the vector of size t with all zeros.
- not linearly separable if and only if $\exists i, \xi_i < 0$.

3.3 Surrogate Loss for Classification

To overcome the difficulty of non-continuous sign transform, one can apply two standard approaches. The first is to approximate the sign function with tanh function; the second is to approximate the loss with surrogate loss.

The tanh function is defined as

$$\widehat{y} = \tanh(\widehat{z}) = \frac{e^{\widehat{z}} - e^{-\widehat{z}}}{e^{\widehat{z}} + e^{-\widehat{z}}}.$$

Luckily, this replacement recovers differentiability and strictly increasing property, so we can construct its matching loss and hence, convex objective. However, the transformed output $\widehat{y}$ is in the range of $[-1, 1]$, which is a "soft" classification, rather than $\{-1, +1\}$, the "hard" classification. It does not automatically guarantee good approximation of minimum classification error.

The second approach is to retain the sign transform, but use some surrogate losses. The problem is, how to find effective approximation of the indicator loss (3.2) that is efficiently computable? To illustrate alternative approximations, let's first define a new concept, the margin.

Definition 3. *The **margin** m is the product of true output (label) y and pre-prediction $\widehat{z}$. That is $m = y\widehat{z}$.*

Note that $m > 0$ indicates $\text{sign}(\widehat{z}) = y$ and the instance is correctly classified; while $m < 0$ indicates $\text{sign}(\widehat{z}) \neq y$ and the instance is misclassified. Therefore, we can try some surrogate losses based on m. For example, the squared loss

$$\begin{aligned} L(m) &= (\boldsymbol{w}^T \boldsymbol{x} - y)^2 \\ &= (\widehat{z} - y)^2 \\ &= \widehat{z}^2 - 2\widehat{z}y + y^2 \\ &= m^2 - 2m + 1 \\ &= (m - 1)^2. \end{aligned}$$

The derivation is based on the fact that $y^2 = 1$ for all instances. Now we can see the severe drawback of this loss. It penalizes the model when the outcome is too correct ($m > 1$), which does not make sense in the context of classification ($\widehat{y} = \text{sign}(\boldsymbol{w}^T \boldsymbol{x})$). Therefore, here we list some more reasonable choices as surrogate loss, and their illustrative plots are shown in Figure 3.2.

- Exponential loss: $L(m) = e^{-m}$. Obviously, this surrogate loss is convex in $\widehat{z}$, so the training is efficient. However, it is unbounded, grows exponentially when the model misclassifies an instance ($m < 0$), and as a result, not robust.

- Binomial deviance: $L(m) = \ln(1+e^{-m})$. It is more robust compared with exponential loss because as $m \to -\infty$, the slope approaches -1. However, it is *not* an upper bound of the indicator loss, see Figure 3.2.

- Hinge loss: $L(m) = \max(0, 1 - m)$. This loss is piece-wise linear, so we can solve it using linear programming. Moreover, the growth rate is fixed (-1) for $m < 1$. In fact, this surrogate loss is widely adopted in the classification literature.

- Robust loss: $L(m) = 1 - \tanh(m)$. As shown in Figure 3.2, this surrogate loss is bounded. Therefore the model does not trace outlier much and is very robust. Unfortunately, unlike previous surrogate losses, this one is not convex in $\widehat{z}$.

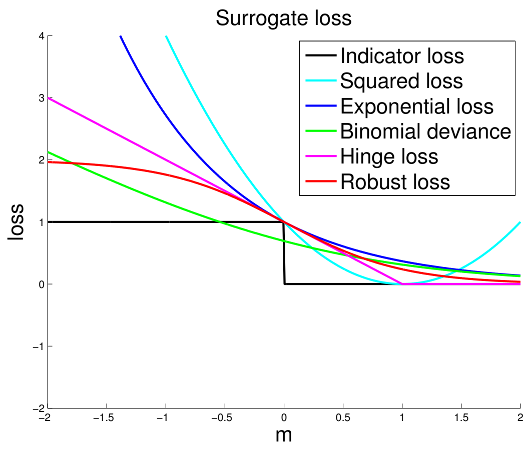


Figure 3.2: Surrogate loss.

Given convex surrogate loss, efficient minimization is guaranteed and extensions such as feature expansion, kernelization (L_2 regularization) and feature selection (L_1 regularization) are retained.

Chapter 4

Support Vector Machine

4.1 Large Margin Classification

In this section, we talk about how to effectively find a classifier from geometrical point of view. As a threshold example, consider a dataset of classification that is linearly separable. Probably, there are many linear classifiers ($(\boldsymbol{w}, b)$ pairs) that are consistent with the dataset as shown in Figure 4.1.

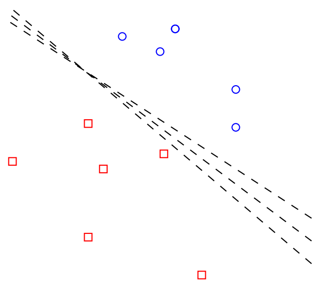


Figure 4.1: Linearly separable dataset, where squares and circles are from different classes and dashed lines are possible line classifiers that are consistent with the dataset.

Are some of them better than the others? Intuitively, the one that maximizes the margin probably generalize better, that is, the linear classifier that *maximizes the minimum distance between data points and decision hyperplane.* As shown in Figure 4.2.

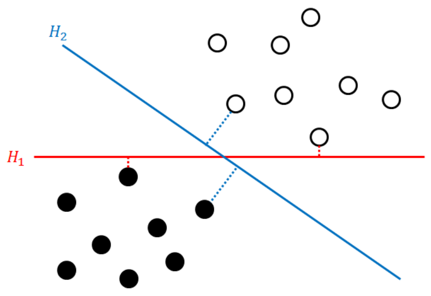


Figure 4.2: Large margin linear classifier. Both H_1 and H_2 are consistent with the dataset, but H_2 has larger margin than H_1.

To maximize such margin, we first need to know how to compute it efficiently. Recall that the classification hyperplane is defined as $\{\boldsymbol{x} | \boldsymbol{w}^\top \boldsymbol{x} - b = 0\}$. The distance between a point $\boldsymbol{x}_i$ and the hyperplane is

$$\delta_i = \frac{|\boldsymbol{w}^\top \boldsymbol{x}_i - b|}{\|\boldsymbol{w}\|_2}.$$

Recall that $y_i \in \{-1, +1\}$ for binary classification, so we can get rid of the absolute operation:

$$\delta_i = \frac{y_i(\boldsymbol{w}^\top \boldsymbol{x}_i - b)}{\|\boldsymbol{w}\|_2}.$$

Therefore, formally speaking, to find the maximum margin classifier, we solve the following problem:

$$
\begin{aligned}
& \max_{\boldsymbol{w},b} \min_i \quad \delta_i \\
\iff & \max_{\boldsymbol{w},b,\gamma} \quad \gamma \quad \text{s.t.} \quad \frac{y_i(\boldsymbol{w}^\top \boldsymbol{x}_i - b)}{\|\boldsymbol{w}\|_2} \geq \gamma \quad \forall i \\
\iff & \max_{\boldsymbol{w},b,\gamma} \quad \gamma \quad \text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i - b) \geq \gamma \|\boldsymbol{w}\|_2 \quad \forall i,
\end{aligned}
$$

where γ serves as a universal (for all i) lower bound of margin. Note that the last optimization is under-constrained: we can rescale $\boldsymbol{w}, b$ and γ simultaneously so that the objective value goes to infinity. Hence, we impose one more constraint, which is $\gamma = \frac{1}{\|\boldsymbol{w}\|_2}$ and optimize

$$
\begin{aligned}
& \max_{\boldsymbol{w},b} \quad \frac{1}{\|\boldsymbol{w}\|_2} \quad \text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i - b) \geq 1, \forall i \\
\iff & \min_{\boldsymbol{w},b} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 \quad \text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i - b) \geq 1, \forall i,
\end{aligned}
$$

which is a quadratic programming.

The above derivation is based on the fact that the dataset is linearly separable. Actually, after solving the quadratic programming, we will find that only a few data points are critical to determine the hyperplane's parameters. These points are called *support vectors*, since they are closest to the hyperplane and hence, "support" it.

support vector

Definition 4. ***Support vectors*** *are subset of training points that are closest to the optimal hyperplane defined by* $\boldsymbol{w}, b$.

Mathematically, support vectors are the points that satisfy the equality in the constraints $y_i(\boldsymbol{w}^\top \boldsymbol{x}_i - b) = 1$. In fact, removing data other than support vectors does not change the optimal solution. The resultant classifier is called *support vector machine* (SVM).

support vector machine

What if the dataset is not linearly separable? In this case, there is no linear classifier that is consistent with the dataset. The previous quadratic programming is infeasible (no solution can be found). A standard (heuristic) approach is to introduce slack variable ξ_i to each margin constraint and at the same time, try to minimize minimum margin at the price of sum of slacks. Formally,

$$
\begin{aligned}
\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{\beta}{2}\|\boldsymbol{w}\|_2^2 + \boldsymbol{\xi}^\top \mathbf{1}_t \\
\text{s.t.} \quad & y_i(\boldsymbol{w}^\top \boldsymbol{x}_i - b) \geq 1 - \xi_i, \forall i \\
& \boldsymbol{\xi} \succeq \mathbf{0}_t
\end{aligned}
\tag{4.1}
$$

$\beta > 0$ is a tuning trade-off parameter. ξ_i allows the classifier to "give up" some noisy points and introduces a "cost" for such slackness in the objective (see Figure 4.3). This is called soft support vector machine.

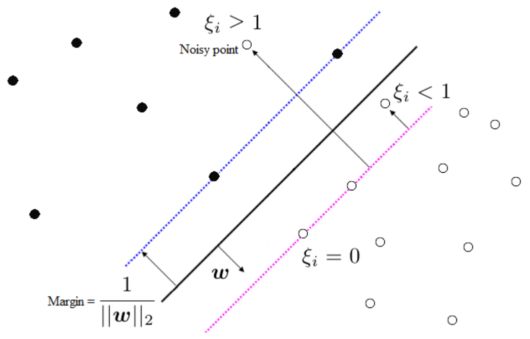


Figure 4.3: Soft SVM with slack variable.

One interesting fact is that if we treat $\frac{\beta}{2}\|\boldsymbol{w}\|_2^2$ as a regularizer, problem (4.1) is equivalent to using hinge loss as surrogate loss with L_2 regularization. The geometric approach leads to the same solution as discussed in previous chapter.

4.2 Duality

In this section, we revisit the concept of duality in optimization. Consider a constrained optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & l(\boldsymbol{w}) \\ \text{s.t.} \quad & A\boldsymbol{w} \succeq \boldsymbol{b} \\ & C\boldsymbol{w} = \boldsymbol{d}, \end{aligned}$$

which is called *primal problem.* Its *Lagrangian* is

Lagrangian

$$L(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = l(\boldsymbol{w}) + \boldsymbol{\lambda}^\top(\boldsymbol{b} - A\boldsymbol{w}) + \boldsymbol{\mu}^\top(\boldsymbol{d} - C\boldsymbol{w})$$

where $\boldsymbol{\lambda} \succeq \boldsymbol{0}_t$ and $\boldsymbol{\mu} \in \mathbb{R}^\top$ are Lagrangian multipliers for constraint violations. The *dual function* is defined as

dual function

$$\begin{aligned} D(\boldsymbol{\lambda}, \boldsymbol{\mu}) &= \min_{\boldsymbol{w}} \quad L(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \\ &= \min_{\boldsymbol{w}} \quad l(\boldsymbol{w}) + \boldsymbol{\lambda}^\top(\boldsymbol{b} - A\boldsymbol{w}) + \boldsymbol{\mu}^\top(\boldsymbol{d} - C\boldsymbol{w}) \end{aligned}$$

The dual function has two important properties:

- $D(\boldsymbol{\lambda}, \boldsymbol{\mu})$ is always concave in $\boldsymbol{\lambda}, \boldsymbol{\mu}$, because the min of linear functions (in $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$) is concave.
- $D(\boldsymbol{\lambda}, \boldsymbol{\mu})$ provides a lower bound on the objective of the primal problem.

Let

$$\begin{aligned} p^* &= \min_{\boldsymbol{w}} \quad l(\boldsymbol{w}) \quad \text{s.t.} \quad A\boldsymbol{w} \succeq \boldsymbol{b}, C\boldsymbol{w} = \boldsymbol{d} \\ \boldsymbol{w}^* &= \operatorname*{argmin}_{\boldsymbol{w}} \quad l(\boldsymbol{w}) \quad \text{s.t.} \quad A\boldsymbol{w} \succeq \boldsymbol{b}, C\boldsymbol{w} = \boldsymbol{d}. \end{aligned}$$

The following theorem states *weak duality*:

weak duality

Theorem 3. *For any $\boldsymbol{\lambda} \succeq \boldsymbol{0}_t, \boldsymbol{\mu}$, we have $D(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq p^*$.*

Proof. Fix any $\boldsymbol{\lambda} \succeq \boldsymbol{0}_t, \boldsymbol{\mu}$

$$\begin{aligned} D(\boldsymbol{\lambda}, \boldsymbol{\mu}) &= \min_{\boldsymbol{w}} \quad l(\boldsymbol{w}) + \boldsymbol{\lambda}^\top(\boldsymbol{b} - A\boldsymbol{w}) + \boldsymbol{\mu}^\top(\boldsymbol{d} - C\boldsymbol{w}) \\ &\leq l(\boldsymbol{w}^*) + \boldsymbol{\lambda}^\top(\boldsymbol{b} - A\boldsymbol{w}^*) + \boldsymbol{\mu}^\top(\boldsymbol{d} - C\boldsymbol{w}^*) \\ &\leq l(\boldsymbol{w}^*) = p^* \end{aligned}$$

For the first inequality, $\boldsymbol{w}^*$ is a particular choice of $\boldsymbol{w}$, so its objective value must be greater than (or equal to, if $\boldsymbol{w}^*$ is indeed the minimizer in the Lagrangian) that of the dual function. For the second inequality, note that $\boldsymbol{w}^*$ is feasible so $A\boldsymbol{w}^* \succeq \boldsymbol{b}$ and $C\boldsymbol{w}^* = \boldsymbol{d}$. As a result, we have $\boldsymbol{\lambda}^\top(\boldsymbol{b} - A\boldsymbol{w}^*) \leq 0$ and $\boldsymbol{\mu}^\top(\boldsymbol{d} - C\boldsymbol{w}^*) = 0$ since $\boldsymbol{\lambda} \succeq \boldsymbol{0}_t$. □

Now we define the *dual problem* (in contrast to the primal problem) as

$$\begin{aligned} & \max_{\boldsymbol{\lambda} \succeq \boldsymbol{0}_t, \boldsymbol{\mu}} \quad D(\boldsymbol{\lambda}, \boldsymbol{\mu}) \\ \iff & \max_{\boldsymbol{\lambda} \succeq \boldsymbol{0}_t, \boldsymbol{\mu}} \quad \min_{\boldsymbol{w}} \quad l(\boldsymbol{w}) + \boldsymbol{\lambda}^\top(\boldsymbol{b} - A\boldsymbol{w}) + \boldsymbol{\mu}^\top(\boldsymbol{d} - C\boldsymbol{w}). \end{aligned}$$

Sometimes we can solve the inner minimization over $\boldsymbol{w}$ analytically to get a closed form of $D(\boldsymbol{\lambda}, \boldsymbol{\mu})$. Notice that since $D(\boldsymbol{\lambda}, \boldsymbol{\mu})$ is concave and thus the outer maximization is efficiently solvable, even when the primal problem is not convex.

Let

$$\begin{aligned} d^* &= \max_{\boldsymbol{\lambda} \succeq \boldsymbol{0}_t, \boldsymbol{\mu}} \quad D(\boldsymbol{\lambda}, \boldsymbol{\mu}) \\ (\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) &= \operatorname*{argmax}_{\boldsymbol{\lambda} \succeq \boldsymbol{0}_t, \boldsymbol{\mu}} \quad D(\boldsymbol{\lambda}, \boldsymbol{\mu}). \end{aligned}$$

The following theorem states *strong duality*:

strong duality

Theorem 4 (ref)**.** *If $l(\boldsymbol{w})$ is convex in $\boldsymbol{w}$ and $A\boldsymbol{w} \succeq \boldsymbol{b}, C\boldsymbol{w} = \boldsymbol{d}$ are strictly feasible (i.e., there exists $\boldsymbol{w}_0$ such that both constraints are satisfied), then $d^* = p^*$.*

The strong duality guarantees that the optimal objective value of the dual problem is identical to that of the primal problem. Therefore, the efficiently computable dual problem is a bridge to find the solution to the primal problem. The remaining problem is how to recover $\boldsymbol{w}^*$ from $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$. The following theorem provides an effective approach to recover $\boldsymbol{w}^*$.

Theorem 5. *If strong duality holds, then*

$$\max_{\boldsymbol{\lambda}\succeq\mathbf{0}_t,\boldsymbol{\mu}} \min_{\boldsymbol{w}} L(\boldsymbol{w},\boldsymbol{\lambda},\boldsymbol{\mu}) \quad = \quad \min_{\boldsymbol{w}} \max_{\boldsymbol{\lambda}\succeq\mathbf{0}_t,\boldsymbol{\mu}} L(\boldsymbol{w},\boldsymbol{\lambda},\boldsymbol{\mu}).$$

Proof. We first prove a lemma. Note that for any feasible $\boldsymbol{w}$ (i.e., $A\boldsymbol{w} \geq \boldsymbol{b}$ and $C\boldsymbol{w} = \boldsymbol{d}$), we have

$$l(\boldsymbol{w}) \geq \max_{\boldsymbol{\lambda}\succeq\mathbf{0}_t,\boldsymbol{\mu}} l(\boldsymbol{w}) + \boldsymbol{\lambda}^\top(\boldsymbol{b}-A\boldsymbol{w}) + \boldsymbol{\mu}^\top(\boldsymbol{d}-C\boldsymbol{w}) = \max_{\boldsymbol{\lambda}\succeq\mathbf{0}_t,\boldsymbol{\mu}} L(\boldsymbol{w},\boldsymbol{\lambda},\boldsymbol{\mu}). \tag{4.2}$$

This is because the term $\boldsymbol{\lambda}^\top(\boldsymbol{b}-A\boldsymbol{w})$ is always non-negative given the $\boldsymbol{w}$ and chosen $\boldsymbol{\lambda}$. If $\boldsymbol{w}^*$ is feasible and minimizes $l(\boldsymbol{w})$, then Eq.(4.2) implies

$$l(\boldsymbol{w}^*) \geq \max_{\boldsymbol{\lambda}\succeq\mathbf{0}_t,\boldsymbol{\mu}} L(\boldsymbol{w}^*,\boldsymbol{\lambda},\boldsymbol{\mu}) \geq \min_{\boldsymbol{w}} \max_{\boldsymbol{\lambda}\succeq\mathbf{0}_t,\boldsymbol{\mu}} L(\boldsymbol{w},\boldsymbol{\lambda},\boldsymbol{\mu}).$$

This inequality means

$$\begin{aligned} \min_{\boldsymbol{w}:A\boldsymbol{w}\succeq\boldsymbol{b},C\boldsymbol{w}=\boldsymbol{d}} l(\boldsymbol{w}) &\geq \min_{\boldsymbol{w}:A\boldsymbol{w}\succeq\boldsymbol{b},C\boldsymbol{w}=\boldsymbol{d}} \max_{\boldsymbol{\lambda}\succeq\mathbf{0}_t,\boldsymbol{\mu}} L(\boldsymbol{w},\boldsymbol{\lambda},\boldsymbol{\mu}) \\ &\geq \min_{\boldsymbol{w}} \max_{\boldsymbol{\lambda}\succeq\mathbf{0}_t,\boldsymbol{\mu}} L(\boldsymbol{w},\boldsymbol{\lambda},\boldsymbol{\mu}) \end{aligned} \tag{4.3}$$

Now we are ready to prove the theorem:

$$\begin{aligned} \min_{\boldsymbol{w}} \max_{\boldsymbol{\lambda}\succeq\mathbf{0}_t,\boldsymbol{\mu}} L(\boldsymbol{w},\boldsymbol{\lambda},\boldsymbol{\mu}) &\geq \max_{\boldsymbol{\lambda}\succeq\mathbf{0}_t,\boldsymbol{\mu}} \min_{\boldsymbol{w}} L(\boldsymbol{w},\boldsymbol{\lambda},\boldsymbol{\mu}) && \text{(by max-min inequality[1])} \\ &= \min_{\boldsymbol{w}:A\boldsymbol{w}\succeq\boldsymbol{b},C\boldsymbol{w}=\boldsymbol{d}} l(\boldsymbol{w}) && \text{(by strong duality)} \\ &\geq \min_{\boldsymbol{w}} \max_{\boldsymbol{\lambda}\succeq\mathbf{0}_t,\boldsymbol{\mu}} L(\boldsymbol{w},\boldsymbol{\lambda},\boldsymbol{\mu}) && \text{(by the lemma 4.3)} \end{aligned} \tag{4.4}$$

Because the LHS (left-hand-side) is the same as the last row, the inequalities in between must be equalities. Therefore the claim is proved. □

complementary slackness

Let $\boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*$ be the optimal solutions of the primal/dual problems. The theorem above implies *complementary slackness* property:

- if $\lambda_i^* > 0$, then $A_{i:}\boldsymbol{w}^* = b_i$;
- if $A_{i:}\boldsymbol{w}^* > b_i$, then $\lambda_i^* = 0$.

To see this, note that:

$$\begin{aligned} l(\boldsymbol{w}^*) &= \max_{\boldsymbol{\lambda}\succeq\mathbf{0}_t,\boldsymbol{\mu}} \min_{\boldsymbol{w}} L(\boldsymbol{w},\boldsymbol{\lambda},\boldsymbol{\mu}) && \text{(from Eq.(4.4))} \\ &= D(\boldsymbol{\lambda}^*,\boldsymbol{\mu}^*) && \text{(definition of } D) \\ &= \min_{\boldsymbol{w}} L(\boldsymbol{w},\boldsymbol{\lambda}^*,\boldsymbol{\mu}^*) && \text{(definition of } D) \\ &\leq L(\boldsymbol{w}^*,\boldsymbol{\lambda}^*,\boldsymbol{\mu}^*) && (\boldsymbol{w}^* \text{ may not attain the } \min_{\boldsymbol{w}}) \\ &= l(\boldsymbol{w}^*) + \boldsymbol{\lambda}^{*\top}(\boldsymbol{b}-A\boldsymbol{w}^*) + \boldsymbol{\mu}^{*\top}(\boldsymbol{d}-C\boldsymbol{w}^*) \\ &\leq l(\boldsymbol{w}^*) \end{aligned}$$

The last inequality is because $\boldsymbol{\lambda}^* \succeq \mathbf{0}_t$ and $A\boldsymbol{w}^* \succeq \boldsymbol{b}$ so $\boldsymbol{\lambda}^{*\top}(\boldsymbol{b}-A\boldsymbol{w}^*)$ must be non-positive. However, the LHS and the last row are again the same, so equality holds in between. Specifically, this indicates $\lambda_i^*(b_i - A_{i:}\boldsymbol{w}^*) = 0$ and as a result, complementary slackness follows.

Now, back to our question on finding $\boldsymbol{w}^*$ given $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$. With the help of complementary slackness property, we can find $\boldsymbol{w}^*$ such that $\widetilde{A}\boldsymbol{w}^* = \widetilde{\boldsymbol{b}}, C\boldsymbol{w}^* = \boldsymbol{d}$, where $\widetilde{A}, \widetilde{\boldsymbol{b}}$ are built from the constraints with $\lambda_i^* > 0$.

[1]`https://en.wikipedia.org/wiki/Max-min_inequality`

4.3 Application of Duality: SVM

Recall the soft SVM optimization (hinge loss with L_2 regularizer):

$$\min_{\boldsymbol{w},b} \quad \frac{\beta}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{\top} \max\left(0, 1 - y_i\left(\boldsymbol{w}^\top \boldsymbol{x}_i - b\right)\right)$$

which, in matrix form, is equivalent to

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{\beta}{2}\|\boldsymbol{w}\|_2^2 + \boldsymbol{\xi}^\top \mathbf{1}_t \\ \text{s.t.} \quad & \boldsymbol{\xi} \succeq \mathbf{1}_t - \Delta(\boldsymbol{y})(X\boldsymbol{w} - b\mathbf{1}_t) \\ & \boldsymbol{\xi} \succeq \mathbf{0}_t, \end{aligned}$$

where $\Delta(\boldsymbol{y})$ is diagonalized matrix of $\boldsymbol{y}$. Introduce the Lagrange function

$$\begin{aligned} L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\mu}) &= \frac{\beta}{2}\boldsymbol{w}^\top \boldsymbol{w} + \boldsymbol{\xi}^\top \mathbf{1}_t + \boldsymbol{\mu}^\top(\mathbf{0}_t - \boldsymbol{\xi}) + \boldsymbol{\lambda}^\top[\mathbf{1}_t - \Delta(\boldsymbol{y})(X\boldsymbol{w} - b\mathbf{1}_t) - \boldsymbol{\xi}] \\ &= \frac{\beta}{2}\boldsymbol{w}^\top \boldsymbol{w} + \boldsymbol{\xi}^\top(\mathbf{1}_t - \boldsymbol{\mu} - \boldsymbol{\lambda}) + \boldsymbol{\lambda}^\top[\mathbf{1}_t - \Delta(\boldsymbol{y})(X\boldsymbol{w} - b\mathbf{1}_t)]. \end{aligned}$$

where $\boldsymbol{\lambda} \succeq \mathbf{0}_t, \boldsymbol{\mu} \succeq \mathbf{0}_t$ are Lagrangian multipliers for constraint violations. Now, we find the dual problem analytically. Given fixed $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, consider the minimization over $\boldsymbol{w}, b, \boldsymbol{\xi}$. We first eliminate $\boldsymbol{\xi}$:

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{\xi}} &= \mathbf{1}_t - \boldsymbol{\mu} - \boldsymbol{\lambda} = 0 \\ \Longrightarrow \boldsymbol{\mu} &= \mathbf{1}_t - \boldsymbol{\lambda} \\ \Longrightarrow \boldsymbol{\lambda} &\preceq \mathbf{1}_t \quad (\text{as } \boldsymbol{\mu} \succeq \mathbf{0}_t) \end{aligned}$$

This eliminates $\boldsymbol{\xi}$ and $\boldsymbol{\mu}$ simultaneously:

$$L(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \frac{\beta}{2}\boldsymbol{w}^\top \boldsymbol{w} + \boldsymbol{\lambda}^\top[\mathbf{1}_t - \Delta(\boldsymbol{y})(X\boldsymbol{w} - b\mathbf{1}_t)] \qquad \mathbf{0}_t \preceq \boldsymbol{\lambda} \preceq \mathbf{1}_t.$$

Next, we eliminate b:

$$\begin{aligned} \frac{\partial L}{\partial b} &= \boldsymbol{\lambda}^\top \boldsymbol{y} = 0, \\ \Longrightarrow L(\boldsymbol{w}, \boldsymbol{\lambda}) &= \frac{\beta}{2}\boldsymbol{w}^\top \boldsymbol{w} + \boldsymbol{\lambda}^\top \mathbf{1}_t - \boldsymbol{\lambda}^\top \Delta(\boldsymbol{y}) X\boldsymbol{w} \qquad \mathbf{0}_t \preceq \boldsymbol{\lambda} \preceq \mathbf{1}_t, \boldsymbol{\lambda}^\top \boldsymbol{y} = 0. \end{aligned}$$

Finally, we eliminate $\boldsymbol{w}$:

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{w}} &= \beta \boldsymbol{w} - X^\top \Delta(\boldsymbol{y})\boldsymbol{\lambda} = 0 \\ \Longrightarrow \boldsymbol{w} &= \frac{1}{\beta} X^\top \Delta(\boldsymbol{y})\boldsymbol{\lambda}. \\ \Longrightarrow L(\boldsymbol{\lambda}) &= \boldsymbol{\lambda}^\top \mathbf{1}_t - \frac{1}{2\beta}\boldsymbol{\lambda}^\top \Delta(\boldsymbol{y}) X X^\top \Delta(\boldsymbol{y})\boldsymbol{\lambda} \qquad \mathbf{0}_t \preceq \boldsymbol{\lambda} \preceq \mathbf{1}_t, \boldsymbol{\lambda}^\top \boldsymbol{y} = 0. \end{aligned}$$

Therefore, the dual problem of the regularized hinge loss is

$$\begin{aligned} \max_{\boldsymbol{\lambda}} \quad & \boldsymbol{\lambda}^\top \mathbf{1}_t - \frac{1}{2\beta}\boldsymbol{\lambda}^\top \Delta(\boldsymbol{y}) X X^\top \Delta(\boldsymbol{y})\boldsymbol{\lambda} \\ \text{s.t.} \quad & \mathbf{0}_t \preceq \boldsymbol{\lambda} \preceq \mathbf{1}_t, \\ & \boldsymbol{\lambda}^\top \boldsymbol{y} = 0. \end{aligned} \tag{4.5}$$

This is a quadratic programming. Once we find the optimal $\boldsymbol{\lambda}^*$, the remaining issue is to recover the classifier $(\boldsymbol{w}^*, b^*)$ from it. $\boldsymbol{w}^*$ is easy to compute:

$$\boldsymbol{w}^* = \frac{1}{\beta} X^\top \Delta(\boldsymbol{y})\boldsymbol{\lambda}^*.$$

As for b^*, complementary slackness is required. For any $0 < \lambda_i^* < 1$, we have

$$\begin{aligned} \lambda_i^* > 0 &\Longrightarrow 1 - y_i(X_{i:}\boldsymbol{w}^* - b^*) - \xi_i^* = 0, \\ \lambda_i^* < 1 \Longrightarrow \mu_i^* > 0 &\Longrightarrow \xi_i^* = 0. \end{aligned}$$

Therefore,

$$\begin{aligned} & y_i(X_{i:}\boldsymbol{w}^* - b^*) = 1 \\ \implies & b^* = X_{i:}\boldsymbol{w}^* - y_i = \frac{1}{\beta} X_{i:} X^\top \Delta(\boldsymbol{y}) \boldsymbol{\lambda}^* - y_i. \end{aligned}$$

For test point $\boldsymbol{x}$,

$$\begin{aligned} \widehat{y} &= \operatorname{sign}(\boldsymbol{w}^{*T}\boldsymbol{x} - b^*) \\ &= \operatorname{sign}\left(\frac{1}{\beta} \boldsymbol{x}^\top X^\top \Delta(\boldsymbol{y}) \boldsymbol{\lambda}^* - b^* \right) \end{aligned}$$

Note that the above classification rule and problem (4.5) can be kernelized. All we need is the inner product $\boldsymbol{x}_i^\top \boldsymbol{x}_j$.

Chapter 5

Large Output

5.1 Multivariate Prediction

So far we have talked about single output prediction, where $y_i \in \mathbb{R}$ is a one-dimensional output. What if the target is a vector of multiple values, say $\boldsymbol{y}_i \in \mathbb{R}^k$? Consider the linear prediction $y_i = \boldsymbol{w}^\top \boldsymbol{x}_i$, because of multiple outputs, the linear model $\boldsymbol{w}$ becomes a matrix $W \in \mathbb{R}^{n\times k}$, whose rows correspond to features and columns correspond to outputs. That is $\boldsymbol{y}_i^\top = \boldsymbol{x}_i^\top W$. The matrix form is

$$\widehat{Y} = XW$$

where $X \in \mathbb{R}^{t\times n}$, $\widehat{Y} \in \mathbb{R}^{t\times k}$.

Square loss with multiple outputs

$$\begin{aligned}
&\min_W \quad \sum_{i=1}^{t}\sum_{j=1}^{k}(X_{i:}W_{:j} - Y_{ij})^2\\
\iff &\min_W \quad \mathrm{tr}[(XW-Y)^\top(XW-Y)]\\
\iff &\min_W \quad \mathrm{tr}(W^\top X^\top XW) - 2\mathrm{tr}(Y^\top XW) + \mathrm{tr}(Y^\top Y).
\end{aligned}$$

Setting its derivative with respect to W to zero:

$$\frac{d\mathrm{Obj}}{dW} = 2X^\top X - 2X^\top Y = 0 \Longrightarrow X^\top XW = X^\top Y.$$

When X is of full rank (i.e., features are not correlated when $t > n$), $X^\top X$ is invertible. Then

$$W^* = (X^\top X)^{-1}X^\top Y.$$

(In Matlab implementation, it is one-line code: $W = (X'X)\backslash(X'Y)$.)

Trace of matrix

Here we list some properties of matrix trace (assume proper sizes):

- $\mathrm{tr}(A) = \mathrm{tr}(A^\top)$;
- $\mathrm{tr}(A^\top B) = \sum_{ij} A_{ij}B_{ij}$;
- $\mathrm{tr}(\alpha A) = \alpha\mathrm{tr}(A), \alpha \in \mathbb{R}$;
- $\mathrm{tr}(ABC) = \mathrm{tr}(CAB) = \mathrm{tr}(BCA)$.

The derivative [a] of matrix trace:

- $\frac{d}{dW}\mathrm{tr}(C^\top W) = C$;
- $\frac{d}{dW}\mathrm{tr}(W^\top AW) = (A + A^\top)W$.

[a]One easy way to check the correctness of derivative is to see whether the resultant derivative is of the same size as the argument.

Now we investigate the properties we have before:

1. feature expansion;
2. L_2 regularization $\rightarrow$ kernelization;
3. L_1 regularization $\rightarrow$ sparsity;
4. output transformation and matching loss;
5. surrogate loss for classification.

5.1.1 Feature Expansion

Feature expansion of data point is the same as before: construct the expanded representation $\Phi \in \mathbb{R}^{t\times l}$, where l is the number of features. Accordingly, $W \in \mathbb{R}^{l\times k}$, and

$$\widehat{Y} = \Phi W.$$

5.1.2 Kernelization

Frobenius norm

Analogize to the L_2 regularizer, for matrix W, $R(W) = \sum_{ij} W_{ij}^2 = \text{tr}(W^\top W) = \|W\|_F^2$, where $\|\cdot\|_F$ is the *Frobenius norm*. The representer theorem holds: for

$$\min_W \quad \frac{\beta}{2}\text{tr}\left(W^\top W\right) + L(XW, Y),$$

the optimal $W^* = X^\top A$ for some $A \in \mathbb{R}^{t\times k}$. Therefore, the adjoint form is

$$\min_A \quad \frac{\beta}{2}\text{tr}\left(A^\top X X^\top A\right) + L\left(XX^\top A, Y\right).$$

To predict the output of a new point $\boldsymbol{x}$,

$$\widehat{\boldsymbol{y}}^\top = \boldsymbol{x}^\top W^* = \boldsymbol{x}^\top X^\top A^*.$$

5.1.3 Feature Selection: Sparsity

As for feature selection, we want the rows of W to be zeros (remember rows of W correspond to feature, while columns of W correspond to output). Can we choose a matrix norm that is similar to the L_1 regularizer in single output formulation? Yes, we can. However, first we need to investigate several matrix norms and check their properties.

Matrix Norms

The definition of matrix norm $\|\cdot\|$ satisfies three properties

- $\|\alpha X\| = |\alpha|\|X\|, \alpha \in \mathbb{R}$;
- $\|X + Y\| \leq \|X\| + \|Y\|$;
- $\|X\| = 0 \iff X = 0$.

Convexity of matrix norm

The definition of matrix norm guarantees that a norm must be convex. For $0 \leq \alpha \leq 1$,

$$\begin{aligned}\|\alpha X + (1-\alpha)Y\| \leq & \|\alpha X\| + \|(1-\alpha)Y\| \\ = & |\alpha|\|X\| + |1-\alpha|\|Y\| \\ = & \alpha\|X\| + (1-\alpha)\|Y\|.\end{aligned}$$

Therefore, using matrix norm as regularizer maintains convexity as long as the loss $L(\cdot,\cdot)$ is convex.

Here we introduce three types of norms: vector induced norm, Schatten p-norm and block norm.

- **Vector induced norm**. Given vector norm $\|\boldsymbol{x}\|_p$ for $\boldsymbol{x} \in \mathbb{R}^n$, define

$$\|W\|_p = \max_{\boldsymbol{x} \neq \mathbf{0}_t} \frac{\|W\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p}.$$

Vector induced norm examples

$$\|W\|_1 = \max_j \|W_{:j}\|_1$$

is the maximum *column* absolute sum.

$$\|W\|_\infty = \max_i \|W_{i:}\|_1$$

is the maximum *row* absolute sum.

$$\|W\|_2 = \sigma_{\max}(W)$$

is the biggest singular value of W.

- **Schatten p-norm**. The Schatten p-norm is based on the singular values of a matrix, $\boldsymbol{\sigma}(W)$. It is defined as $\|\boldsymbol{\sigma}(W)\|_p$, which is the vector p-norm of singular value vector.

Schatten p-norm examples

Trace norm:

$$\|W\|_{\text{tr}} = \|\boldsymbol{\sigma}(W)\|_1.$$

Spectral norm:

$$\|W\|_{sp} = \|\boldsymbol{\sigma}(W)\|_\infty = \sigma_{\max}(W) = \|W\|_2.$$

Frobenius norm:

$$\|W\|_F = \|\boldsymbol{\sigma}(W)\|_2 = \sqrt{\text{tr}(W^\top W)} = \sqrt{\sum_{ij} W_{ij}^2}.$$

- **Block norm**. It is defined as

$$\|W\|_{p,q} = \left(\sum_i \left[\left(\sum_j W_{ij}^q\right)^{\frac{1}{q}}\right]^p\right)^{\frac{1}{p}}.$$

which looks complicated. In fact, assume $W \in \mathbb{R}^{n\times m}$, let $\boldsymbol{v} \in \mathbb{R}^n$, whose entries are $v_i = \|W_{i:}\|_q$, then $\|W\|_{p,q} = \|\boldsymbol{v}\|_p$. That is, $\|W\|_{p,q}$ first compute the q vector norm of rows, then compute p vector norm of the resultant column vector.

Block norm examples

$$\|W\|_{2,1} = \sum_i \|W_{i:}\|_2;$$

$$\|W\|_{2,2} = \|W\|_F.$$

To illustrate the properties of these norms, let's look at some concrete examples. Let

$$U = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, V = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}, W = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Table 5.1 summarizes the values of diverse norms.

Now, come back to our feature selection problem. We want sparse features. Remember what we want is to find a norm that penalizes less on matrix with sparse rows. Because U has the sparsest rows compared with V and W, so $\|\cdot\|_{2,1}$ norm is our choice. Therefore, for feature selection with multivariate output, we optimize

$$\min_W \quad \frac{\beta}{2}\|W\|_{2,1}^2 + L(XW, Y).$$

Table 5.1: Matrix norm concrete examples.

Norm	U	V	W
$\|\cdot\|_1$	1	2	1
$\|\cdot\|_\infty$	2	1	1
$\|\cdot\|_2$	$\sqrt{2}$	$\sqrt{2}$	1
$\|\cdot\|_{\text{tr}}$	$\sqrt{2}$	$\sqrt{2}$	2
$\|\cdot\|_F$	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{2}$
$\|\cdot\|_{2,1}$	$\sqrt{2}$	2	2

As a side note, sometimes we want to reduce not only the number of feature, but also the number of output. Say, we want to eliminate some of the outputs in $\boldsymbol{y}_i$ for conciseness. Formally, we want W with sparse rows *and* columns. So a norm that penalizes less on matrix with sparse rows and columns is favoured. Trace norm $\|\cdot\|_{\text{tr}}$ is our choice according to Table 5.1, because U and V are penalized less compared to W. Then we optimize

$$\min_W \quad \frac{\beta}{2}\|W\|_{\text{tr}}^2 + L(XW, Y).$$

5.1.4 Output Transformation and Matching Loss

As for output transformation, again we introduce the transform function $f(\cdot)$ such that its range is the desired range $\mathcal{Y}$. If $\widehat{\boldsymbol{z}}^\top = \boldsymbol{x}^\top W$ is pre-prediction, then $\widehat{\boldsymbol{y}} = f(\widehat{\boldsymbol{z}})$ is post-prediction.

- For $\boldsymbol{y} \succeq \boldsymbol{0}_k$, $f(\boldsymbol{z}) = e^{\boldsymbol{z}}$;
- For $\boldsymbol{y} \succeq \boldsymbol{0}_k, \boldsymbol{y}^\top \boldsymbol{1}_k = 1$, $f(\boldsymbol{z}) = e^{\boldsymbol{z}} / \left(\boldsymbol{1}_k^\top e^{\boldsymbol{z}}\right)$, which is sometimes called "softmax" transform in machine learning literature;
- For $\boldsymbol{y} = \mathbb{1}_c = [0, 0, \cdots, 1, 0, 0, \cdots, 0]^\top$, which is a class indicator that has 1 in the cth location and 0s everywhere else, $f(\boldsymbol{z}) = \text{indmax}(\boldsymbol{z})$, which puts 1 in the maximum entry and 0s everywhere else.

For matching loss, we first pick a strictly convex function $F : \mathbb{R}^k \mapsto \mathbb{R}$ such that $f(\boldsymbol{z}) = \nabla F(\boldsymbol{z})$. Then define loss function as

$$\begin{aligned} L(\widehat{\boldsymbol{y}}, \boldsymbol{y}) &= L(f(\widehat{\boldsymbol{z}}), f(\boldsymbol{z})) \\ &= F(\widehat{\boldsymbol{z}}) - F(\boldsymbol{z}) - f^\top(\boldsymbol{z})(\widehat{\boldsymbol{z}} - \boldsymbol{z}) \end{aligned}$$

Because of strict convexity, we have

$$F(\widehat{\boldsymbol{z}}) \geq F(\boldsymbol{z}) + f^\top(\boldsymbol{z})(\widehat{\boldsymbol{z}} - \boldsymbol{z}).$$

Hence, $L(\widehat{\boldsymbol{y}}, \boldsymbol{y}) \geq 0$ and

$$L(\widehat{\boldsymbol{y}}, \boldsymbol{y}) = 0 \iff f(\widehat{\boldsymbol{z}}) = f(\boldsymbol{z}).$$

For previous examples:

- $\boldsymbol{y} \succeq \boldsymbol{0}_k$. $f(\boldsymbol{z}) = e^{\boldsymbol{z}}$, $F(\boldsymbol{z}) = \boldsymbol{1}_k^\top e^{\boldsymbol{z}}$, $\boldsymbol{z} = \ln(\boldsymbol{y})$, pair-wisely. So,

$$\begin{aligned} L(\widehat{\boldsymbol{y}}, \boldsymbol{y}) &= F(\widehat{\boldsymbol{z}}) - F(\boldsymbol{z}) - f^\top(\boldsymbol{z})(\widehat{\boldsymbol{z}} - \boldsymbol{z}) \\ &= \boldsymbol{1}_k^\top e^{\widehat{\boldsymbol{z}}} - \boldsymbol{1}_k^\top e^{\boldsymbol{z}} - (e^{\boldsymbol{z}})^\top(\widehat{\boldsymbol{z}} - \boldsymbol{z}) \\ &= \boldsymbol{1}_k^\top \widehat{\boldsymbol{y}} - \boldsymbol{1}_k^\top \boldsymbol{y} - \boldsymbol{y}^\top(\ln \widehat{\boldsymbol{y}} - \ln \boldsymbol{y}) \\ &= \boldsymbol{y}^\top(\ln \boldsymbol{y} - \ln \widehat{\boldsymbol{y}}) + \boldsymbol{1}_k^\top(\widehat{\boldsymbol{y}} - \boldsymbol{y}) \end{aligned}$$

  which is known as un-normalized entropy.

- $\boldsymbol{y} \succeq \mathbf{0}_k, \boldsymbol{y}^\top \mathbf{1}_k = 1$. $f(\boldsymbol{z}) = e^{\boldsymbol{z}} / \left(\mathbf{1}_k^\top e^{\boldsymbol{z}}\right)$, $F(\boldsymbol{z}) = \ln\left(\mathbf{1}_k^\top e^{\boldsymbol{z}}\right)$. So,

$$\begin{aligned}
L(\widehat{\boldsymbol{y}}, \boldsymbol{y}) &= F(\widehat{\boldsymbol{z}}) - F(\boldsymbol{z}) - f^\top(\boldsymbol{z})(\widehat{\boldsymbol{z}} - \boldsymbol{z}) \\
&= \ln\left(\mathbf{1}_k^\top e^{\widehat{\boldsymbol{z}}}\right) - \ln\left(\mathbf{1}_k^\top e^{\boldsymbol{z}}\right) - \left(\frac{e^{\boldsymbol{z}}}{\mathbf{1}_k^\top e^{\boldsymbol{z}}}\right)^\top (\widehat{\boldsymbol{z}} - \boldsymbol{z}) \\
&= \mathbf{1}_k^\top \left(\frac{e^{\boldsymbol{z}}}{\mathbf{1}_k^\top e^{\boldsymbol{z}}}\right) \ln\left(\mathbf{1}_k^\top e^{\widehat{\boldsymbol{z}}}\right) - \mathbf{1}_k^\top \left(\frac{e^{\boldsymbol{z}}}{\mathbf{1}_k^\top e^{\boldsymbol{z}}}\right) \ln\left(\mathbf{1}_k^\top e^{\boldsymbol{z}}\right) - \left(\frac{e^{\boldsymbol{z}}}{\mathbf{1}_k^\top e^{\boldsymbol{z}}}\right)^\top \left(\ln e^{\widehat{\boldsymbol{z}}} - \ln e^{\boldsymbol{z}}\right) \\
&= \left(\frac{e^{\boldsymbol{z}}}{\mathbf{1}_k^\top e^{\boldsymbol{z}}}\right)^\top \left(\mathbf{1}_k \ln\left(\mathbf{1}_k^\top e^{\widehat{\boldsymbol{z}}}\right) - \mathbf{1}_k \ln\left(\mathbf{1}_k^\top e^{\boldsymbol{z}}\right) - \ln e^{\widehat{\boldsymbol{z}}} + \ln e^{\boldsymbol{z}}\right) \\
&= \left(\frac{e^{\boldsymbol{z}}}{\mathbf{1}_k^\top e^{\boldsymbol{z}}}\right)^\top \left(\ln \frac{e^{\boldsymbol{z}}}{\mathbf{1}_k^\top e^{\boldsymbol{z}}} - \ln \frac{e^{\widehat{\boldsymbol{z}}}}{\mathbf{1}_k^\top e^{\widehat{\boldsymbol{z}}}}\right) \\
&= \boldsymbol{y}^\top (\ln \boldsymbol{y} - \ln \widehat{\boldsymbol{y}})
\end{aligned}$$

which is known as *KL-divergence*. KL-divergence

5.1.5 Surrogate Loss for Classification

The last piece of multivariate output is choosing a reasonable surrogate loss for classification transform $f(\boldsymbol{z}) = \text{indmax}(\boldsymbol{z})$. The ideal loss is an indicator loss function

$$L(\widehat{\boldsymbol{y}}, \boldsymbol{y}) = \mathbb{1}(\boldsymbol{y} \neq \widehat{\boldsymbol{y}}),$$

but it leads to intractability. Here we list some surrogate losses for multivariate classification.

- Multinomial deviance (logistic margin loss).

$$L(\widehat{\boldsymbol{z}}, \boldsymbol{y}) = \ln(\mathbf{1}_k^\top e^{\widehat{\boldsymbol{z}}}) - \boldsymbol{y}^\top \widehat{\boldsymbol{z}}. \tag{5.1}$$

- Multiclass SVM loss (analogized to hinge loss in univariate case).

$$L(\widehat{\boldsymbol{z}}, \boldsymbol{y}) = \max(\underbrace{\mathbf{1}_k - \boldsymbol{y}}_{\substack{\text{misclassifi-} \\ \text{cation error}}} + \underbrace{\widehat{\boldsymbol{z}} - \boldsymbol{y}^\top \widehat{\boldsymbol{z}} \mathbf{1}_k}_{\text{margin}}). \tag{5.2}$$

where the max(·) operation is to find the maximum value over k entries of given vector.

Therefore, the multiclass SVM is

$$\begin{aligned}
&\min_{W} \quad \frac{\beta}{2}\|W\|_F^2 + \sum_i \max(\mathbf{1}_k^\top - Y_{i:} + X_{i:}W - X_{i:}WY_{i:}^\top \mathbf{1}_k^\top) \\
\Longleftrightarrow &\min_{W,\boldsymbol{\xi}} \quad \frac{\beta}{2}\|W\|_F^2 + \mathbf{1}_t^\top \boldsymbol{\xi} \\
&\text{s.t.} \quad \boldsymbol{\xi}\mathbf{1}_k^\top \succeq \mathbf{1}_t \mathbf{1}_k^\top - Y + XW - \delta(XWY^\top)\mathbf{1}_k^\top
\end{aligned}$$

where $\delta(A)$ is the column vector of the diagonal of A (opposite of $\Delta(\boldsymbol{a})$) and the $\succeq$ in the constraint means entry-wise grater than or equal to instead of PSD notation. This is a quadratic programming and can be kernelized because of $\|W\|_F^2$. During test time, given $\boldsymbol{x}_0$,

$$\widehat{\boldsymbol{y}}_0^\top = \text{indmax}(\boldsymbol{x}_0^\top W)$$

5.2 Structured Output

In many real world scenarios, the number of classes is really large, say exponential. One illustrative example is optical character recognition (OCR).

Optical character recognition: exponential number of classes

Consider the simple hand-written phrase "*the red apple*", shown as images. We know that, generally speaking, there are 26 possibilities for each letter. A simple phrase like this has $(3 + 3 + 5)^{26}$ possible "classes", one of which is "the rcd qpple". This set is too large to handle without further thinking. One solution is to decouple the phrase into individual letters. Indeed, this approach simplifies the problem, but ignores the strong correlation or context. For example, given "r_d", "e" is much more likely than "c"; "a" is much more likely than "q" given "_pple".

When the number of class, $|\mathcal{C}| = k$, is really large, we have to utilize the structure of the output. We first assume that the output label is decoupled as $\boldsymbol{c}_i = (c_{i1}, c_{i2}, \cdots, c_{il}) \in \mathcal{C}'^l$

instead of $\boldsymbol{c}_i \in \mathcal{C}$, where $\mathcal{C}'$ is the candidate set for individual c_{ij} and l is the length of the structure. In the OCR example or speech recognition, this assumption is reasonable since OCR image is composed of sequential letter images and speech consists of decoupled phonemes. Then we give the model a decoupled representation, i.e.,

$$\widetilde{\boldsymbol{\varphi}}(\boldsymbol{x}_i, \boldsymbol{c}_i) = \sum_{j=1}^{l-1} \boldsymbol{\varphi}(\boldsymbol{x}_i, c_{ij}, c_{i(j+1)}).$$

In the OCR example, $\boldsymbol{\varphi}(\boldsymbol{x}_i, c_{ij}, c_{i(j+1)})$ can be an image (vectorized pixel values) of two consecutive letters. This implies

$$\widehat{\boldsymbol{z}}_i = \widetilde{\boldsymbol{w}}^\top \widetilde{\boldsymbol{\varphi}}(\boldsymbol{x}_i, \boldsymbol{c}_i) = \sum_{j=1}^{l-1} \boldsymbol{w}^\top \boldsymbol{\varphi}(\boldsymbol{x}_i, c_{ij}, c_{i(j+1)}).$$

This decouple approach maintains (at lease partially) the context information, and more importantly, creates tractability as we will see soon. Consider the multivariate surrogate losses introduced in previous section, multinomial deviance (Eq.(5.1)) and multiclass SVM loss (Eq.(5.2)). The main difficulty in computing multinomial deviance is to sum up exponential number of values, while for multiclass SVM, is to find the maximum in exponentially many values.

- Case of multinomial deviance. To compute $\mathbf{1}_k^\top e^{\widehat{\boldsymbol{z}}_i} = \sum e^{\widehat{\boldsymbol{z}}_i}$ is hard when k, the number of class, is really large. However, with the help of decouple formulation, we have

$$\begin{aligned}\sum e^{\widehat{\boldsymbol{z}}_i} &= \sum_{\{c_{i1},\cdots,c_{il}\}} \exp\left(\sum_{j=1}^{l-1} \boldsymbol{w}^\top \boldsymbol{\varphi}(\boldsymbol{x}_i, c_{ij}, c_{i(j+1)})\right) \\ &= \sum_{\{c_{i1},\cdots,c_{il}\}} \prod_{j=1}^{l-1} \exp\left(\boldsymbol{w}^\top \boldsymbol{\varphi}(\boldsymbol{x}_i, c_{ij}, c_{i(j+1)})\right),\end{aligned}$$

  which can be calculated using “sum-product” algorithm.

“Sum-product” algorithm

Consider the following example:

$$p = \sum_{\{c_1,c_2,c_3,c_4,c_5\}} f_4(c_4,c_5) f_3(c_3,c_4) f_2(c_2,c_3) f_1(c_1,c_2)$$

where $\{c_1, c_2, c_3, c_4, c_5\}$ is for purpose of illustration. In fact, there may be many c_i. To efficiently compute this, we push the summation inward,

$$p = \sum_{c_5} \underbrace{\sum_{c_4} f_4(c_4,c_5) \underbrace{\sum_{c_3} f_3(c_3,c_4) \underbrace{\sum_{c_2} f_2(c_2,c_3) \underbrace{\sum_{c_1} f_1(c_1,c_2)}_{m_2(c_2)}}_{m_3(c_3)}}_{m_4(c_4)}}_{m_5(c_5)}$$

By doing so, the computation is reduced from $O(|\mathcal{C}'|^l)$ to $O(l|\mathcal{C}'|^2)$.

- Case of multiclass SVM. To compute max $\widehat{\boldsymbol{z}}_i$, we have

$$\max\ \widehat{\boldsymbol{z}}_i = \max_{\{c_{i1},\cdots,c_{il}\}} \left(\sum_{j=1}^{l-1} \boldsymbol{w}^\top \boldsymbol{\varphi}(\boldsymbol{x}_i, c_{ij}, c_{i(j+1)})\right),$$

  which can be solved efficiently using “max-sum” algorithm.

"Max-sum" algorithm

Consider the following example:

$$m = \max_{\{c_1,c_2,c_3,c_4,c_5\}} f_4(c_4,c_5) + f_3(c_3,c_4) + f_2(c_2,c_3) + f_1(c_1,c_2)$$

where $\{c_1, c_2, c_3, c_4, c_5\}$ is for purpose of illustration. In fact, there may be many c_i. To efficiently compute this, we push the max operation inward,

$$m = \max_{c_5} \underbrace{\max_{c_4} f_4(c_4,c_5) + \underbrace{\max_{c_3} f_3(c_3,c_4) + \underbrace{\max_{c_2} f_2(c_2,c_3) + \underbrace{\max_{c_1} f_1(c_1,c_2)}_{m_2(c_2)}}_{m_3(c_3)}}_{m_4(c_4)}}_{m_5(c_5)}$$

By doing so, the computation is reduced from $O(|\mathcal{C}'|^l)$ to $O(l|\mathcal{C}'|^2)$.

These structure learning techniques have been generalized from sequential structure to tree and graph, i.e., any structure that satisfies the sum-product or max-sum properties.

Chapter 6

Unsupervised Learning

In supervised learning, the label information ($Y \in \mathcal{Y}$) is given in training time. For unsupervised learning, the label is removed, i.e., we know $X \in \mathcal{X}$ but do not know $Y \in \mathcal{Y}$. What if we still want to find a mapping $h : \mathcal{X} \mapsto \mathcal{Y}$ for specific $\mathcal{Y}$? For example, we may want to learn a projection matrix $W \in \mathbb{R}^{n\times k}$ that project $X \in \mathbb{R}^{t\times n}$ to $Y \in \mathbb{R}^{t\times k}$. Here we list some well studied unsupervised learning scenarios:

- Dimensionality reduction: $Y \in \mathbb{R}^{t\times k}$, where $0 < k < n$.
- Hard clustering (unsupervised classification): $Y \in \{0, 1\}^{t\times k}$, such that $Y\mathbf{1}_k = \mathbf{1}_t$.
- Soft clustering: $Y \in \mathbb{R}^{t\times k}$, such that $Y\mathbf{1}_k = \mathbf{1}_t$.

6.1 Dimensionality Reduction

Before we precede, let us first investigate a reverse formulation of regular supervised least square problem. In regular least square formulation, given X and Y,

$$\min_{W\in\mathbb{R}^{n\times k}} \quad \frac{1}{2}\|XW - Y\|_F^2.$$

It is not difficult to solve this and we can show that $W^* = (X^\top X)^{-1}X^\top Y$ if $X^\top X$ has full rank. Define $X^+ = (X^\top X)^{-1}X^\top$, which is called left *Moore-Penrose pseudoinverse*, then $W^* = X^+Y$. Let the singular value decomposition (SVD) of X be $X = Q_X\Sigma_X V_X^\top$ where Σ_X is diagonal matrix with singular values and Q_X and V_X are unitary such that $Q_X^\top Q_X = I_{\text{rank}(X)}, V_X^\top V_X = I_{\text{rank}(X)}$, then we have $X^+ = V_X\Sigma_X^{-1}Q_X^\top$. Now consider the reverse formulation of the least square problem, given X and Y,

Moore-Penrose pseudoinverse

$$\min_{U\in\mathbb{R}^{k\times n}} \quad \frac{1}{2}\|X - YU\|_F^2.$$

The optimal reverse projection is given by $U^* = Y^+X$. Notice that on optimal W and U for the above problems, we have

$$\begin{aligned} X^\top XW &= X^\top Y = U^\top Y^\top Y \\ \Longrightarrow W &= (X^\top X)^{-1}U^\top(Y^\top Y), \\ U &= (Y^\top Y)^{-1}W^\top(X^\top X) \end{aligned}$$

Therefore, given X and Y, we can always compute one of W and U from the other.

Now back to unsupervised learning where Y is unknown. The main idea to solve unsupervised learning is to guess a Y, while optimizing W. Consider

$$\min_{Y,W} \quad \frac{1}{2}\|XW - Y\|_F^2.$$

This formulation is meaningless without constraints since we can always find a trivial solution, for example, $W = 0, Y = 0$, that obtains zero loss. However, when it comes to its reverse formulation, we have

$$\min_{Y,U} \quad \frac{1}{2}\|X - YU\|_F^2. \tag{6.1}$$

Fix Y, we know that $U^* = Y^+X$. Plug in the reverse formulation, we have

$$\begin{aligned} &\min_Y \quad \frac{1}{2}\|X - YY^+X\|_F^2 \\ \Longrightarrow &\min_Y \quad \frac{1}{2}\mathrm{tr}\left[(I_t - YY^+)XX^\top(I_t - YY^+)^\top\right] \\ \Longrightarrow &\min_Y \quad \frac{1}{2}\mathrm{tr}\left[(I_t - YY^+)K\right] \\ \Longrightarrow &\max_Y \quad \mathrm{tr}(YY^+K), \end{aligned} \tag{6.2}$$

where $K = XX^\top$ is the kernel matrix. The second derivation above is based on the fact that YY^+ is symmetric and $\mathrm{tr}(YY^+XX^\top YY^+) = \mathrm{tr}(YY^+YY^+XX^\top) = \mathrm{tr}(YY^+XX^\top)$. Let the SVD of Y be $Y = Q_Y\Sigma_Y V_Y^\top$, then $Y^+ = V_Y\Sigma_Y^{-1}Q_Y^\top$, $YY^+ = Q_YQ_Y^\top$. With such notation, the reverse formulation becomes

$$\begin{aligned} &\max_{Q_Y} \quad \mathrm{tr}(Q_YQ_Y^\top K), \quad \text{s.t.} \quad Q_Y^\top Q_Y = I_{\mathrm{rank}(Y)} \\ \Longrightarrow &\max_{Q_Y} \quad \mathrm{tr}(Q_Y^\top KQ_Y), \quad \text{s.t.} \quad Q_Y^\top Q_Y = I_{\mathrm{rank}(Y)} \\ \Longrightarrow &\max_{\boldsymbol{q}_i} \quad \sum_{i=1}^{\mathrm{rank}(Y)} \boldsymbol{q}_i^\top K\boldsymbol{q}_i, \end{aligned}$$

where $\boldsymbol{q}_i$ are mutually orthogonal with unit length. Assume $\mathrm{rank}(Y) = k$, it is intuitive to see that by setting the $\boldsymbol{q}_i$ to be the top k eigenvectors of K, the objective value is maximized. By top k, we mean the k eigenvectors that correspond to k maximal eigenvalues. We denote the matrix of top k eigenvectors of K as $Q_{X;1:k}$. If $K = XX^\top$, with $X = Q_X\Sigma_XV_X^\top$, then it is equivalent to setting $\boldsymbol{q}_i$ to be the top k left singular vectors of X.

To conclude, optimal solutions are given by $Y^* = Q_{X;1:k}, U^* = Q_{X;1:k}^\top X$. The original data matrix X is approximated by $X \approx Q_{X;1:k}Q_{X;1:k}^\top X$. This technique is known as *principal component analysis* (PCA).

One may notice that to deal with new data points, we still need the regular projection matrix W. Thanks to the relation between W and U, we can reconstruct W by setting

$$W = (X^\top X)^{-1}U^\top(Y^\top Y) = X^+Q_{X,1:k} = V_{X;1:k}\Sigma_{X;1:k}^{-1}.$$

To this point, we have done dimensionality reduction.

PCA and data centering

PCA is also derived from a different perspective in statistics, by preserving maximum variance in the new feature space. In order to capture covariance information, the dataset must be *centered* before eigenvalue decomposition:

$$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i - \frac{1}{t}\sum_{j=1}^{t}\boldsymbol{x}_j. \tag{6.3}$$

Data centering is beneficial in general since we mainly want to preserve the relative information between data points instead of the shift of the whole set in the $\mathcal{X}$ space.

6.2 Hard Clustering

Next we will talk about clustering problem, where $Y \in \{0,1\}^{t\times k}$ and $Y\mathbf{1}_k = \mathbf{1}_t$. The problem is to find W and Y simultaneously. The clustering assignment rule is $\widehat{y}^\top = \mathrm{indmax}(\boldsymbol{x}^\top W)$. The regular supervised problem with square surrogate loss is

$$\min_{W, Y\in\mathcal{Y}} \quad \frac{1}{2}\|XW - Y\|_F^2,$$

where $\mathcal{Y} = \{Y \in \{0,1\}^{t\times k}, Y\mathbf{1}_k = \mathbf{1}_t\}$. Its reverse formulation is

$$\min_{U, Y\in\mathcal{Y}} \quad \frac{1}{2}\|X - YU\|_F^2.$$

- Given fixed Y, we already know that the optimal U is given by $U^* = Y^+X = (Y^\top Y)^{-1}Y^\top X$. Note that $Y^\top Y = \Delta([t_{c_1}, t_{c_2}, \cdots, t_{c_k}]^\top)$, where t_{c_j} is the number of points in the jth cluster c_j, i.e., $Y^\top Y$ is a diagonal matrix with numbers of points of each cluster on the diagonal. Also note that $Y^\top X = [\sum_{\boldsymbol{x}\in c_1} \boldsymbol{x}, \sum_{\boldsymbol{x}\in c_2} \boldsymbol{x}, \cdots, \sum_{\boldsymbol{x}\in c_k} \boldsymbol{x}]^\top$. To combine, $(Y^\top Y)^{-1}Y^\top X = Y^+X = [\boldsymbol{m}_1, \boldsymbol{m}_2, \cdots, \boldsymbol{m}_k]^\top$, where $\boldsymbol{m}_j$ is the mean of points in cluster c_j, i.e., $U^* = Y^+X$ is a matrix with rows of cluster means.

- If U is fixed, how to solve Y? Notice that

$$\|X - YU\|_F^2 = \sum_i \|X_{i:} - Y_{i:}U\|_2^2.$$

Due to the fact that $Y_{i:} = [0, 0, \cdots, 1, \cdots, 0]$, it only has a single one indicating that the ith point belongs to the jth cluster. Obviously, the best strategy is to assign the ith data point to the jth cluster such that the distance between $X_{i:}$ and cluster mean $Y_{i:}U = U_{j:}$ is minimized.
In another point of view, define the squared (Euclidean) distance matrix $D \in \mathbb{R}^{t\times k}$ that represents the distances between points data X and cluster center U such that (compared to one-dimensional counterpart $(x-u)^2 = x^2 + u^2 - 2xu$)

$$D(X, U) = \delta(XX^\top)\mathbf{1}_k^\top + \mathbf{1}_t\delta^\top(UU^\top) - 2XU^\top.$$

Then we have $Y_{i:}^*$ equals to $[0, 0, \cdots, 1, \cdots, 0]$, where the single one is in the jth entry and $j = \operatorname{argmin}_j \; D_{ij}$.

Therefore, to solve the problem (reverse clustering formulation with square surrogate loss), we have a two-step optimization procedure:

- Fix Y, solve U to be the matrix of cluster means.
- Fix U, solve Y to be the indicator matrix such that each point is assigned to the closest cluster (based on the distance to cluster mean).

This procedure is the well-known *k-means clustering* algorithm. (And we know its drawback immediately because of square loss: not robust, chasing outliers)

As a side note, solution to problem 6.1 is globally optimal, because after plugging in $U^* = Y^+X$, we can still effectively find a globally optimal Y to problem 6.2. However, in the case of hard clustering, we cannot effectively find the optimal Y for

$$\min_{Y\in\mathcal{Y}} \quad \frac{1}{2}\|X - Y(Y^\top Y)^{-1}Y^\top X\|_F^2.$$

after plugging in $U^* = Y^+X = (Y^\top Y)^{-1}Y^\top X$. So k-means is a local optimization procedure, minimizing U and Y alternatively. Generally, we cannot solve the clustering problem effectively, not even for square loss, due to non-continuous output space $\mathcal{Y}$.

6.3 Kernelization

6.3.1 Kernel Principal Component Analysis

Recall the supervised formulation that leads to kernelization

$$\begin{aligned} &\min_W \quad \frac{\beta}{2}\|W\|_F^2 + \frac{1}{2}\|XW - Y\|_F^2 \\ \iff &\min_A \quad \frac{\beta}{2}\operatorname{tr}(A^\top KA) + \frac{1}{2}\|KA - Y\|_F^2 \end{aligned} \tag{6.4}$$

which is based on $W^* = X^\top A^*$ and the representor Theorem. Its derivative

$$\begin{aligned} \frac{d\text{Obj}}{dA} &= \beta KA + K(KA - Y) = 0 \\ &\iff K(K + \beta I_t)A = KY \\ &\impliedby (K + \beta I_t)A = Y \\ &\qquad \iff A = (K + \beta I_t)^{-1}Y. \end{aligned} \tag{6.5}$$

The reverse formulation is

$$\min_U \quad \frac{1}{2}\|X - YU\|_F^2.$$

It is easy to see that $U^* = Y^+X = BX$ for some particular $B \in \mathbb{R}^{k\times t}$. Therefore,

$$\begin{aligned}
&\min_U \quad \frac{1}{2}\|X - YU\|_F^2 \\
\iff &\min_B \quad \frac{1}{2}\|X - YBX\|_F^2 \\
\iff &\min_B \quad \frac{1}{2}\mathrm{tr}\left((I - YB)K(I - YB)^\top\right)
\end{aligned}$$

We take the derivative

$$\begin{aligned}
\frac{d\mathrm{Obj}}{dB} &= \beta Y^\top(YB - I_t)K = 0 \\
&\iff Y^\top YBK = Y^\top K \\
&\Longleftarrow Y^\top YB = Y^\top \\
&\iff B = (Y^\top Y)^{-1}Y^\top = Y^+.
\end{aligned} \tag{6.6}$$

Here we may assume that $Y^\top Y$ is invertible ($\mathrm{rank}(Y) = k, k < t$). From Eq. (6.5) and Eq. (6.6), we know that

$$\begin{aligned}
(K + \beta I_t)A &= Y = B^\top Y^\top Y \\
A &= (K + \beta I_t)^{-1}B^\top Y^\top Y.
\end{aligned} \tag{6.7}$$

Thus, we also have a relation between A and B given X and Y.

Now come back to the unsupervised case, where K is given but Y is not. As before, minimizing Y and A simultaneously for Eq.(6.4) is meaningless because we can find trivial solution that lead to zero loss. However, the reverse form is readily usable, i.e.,

$$\min_{B,Y} \quad \frac{1}{2}\mathrm{tr}\left((I_t - YB)K(I_t - YB)^\top\right).$$

With $B^* = Y^+$, we have

$$\min_Y \quad \frac{1}{2}\mathrm{tr}\left((I_t - YY^+)K(I_t - YY^+)^\top\right).$$

The rest of our derivative is identical to problem 6.2, and the optimal Y is given by $Y = Q_{X;1:k}$, the top k eigenvectors of K. Here K is the kernel matrix and not necessarily equals $XX^\top$. This method is referred to as *kernel principal component analysis* (KPCA). By comparison, PCA always takes the top k left singular vectors of X. If $K = XX^\top$, then KPCA is identical to PCA.

6.3.2 Kernelized Clustering

In clustering setting, the output space is constrained as $\mathcal{Y} = \{Y \in \{0,1\}^{t\times k}, Y\mathbf{1}_k = \mathbf{1}_t\}$. With square surrogate loss, we have

$$\min_{B,Y\in\mathcal{Y}} \quad \frac{1}{2}\mathrm{tr}\left((I_t - YB)K(I_t - YB)^\top\right).$$

Unsurprisingly, this problem is NP-hard due the the non-continuous output space, but we can always alternate between Y and B and get a local solution. We already know that the optimal B given Y is $B = Y^+$. With fixed B, how to compute Y?

Let us, for a moment, convert the kernel to original X representation:

$$(I_t - YB)K(I_t - YB)^\top = (X - YBX)(X - YBX)^\top.$$

Let $U = BX \in \mathbb{R}^{k\times n}$. Then consider the matrix $D \in \mathbb{R}^{t\times k}$ such that

$$\begin{aligned}
D(X, U) &= \delta(XX^\top)\mathbf{1}_k^\top + \mathbf{1}_t^\top\delta^\top(UU^\top) - 2XU^\top \\
&= \delta(K)\mathbf{1}_k^\top + \mathbf{1}_t^\top\delta^\top(BKB^\top) - 2KB^\top.
\end{aligned}$$

This is analogized to the Euclidean distance matrix of each point $X_{i:}$ to the cluster center $U_{j:}$. Now given B, we assign $Y_{i:}$ to be $[0, 0, \cdots, 1, \cdots, 0]$, where the single one is in the jth entry and $j = \operatorname*{argmin}_j \; D_{ij}$. This is called *kernel k-means clustering.*

One benefit of regular and reverse formulations is that, we can recover the regular solution A with the relation between A and B. With Eq. 6.7, we can extend our model to newly observed data, even for clustering problem. That is, given $\boldsymbol{x}_0$, its cluster index is computed as

$$\begin{aligned}\widehat{\boldsymbol{y}}_0^\top &= \operatorname{indmax}(\boldsymbol{x}_0^\top W)\\ &= \operatorname{indmax}(\boldsymbol{x}_0^\top X^\top A)\\ &= \operatorname{indmax}(\boldsymbol{k}_0^\top A),\end{aligned}$$

where $\boldsymbol{k}_0^\top = \boldsymbol{\varphi}(\boldsymbol{x}_0)^\top \Phi^\top$ is the inner product of (feature-extended) new point $\boldsymbol{x}_0$ and (feature-extended) training set Φ.

6.3.3 Normalized Kernel Methods

Here we introduce the optimization scheme that utilizes normalized kernels. Assume the kernel matrix K is doubly non-negative, i.e., $K_{ij} \geq 0, \forall i, j$ and $K \succeq 0$. Let

$$\begin{aligned}\Lambda &= \Delta(K\mathbf{1}_t)\\ \widetilde{K} &= \Lambda^{-\frac{1}{2}} K \Lambda^{-\frac{1}{2}}\\ \widetilde{A} &= \Lambda^{-\frac{1}{2}} A\\ \widetilde{Y} &= \Lambda^{\frac{1}{2}} Y\\ \widetilde{B} &= B\Lambda^{\frac{1}{2}}.\end{aligned}$$

Supervised learning. Compared to (6.4), consider a different (supervised) formulation:

$$\min_A \quad \frac{\beta}{2}\operatorname{tr}(A^\top \underbrace{\Lambda^{-1}K\Lambda^{-1}}_{\text{rescaled kernel}} A) + \frac{1}{2}\operatorname{tr}(\underbrace{\Lambda}_{\text{instance weight}}(\Lambda^{-1}K\Lambda^{-1}A - Y)(\underbrace{\Lambda^{-1}K\Lambda^{-1}A}_{\text{prediction of Y from rescaled kernel}} - Y)^\top). \tag{6.8}$$

The motivation of this optimization is to minimize square training loss for *rescaled* kernel $\Lambda^{-1}K\Lambda^{-1}$ where each entry is normalized by the importance weight $\Lambda = \Delta(K\mathbf{1}_t)$. Moreover, the square loss is also rescaled according to the instance importance Λ. It is equivalent to

$$\min_{\widetilde{A}} \quad \frac{\beta}{2}\operatorname{tr}(\widetilde{A}^\top \widetilde{K}\widetilde{A}) + \frac{1}{2}\operatorname{tr}\left((\widetilde{K}\widetilde{A} - \widetilde{Y})(\widetilde{K}\widetilde{A} - \widetilde{Y})^\top\right). \tag{6.9}$$

After taking derivative w.r.t. $\widetilde{A}$ and setting it to zero, we have

$$A^* = \Lambda^{\frac{1}{2}}(\widetilde{K} + \beta I_t)^{-1}\Lambda^{\frac{1}{2}}Y. \tag{6.10}$$

Recall from (6.8) that we predict the output Y by

$$Y \approx \Lambda^{-1}K\Lambda^{-1}A^* = \Lambda^{-1}XX^\top\Lambda^{-1}A^* = \Lambda^{-1}XW^*$$

This can be seen as to rescale input X then predict via projection W^*. For a new data point $\boldsymbol{x}$, define $\lambda = \boldsymbol{x}^\top X^\top \mathbf{1}_t = \boldsymbol{k}^\top \mathbf{1}_t$ as normalization constant. To predict its label,

$$\widehat{y} = \lambda^{-1}\boldsymbol{x}^\top W^* = \frac{\boldsymbol{k}^\top \Lambda^{-1} A^*}{\boldsymbol{k}^\top \mathbf{1}_t}.$$

Unsupervised regression (dimensionality reduction): normalized PCA. The reverse formulation is

$$\begin{aligned}&\min_{\widetilde{B}, \widetilde{Y}} \quad \frac{1}{2}\operatorname{tr}\left((I_t - \widetilde{Y}\widetilde{B})\widetilde{K}(I_t - \widetilde{Y}\widetilde{B})^\top\right)\\ \Longrightarrow &\min_{\widetilde{Y}} \quad \frac{1}{2}\operatorname{tr}\left((I_t - \widetilde{Y}\widetilde{Y}^+)\widetilde{K}(I_t - \widetilde{Y}\widetilde{Y}^+)^\top\right),\end{aligned} \tag{6.11}$$

with

$$B^* = \widetilde{Y}^+ = (Y^\top \Lambda Y)^{-1} Y^\top. \tag{6.12}$$

Similar to KPCA, the optimal $\widetilde{Y}^*$ is the top k eigenvectors of $\widetilde{K}$. Combining (6.12) and (6.10), we have

$$\begin{aligned}\Lambda^{-\frac{1}{2}}(\widetilde{K}+\beta I_t)\Lambda^{-\frac{1}{2}}A = Y = B^\top(Y^\top \Lambda Y)\\ A = \Lambda^{\frac{1}{2}}(\widetilde{K}+\beta I_t)^{-1}\Lambda^{\frac{1}{2}}B^\top(Y^\top\Lambda Y).\end{aligned} \tag{6.13}$$

Again, this relation allows us to predict new data as we did in previous methods.

Side notes about normalized PCA

- **Top eigenvector**. In terms of the original (un-normalized) Y, we can show that $Y_{:1} = \mathbf{1}_t$. That is, the first component of the new representation $\boldsymbol{y}_i$ is always 1 for very point i. To see this, let $\boldsymbol{\lambda}^{1/2} = \Lambda^{1/2}\mathbf{1}_t$, then

$$\begin{aligned}\widetilde{K}\boldsymbol{\lambda}^{1/2} &= \Lambda^{-1/2}K\Lambda^{-1/2}\Lambda^{1/2}\mathbf{1}_t\\ &= \Lambda^{-1/2}K\mathbf{1}_t\\ &= \Lambda^{-1/2}\Lambda\mathbf{1}_t \qquad \text{since } \Lambda = \Delta(K\mathbf{1}_t) \Rightarrow \Lambda\mathbf{1}_t = K\mathbf{1}_t\\ &= \Lambda^{1/2}\mathbf{1}_t = \boldsymbol{\lambda}^{1/2}\end{aligned}$$

  This means $\boldsymbol{\lambda}^{1/2}$ is always the eigenvector of $\widetilde{K}$ with eigenvalue of 1. It is known that the maximum eigenvalue of $\widetilde{K}$ is 1. Therefore, $\boldsymbol{\lambda}^{1/2}$ is the top eigenvector of $\widetilde{K}$ and

$$Y_{:1} = \Lambda_{-1/2}\widetilde{Y}_{:1} = \Lambda_{-1/2}\boldsymbol{\lambda}^{1/2} = \Lambda_{-1/2}\boldsymbol{\lambda}^{1/2} = \mathbf{1}_t.$$

  So we usually ignore the first component of Y.

- **Kernel centering**. We already know how to center dataset when we have explicit feature representation (Eq.(6.3)). However, when we are given the kernel matrix K, how to center the dataset in its implicit feature space? Notice that the centering operation Eq.(6.3) is equivalent in matrix form to

$$X \leftarrow \left(I_t - \frac{1}{t}\mathbf{1}_t\mathbf{1}_t^\top\right)X.$$

  $H = I_t - \frac{1}{t}\mathbf{1}_t\mathbf{1}_t^\top$ is known as centering matrix. In the linear kernel case, $K = XX^\top$ so we can center it by $K \leftarrow HKH$. In fact, even in non-linear cases, $K = \Phi\Phi^\top$ for some (possibly infinite dimensional) representation Φ and $K \leftarrow HKH$ is still valid for such implicit centering.

Unsupervised classification (clustering): normalized k-means, normalized graph cut. We can expand (6.11),

$$\begin{aligned}&\min_{\widetilde{B},\widetilde{Y}\in\widetilde{\mathcal{Y}}} \quad \frac{1}{2}\mathrm{tr}\left((I_t - \widetilde{Y}\widetilde{B})\widetilde{K}(I_t-\widetilde{Y}\widetilde{B})^\top\right)\\ \Longleftrightarrow \; &\min_{B,Y\in\mathcal{Y}} \quad \frac{1}{2}\mathrm{tr}\left(\Lambda(\Lambda^{-1}-YB)K(\Lambda^{-1}-YB)^\top\right)\end{aligned}$$

where $\mathcal{Y} = \{Y \in \{0,1\}^{t\times k}, Y\mathbf{1}_k = \mathbf{1}_t\}$ in clustering setting. This problem is NP-hard. To recover local solution, we can alternate between B and Y as we did in Section 6.2.

- Given Y, $B^* = (Y^\top\Lambda Y)^{-1}Y^\top$.
- Given B, compute the matrix of all rescaled squared distances

$$\begin{aligned}D(\Lambda^{-1}X, BX) &= \delta(\Lambda^{-1}XX^\top\Lambda^{-1})\mathbf{1}_k^\top + \mathbf{1}_t^\top\delta(BXX^\top B^\top)^\top - 2\Lambda^{-1}XX^\top B^\top\\ &= \delta(\Lambda^{-1}K\Lambda^{-1})\mathbf{1}_k^\top + \mathbf{1}_t^\top\delta(BKB^\top)^\top - 2\Lambda^{-1}KB^\top.\end{aligned}$$

  Then $Y_{i:}^*$ equals to $[0,0,\cdots,1,\cdots,0]$, where the single one is in the jth entry and $j = \operatorname*{argmin}_j \quad D_{ij}$.

This method is known as *normalized graph cut clustering*. Standard technique (6.13) can be applied to assign new data point to clusters.

Normalized cut (N-cut) from a different perspective

Normalized cut was also introduced in the context of graph cut. Suppose we have an "affinity" matrix K, where K_{ij} represents the similarity of object i and object j (for example, in image segmentation, this can be seen as the similarity between two pixels). Suppose also that $K \succeq 0$ (in many cases, this can be satisfied when we choose a proper similarity measure). Consider objects as nodes, while their similarities as edges, we may want to cut this graph into k disjoint components. One intuition is to partition the objects to minimize the normalized cost:

$$\min_{Y\in\mathcal{Y}} \quad \sum_j \frac{\operatorname{cut}(\text{class}_j, \overline{\text{class}_j})}{\operatorname{vol}(\text{class}_j)},$$

where

- $\mathcal{Y} = \{Y \in \{0,1\}^{t\times k}, Y\mathbf{1}_k = \mathbf{1}_t\}$;
- $\operatorname{cut}(\text{class}_j, \overline{\text{class}_j})$ is the cost of cutting class j from the rest of the graph:

  $$\operatorname{cut}(\text{class}_j, \overline{\text{class}_j}) = \frac{1}{2}\sum_{i,l:Y_{ij}\neq Y_{lj}} K_{il},$$

  which is the sum of edges being cut;
- $\operatorname{vol}(\text{class}_j)$ is considered to be the volume of class j:

  $$\operatorname{vol}(\text{class}_j) = \sum_{l,i:Y_{ij}=1} K_{il}$$

  which measures the association of class j and the rest of the graph.

Note that

$$\begin{aligned}
\operatorname{cut}(\text{class}_j, \overline{\text{class}_j}) &= \frac{1}{2}\sum_{i,l:Y_{ij}\neq Y_{lj}} K_{il} = \frac{1}{2}\sum_{i,l} K_{il}(Y_{ij}-Y_{lj})^2 \\
&= \frac{1}{2}\sum_{i,l} K_{il}(Y_{ij}^2 + Y_{lj}^2 - 2Y_{ij}Y_{lj}) \\
&= Y_{:j}^\top(\Delta(K\mathbf{1}_t) - K)Y_{:j} = Y_{:j}^\top(\Lambda - K)Y_{:j}, \\
\operatorname{vol}(\text{class}_j) &= \sum_{l,i:Y_{ij}=1} K_{il} = \sum_{i,l} K_{il}Y_{ij}^2 = \sum_i\left(\sum_l K_{il}\right)Y_{ij}^2 \\
&= Y_{:j}^\top\Delta(K\mathbf{1}_t)Y_{:j} = Y_{:j}^\top\Lambda Y_{:j}.
\end{aligned}$$

Therefore, the problem becomes

$$\begin{aligned}
&\min_{Y\in\mathcal{Y}} && \operatorname{tr}\left((Y^\top\Lambda Y)^{-1}Y^\top(\Lambda-K)Y\right) \\
\Longleftrightarrow\ &\min_{Y\in\mathcal{Y}} && \operatorname{tr}\left(I - Y(Y^\top\Lambda Y)^{-1}Y^\top K\right) \\
\Longleftrightarrow\ &\min_{Y\in\mathcal{Y}} && \operatorname{tr}\left(\Lambda^{-1}K - Y(Y^\top\Lambda Y)^{-1}Y^\top K\right) \\
\Longleftrightarrow\ &\min_{Y\in\mathcal{Y}} && \operatorname{tr}\left(\Lambda(\Lambda^{-1} - Y(Y^\top\Lambda Y)^{-1}Y^\top)K(\Lambda^{-1} - Y(Y^\top\Lambda Y)^{-1}Y^\top)\right) \\
\Longleftrightarrow\ &\min_{Y\in\mathcal{Y}}\min_B && \operatorname{tr}\left(\Lambda(\Lambda^{-1} - YB)K(\Lambda^{-1} - YB)\right),
\end{aligned}$$

which is equivalent to what we have seen before.