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., $\mathbf{a}$) as vector$^1$, normal letter (e.g., $a$) as scalar.

• $\mathbf{a} \succeq \mathbf{b}$ means each component of $\mathbf{a}$ is greater than or equal to the corresponding component of $\mathbf{b}$. Specifically, $\mathbf{a} \succeq \mathbf{0}$ means each component of $\mathbf{a}$ is non-negative.

• $\mathbf{a}^\top$ means the transpose of $\mathbf{a}$.

• $A_{ij}$ means the entry of matrix $A$ that is in the $i$th row and $j$th column. $A_i$ means the $i$th row of $A$ (as a row vector) and $A_j$ means the $j$th 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: L1 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:

\[(x_1, y_1) \rightarrow \text{Learner} \rightarrow h : \mathcal{X} \mapsto \mathcal{Y}\]

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 \(\text{err}(\hat{y}, y)\) to measure the difference between the true output \(y\) and our prediction \(\hat{y} = h(x)\) 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

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

1.2 Data Representation and Error Function

As an example that we will further investigate in the following chapters, assume that \(x_i \in \mathbb{R}^n\) 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

\[
\mathcal{H} = \{h_w | w \in \mathbb{R}^n, h_w(x) = w^\top x\}.
\]

Note that for this linear model, the offset (bias) is omitted. Instead of \(h_{w,b}(x) = w^\top x + b\), we simply appended the input \(x_i\) by appending a 1 on each data point: \([x_i; 1]\). The offset is then encoded in the last element of \(w\). 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 \(y \in \mathbb{R}^t\).

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

- \(L_1\) error: \(\text{err}(\hat{y}, y) = \|\hat{y} - y\|_1\)
- \(L_2\) error: \(\text{err}(\hat{y}, y) = \|\hat{y} - y\|_2^2\)
- \(L_p\) error: \(\text{err}(\hat{y}, y) = \|\hat{y} - y\|_p^p\) \(0 \leq p \leq \infty\)

(Recall that the \(L_1\) norm of a vector \(a\) is the sum of its absolute values \(|a_j|\); general information about \(L_p\) norm can be found online1.)

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

\[
\text{cross entropy} \quad \text{err}(\hat{y}, y) = y \ln \left(\frac{y}{\hat{y}}\right) + (1 - y) \ln \left(\frac{1 - y}{1 - \hat{y}}\right)
\]

1http://en.wikipedia.org/wiki/Lp_space
1.3 Solving Optimization

Now that we see the learning problem as an optimization, when is the training efficient? If the error function $\text{err}(\hat{y}, y)$ is convex in $\hat{y}$, we can reliably identify the global minimizer $h^*$. Simply setting the derivative to zero will lead to the global optimal solution in most cases. If the error function is non-convex in $\hat{y}$, 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

$$\min_w \sum_{i=1}^t |w^\top x_i - y_i|$$

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

$$\min_{w, \delta} \delta^\top 1$$

s.t. $\delta \succeq 0$

$$Xw \preceq y + \delta$$

$$Xw \succeq y - \delta.$$  

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

$L_\infty$ error minimization

$$\min_w \max_i |w^\top x_i - y_i|$$

Recall that the $L_\infty$-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

$$\min_{w, \delta} \delta$$

s.t. $\delta \geq 0$

$$w^\top x_i \leq y_i + \delta \quad i = 1, \ldots, t$$

$$w^\top x_i \geq y_i - \delta \quad i = 1, \ldots, t$$

Again, this is a linear programming.

1.4 Geometry Interpretation

The geometry interpretation of the linear model $\hat{y} = Xw$ is shown in Figure 1.2. Our prediction $\hat{y}$ lies in the column span of matrix $X$. Using the notation $X_{i \cdot}$ to represent the $i$th column of $X$, $\hat{y}$ is in fact a linear combination of $X_{i \cdot}$, with $w_i$ as coefficients. The true output $y$ can be factorized as $y = \hat{y} + y_\perp$. To minimize the $L_2$ squared error is equivalent to minimize the “length” of $y_\perp$. Therefore, the best bet would be to choose the $\hat{y}$ such that $y_\perp$ 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 $\hat{y}$ with different $p$ values. If $p \geq 1$, the error function is convex in $\hat{y}$; 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 predictions 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.
1.5. CONVEXITY/ROBUSTNESS TRADEOFF

![Figure 1.2: Learning procedure.](image)

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 $\mu$ and variance $\sigma^2$. We fit three linear model $\hat{y} = w \cdot x$ with $L_1, L_2$ and $L_\infty$ error functions. Obviously, the $L_\infty$ line chases the outlier significantly, followed by the $L_2$ line, then the $L_1$ line, which basically ignores the outlier.

![Figure 1.3: Error function with different $p$ values.](image)

![Figure 1.4: Learning with outlier, different error functions.](image)

Let’s discuss the phenomenon in Figure 1.4. Large $p$ value in $L_p$ indicates that the error grows rapidly as the prediction $\hat{y}$ 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 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.

<table>
<thead>
<tr>
<th>Error type</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convex error</td>
<td>Efficient optimization</td>
<td>Sensitive to noise</td>
</tr>
<tr>
<td>Robust error</td>
<td>Slow growth, robust</td>
<td>Difficult to optimize</td>
</tr>
</tbody>
</table>

### Robust error functions

Example 1:
\[
\text{err}_\sigma(\hat{y}, y) = \frac{(\hat{y} - y)^2}{\sigma^2 + (\hat{y} - y)^2}, \quad \sigma > 0
\]

Example 2:
\[
\text{err}(\hat{y}, y) = \min \left(1, \frac{(\hat{y} - y)^2}{\sigma^2}\right)
\]

Figure 1.5 illustrates how they behave as \( \hat{y} \) getting further away from \( y \).

![Robust error functions](image)

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

1.6 Beyond Linear Model

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

\[
\mathbf{w} = \arg\min_{\mathbf{w}} \| \Phi \mathbf{w} - \mathbf{y} \| \tag{1.1}
\]

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

**Polynomial curve fitting**

Given \( \{(x_1, y_1), (x_2, y_2), \ldots, (x_t, y_t)\} \), 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 \( \mathbf{x} \in \mathbb{R}^n \); one hidden layer, whose node represents an expanded (hidden) feature \( \varphi_j(\mathbf{x}) \) for the input; and one output layer which is a linear combination of hidden features \( \hat{y} = \sum_{j=1}^l w_j \varphi_j(\mathbf{x}) \). Typically, a neural network learning procedure constructs the weight \( w_j \) and hidden feature \( \varphi_j(\mathbf{x}) \) 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)**

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

1. If we use this similarity function as feature function and choose some fixed instances \( \mathbf{\mu}_1, \cdots, \mathbf{\mu}_l \in \mathcal{X} \), the new representation would be

\[
\mathbf{x} \mapsto \varphi(\mathbf{x}) = [\kappa(\mathbf{x}, \mathbf{\mu}_1), \cdots, \kappa(\mathbf{x}, \mathbf{\mu}_l)]^T.
\]

2. If we place the centres \( \mathbf{\mu}_j \) on the inputs \( \mathbf{x}_i \) themselves, then the new data matrix \( \Phi \) would be: \( \Phi_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j) \). Note that this \( \Phi \) is symmetric since \( \kappa(\mathbf{x}_i, \mathbf{x}_j) = \kappa(\mathbf{x}_j, \mathbf{x}_i) \). Moreover, if \( \Phi \) is invertible, the solution to the learning problem (1.1) with \( L_2 \) error will be

\[
\mathbf{w}^* = \arg\min_{\mathbf{w}} \|\Phi \mathbf{w} - \mathbf{y}\|_2^2 = \Phi^{-1} \mathbf{y}.
\]

**k-nearest neighbor similarity**

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

where \( N_k(\mathbf{x}_j) \) is the set of points that are the top (closest) \( k \) nearest neighbors of \( \mathbf{x}_j \). 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 \( w \). Concretely, we add a regularizer \( R(w) \) to our learning problem

\[
\min_w \|\Phi w - y\| + \beta R(w)
\]

where \( \beta \geq 0 \) is a parameter to control the degree of regularization. \( R(w) = \|w\|_1 \) and \( R(w) = \frac{1}{2} \|w\|_2^2 \) 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 \( R(w) = \|w\|_1 \) regularization.

\[L_2\text{ loss with } L_2 \text{ regularization.}\]

\[
\min_w \|Xw - y\|_2^2 + \beta \|w\|_2^2
\]

We can solve the above optimization by setting derivative w.r.t. \( w \) to zero:

\[
X^\top (Xw - y) + \beta w = 0
\]

\[
\implies (X^\top X + \beta I_t)w = X^\top y
\]

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

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

\(^a\)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 \( \Lambda \) is diagonal, with the corresponding eigenvalues (greater than or equal to 0) on the diagonal. Then \((X^\top X + \beta I_t) = Q^\top (\Lambda + \beta I)Q \). All elements on the diagonal of \((\Lambda + \beta I)\) is now greater than 0 as long as \( \beta > 0 \). Thus \((X^\top X + \beta I_t)^{-1} = Q^\top (\Lambda + \beta I)^{-1} Q \) 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 \( L (> -\infty, \text{ i.e., bounded from below}) \), the solution

\[
w^* = \arg\min_w L(Xw, y) + \beta \|w\|_2^2
\]

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

Definition 2. A is positive semidefinite (PSD). Or equivalently,  

Theorem 2. The following statements are equivalent:  

Proof. Any \( w \) can be factorized as \( w = w_0 + w_1 \), where \( w_1 \in \text{RowSpace}(X) \), that is, \( w_1 = X^\top \alpha \) for some \( \alpha \in \mathbb{R}^t \), and \( w_0 \perp \text{RowSpace}(X) \) such that \( Xw_0 = 0 \). First, because \( Xw = X(w_0 + w_1) = Xw_1 \), the \( w_0 \) component cannot affect the loss function \( L \). Second, since \( \|w\|^2 = \|w_0\|^2 + \|w_1\|^2 \), to minimize \( \beta\|w\|^2 \), we simply set \( w_0 \) to zero. Therefore, the optimal \( w^* \) should be \( w_1^* \), which is equal to \( X^\top \alpha \) for some \( \alpha \in \mathbb{R}^t \). \qed  

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

\[
\min_\alpha \, L(XX^\top \alpha, y) + \beta \alpha^\top XX^\top \alpha
\]

To predict the output of a new point \( x_0 \), instead of \( w^\top x_0 \), we compute its prediction as \( \hat{y} = \alpha^\top Xx_0 \). Furthermore, it turns out that we do not care about the input, and what really matters is the inner product of the inputs \( x_i^\top x_j \). With the notation \( K = XX^\top \) and \( k_0 = Xx_0 \), the learning problem (2.1) becomes  

\[
\min_\alpha \, L(K \alpha, y) + \beta \alpha^\top K \alpha
\]

and during test, \( \hat{y} = k_0^\top \alpha \). Recall that in previous chapter, we use a new feature vector \( \varphi(x) \) for training. By the representer theorem, we no longer need the feature vector explicitly if we have access to their inner product \( \varphi^\top(x_i) \varphi(x_j) \) directly. This technique of replacing inner product is sometimes called the 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 \( x_1, x_2, \ldots, x_t \in \mathcal{X} \),  

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

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 \( \Phi \).  

**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: \( \forall \alpha \),  

\[
a^\top Ka = a^\top \Phi \Phi^\top a
\]

\[
= v^\top v \quad \text{with} \quad v = \Phi^\top a
\]

\[
\geq 0
\]

2 \( \Rightarrow \) 3: Trivial, by definition.  
3 \( \Rightarrow \) 1: Let \( K = QAQ^\top \) be the eigen-decomposition of \( K \). Since all the eigenvalues of \( K \) are non-negative, \( \Phi = QA^\frac{1}{2} \) exists and \( K = \Phi \Phi^\top \). \qed  

### 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.
Consider a descent step from a subset of bounded size that minimizes training loss is NP-hard. Generally, finding 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.

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

Consider the problem of feature selection: given $l$ features $\varphi_i(x), i = 1, \cdots, l$, 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:

$$
\min_w \sum_i L(x_i^\top w, y_i) + \beta \|w\|_0, \quad \beta \geq 0,
$$

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

$$
\min_w \sum_i L(x_i^\top w, y_i) + \beta \|w\|_1, \quad \beta \geq 0. \tag{2.2}
$$

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

### Subgradient

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

$$
f(x) \geq f(x_0) + (x - x_0)^\top \nabla f(x_0),
$$

where $\nabla f(x_0)$ is the gradient of $f$ at $x_0$.

A subgradient at $x_0$ is any $d_0$ such that

$$
f(x) \geq f(x_0) + (x - x_0)^\top d_0.
$$

If $f(\cdot)$ is differentiable at $x_0$, then $d_0$ is unique and $d_0 = \nabla f(x_0)$. However, if $f(\cdot)$ is not differentiable at $x_0$, then $d_0$ is not unique (see the figure below).

Let $F(w)$ be our objective:

$$
F(w) = L(w) + \beta \|w\|_1 = \sum_i L(x_i^\top w, y_i) + \beta \|w\|_1.
$$

Consider a descent step from $w^{(k)}$, the parameters in the $k$th iteration. The partial derivative
w.r.t. $w_j^{(k)}$ is
\[
\frac{\partial F}{\partial w_j^{(k)}} = \frac{\partial L}{\partial w_j^{(k)}} + \beta \cdot \text{sign} \left( w_j^{(k)} \right) \\
= \sum_i L_i (x_i^\top w^{(k)}, y_i) X_{ij} + \beta \cdot \text{sign} \left( w_j^{(k)} \right), \quad \text{if } w_j^{(k)} \neq 0.
\] (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 $\left| \frac{\partial L}{\partial w_j^{(k)}} \right| = \left| \sum_i L_i (x_i^\top w, y_i) X_{ij} \right| < \beta$, then $w_j$ remains at 0, otherwise we can reduce the objective by moving $w_j$ away from 0, to the direction of $\left( -\frac{\partial L}{\partial w_j^{(k)}} \right)$. 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 $w^{(k)}$),

\[
L \left( w^{(k+1)} \right) \approx L \left( w^{(k)} \right) + \left( \nabla L (w^{(k)}) \right)^\top \left( w^{(k+1)} - w^{(k)} \right) \\
= L \left( w^{(k)} \right) - \eta \cdot \nabla L \left( w^{(k)} \right)^\top \nabla L \left( w^{(k)} \right) \quad \text{when } w^{(k+1)} = w^{(k)} - \eta \nabla L \left( w^{(k)} \right)
\]

We can see that contribution of the $j$th component to the difference in loss is (approximately) $-\eta \left( \frac{\partial L}{\partial w_j^{(k)}} \right)^2$. For the regularizer, the difference is $\beta \eta \left| \frac{\partial L}{\partial w_j^{(k)}} \right|$ (changing from $w_j^{(k)} = 0$ to $w_j^{(k+1)} = -\eta \frac{\partial L}{\partial w_j^{(k)}}$). 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 $\left| \frac{\partial L}{\partial w_j^{(k)}} \right| < \beta$, this quantity is positive. $w_j$ moving away from 0 will increase the objective.
- If $\left| \frac{\partial L}{\partial w_j^{(k)}} \right| > \beta$, this quantity is negative. $w_j$ moving towards $\left( -\frac{\partial L}{\partial w_j^{(k)}} \right)$ will reduce the objective.

Therefore, larger $\beta$ indicates more zeros in $w$. This sparse solution implicitly select features since $w_j = 0$ means the $j$th 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 $\xi \in \mathbb{R}^n$ and reformulate the problem as

\[
\min_{w, \xi} \sum_i L(x_i^\top w, y_i) + \beta 1_n^\top \xi \\
\text{s.t. } \xi \succeq w \\
\xi \succeq -w
\]

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

---

1We write $\frac{\partial F}{\partial w_j^{(k)}}$ as $\frac{\partial F}{\partial w_j}$
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 may need a special type of output. For instance, 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 $\hat{z} = w^T x$ be the pre-prediction and $\hat{y} = f(\hat{z})$ is the post-prediction.

- For $y \geq 0$, $f_1(\hat{z}) = e^{\hat{z}}$. That is, $\hat{y} = e^{w^T x}$.
- For $y \in [0, 1]$, $f_2(\hat{z}) = \sigma(\hat{z}) = (1 + e^{-\hat{z}})^{-1}$, where $\sigma(\cdot)$ is the so-called sigmoid function in the literature. So, $\hat{y} = (1 + e^{-w^T x})^{-1}$.
- For $y \in \{-1, +1\}$, $f_3(\hat{z}) = \text{sign}(\hat{z})$, which is the sign function. So, $\hat{y} = \text{sign}(w^T x)$.

Technically, when $\hat{z} = 0$, $\hat{y}$ becomes 0, so we need to specify this particular case as $\hat{y} = -1$ or $\hat{y} = +1$.

Figure 3.1 shows the above transform functions.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{transform_functions.pdf}
\caption{Transform functions examples.}
\end{figure}

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 $f(\cdot)$.

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

\[
L(\hat{y}, y) = L(f(\hat{z}), f(z)) = \int_z^{\hat{z}} (f(u) - f(z)) \, du
\]

\[
= F(\hat{z}) - F(z) - f(z)(\hat{z} - z)
\]

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

\[
F(\hat{z}) \geq F(z) + f(z)(\hat{z} - z) \implies L(\hat{y}, y) \geq 0.
\]

Moreover,

\[
L(f(\hat{z}), f(z)) = 0 \iff \hat{z} = z.
\]

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

Now let us look at our previous examples.

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

\[
L(f(\hat{z}), f(z)) = F(\hat{z}) - F(z) - f(z)(\hat{z} - z) = \frac{1}{2}z^2 - \frac{1}{2}z^2 - z(\hat{z} - z) = \frac{1}{2}(\hat{y} - y)^2 \quad \text{(squared loss, } L_2 \text{ loss)}.
\]

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

\[
L(f(\hat{z}), f(z)) = F(\hat{z}) - F(z) - f(z)(\hat{z} - z) = e^\hat{z} - e^z - e^z(\hat{z} - z) = \ln \frac{\hat{y}}{y} + \hat{y} - y \quad \text{(un-normalized entropy)}.
\]

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

\[
L(f(\hat{z}), f(z)) = F(\hat{z}) - F(z) - f(z)(\hat{z} - z) = \ln(e^\hat{z} + 1) - \ln(e^z + 1) - \sigma(z)(\hat{z} - z) = y \ln \frac{\hat{y}}{y} + (1 - y) \ln \frac{1 - y}{1 - \hat{y}} \quad \text{(cross entropy)}.
\]

Sadly, we do not have “the” matching loss for \( \text{sign}(\cdot) \) 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:

\[
L(\hat{y}, y) = \mathbb{1}(\hat{y} \neq y) \quad (3.2)
\]

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

\[
\min_{w, b} \sum_{i=1}^{t} \mathbb{1}(\hat{y}_i \neq y_i)
\]

\[
\text{s.t. } \hat{y} = \text{sign}(Xw - b1_t)
\]

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 \( \delta \in \mathbb{R} \):

\[
\max_{w, b, \delta} \delta
\]

\[
\text{s.t. } y_i(w^T x_i - b) \geq \delta \quad \forall i
\]

\[\delta \leq 1.\]
The last constraint ($\delta \leq 1$) is necessary because otherwise we can amplify $w$, $b$ and $\delta$ 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 $\xi \in \mathbb{R}^t$,

$$
\min_{w/b/\xi} \xi^T 1_t
$$

s.t. $y_i(w^T x_i - b) \geq 1 - \xi_i \quad \forall i$

$\xi \leq 0_t$.

After solving the problem, the dataset is

- linearly separable if and only if $\xi = 0_t$, 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

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

Luckily, this replacement recovers differentiability and strictly increasing property, so we can construct its matching loss and hence, convex objective. However, the transformed output $\hat{y}$ 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 $\hat{z}$. That is $m = y \hat{z}$.

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

$$
L(m) = (w^T x - y)^2 = (\hat{z} - y)^2 = \hat{z}^2 - 2\hat{z}y + y^2 = m^2 - 2m + 1 = (m - 1)^2.
$$

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 ($\hat{y} = \text{sign}(w^T x)$). 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 $\hat{z}$, 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 $\hat{z}$.

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 \((w, b)\) pairs that are consistent with the dataset as shown in Figure 4.1.

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.

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

\[
\delta_i = \frac{|\mathbf{w}^\top x_i - b|}{||\mathbf{w}||_2}.
\]

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

\[
\delta_i = \frac{y_i (\mathbf{w}^\top x_i - b)}{||\mathbf{w}||_2}.
\]
Therefore, formally speaking, to find the maximum margin classifier, we solve the following problem:

$$\max_{w, b} \min_i \delta_i$$

$$\iff \max_{w, b, \gamma} \gamma \text{ s.t. } \frac{y_i (w^\top x_i - b)}{||w||_2} \geq \gamma \forall i$$

$$\iff \max_{w, b, \gamma} \gamma \text{ s.t. } y_i (w^\top x_i - b) \geq \gamma ||w||_2 \forall i,$$

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

$$\max_{w, b} \frac{1}{||w||_2} \text{ s.t. } y_i (w^\top x_i - b) \geq 1, \forall i$$

$$\iff \min_{w, b} \frac{1}{2} ||w||_2^2 \text{ s.t. } y_i (w^\top x_i - b) \geq 1, \forall i,$$

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.

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

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

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 $\xi_i$ to each margin constraint and at the same time, try to minimize minimum margin at the price of sum of slacks. Formally,

$$\min_{w, b, \xi} \frac{\beta}{2} ||w||_2^2 + |\xi| 1_i$$

$$\text{ s.t. } y_i (w^\top x_i - b) \geq 1 - \xi_i, \forall i$$

(4.1)

$\beta > 0$ is a tuning trade-off parameter. $\xi_i$ 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.

![Soft SVM with slack variable.](image)

One interesting fact is that if we treat $\frac{\beta}{2} ||w||_2^2$ 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:

$$\min_{w} \ l(w)$$

subject to:

$$Aw \geq b$$

$$Cw = d,$$

which is called the primal problem. Its Lagrangian is

$$L(w, \lambda, \mu) = l(w) + \lambda^T (b - Aw) + \mu^T (d - Cw)$$

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

$$D(\lambda, \mu) = \min_w L(w, \lambda, \mu)$$

$$= \min_w l(w) + \lambda^T (b - Aw) + \mu^T (d - Cw)$$

The dual function has two important properties:

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

Let

$$p^* = \min_w l(w) \quad \text{s.t.} \quad Aw \geq b, Cw = d$$

$$w^* = \arg\min_w l(w) \quad \text{s.t.} \quad Aw \geq b, Cw = d.$$  

The following theorem states weak duality:

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

**Proof.** Fix any $$\lambda \succeq 0, \mu$$

$$D(\lambda, \mu) = \min_w l(w) + \lambda^T (b - Aw) + \mu^T (d - Cw)$$

$$\leq l(w^*) + \lambda^T (b - Aw^*) + \mu^T (d - Cw^*)$$

$$\leq l(w^*) = p^*$$

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

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

$$\max_{\lambda \succeq \theta, \mu} D(\lambda, \mu)$$

$$\iff \max_{\lambda \succeq \theta, \mu} \min_w l(w) + \lambda^T (b - Aw) + \mu^T (d - Cw).$$

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

Let

$$d^* = \max_{\lambda \succeq \theta, \mu} D(\lambda, \mu)$$

$$(\lambda^*, \mu^*) = \arg\max_{\lambda \succeq \theta, \mu} D(\lambda, \mu).$$

The following theorem states strong duality:

**Theorem 4 (ref).** If $$l(w)$$ is convex in $$w$$ and $$Aw \succeq b, Cw = d$$ are strictly feasible (i.e., there exists $$w_0$$ 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 \( \mathbf{w}^* \) from \( \lambda^* \) and \( \mu^* \). The following theorem provides an effective approach to recover \( \mathbf{w}^* \).

**Theorem 5.** If strong duality holds, then

\[
\max_{\lambda \geq 0, \mu} \min_{\mathbf{w}} L(\mathbf{w}, \lambda, \mu) = \min_{\mathbf{w}} \max_{\lambda \geq 0, \mu} L(\mathbf{w}, \lambda, \mu).
\]

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

\[
l(\mathbf{w}) \geq \max_{\lambda \geq 0, \mu} l(\mathbf{w}) + \lambda^T (\mathbf{b} - A\mathbf{w}) + \mu^T (\mathbf{d} - C\mathbf{w}) = \max_{\lambda \geq 0, \mu} L(\mathbf{w}, \lambda, \mu). \tag{4.2}
\]

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

\[
l(\mathbf{w}^*) \geq \max_{\lambda \geq 0, \mu} L(\mathbf{w}^*, \lambda, \mu) \geq \min_{\mathbf{w}} \max_{\lambda \geq 0, \mu} L(\mathbf{w}, \lambda, \mu).
\]

This inequality means

\[
\min_{\mathbf{w}: A\mathbf{w} \geq \mathbf{b}, C\mathbf{w} = \mathbf{d}} l(\mathbf{w}) \geq \min_{\mathbf{w}: A\mathbf{w} \geq \mathbf{b}, C\mathbf{w} = \mathbf{d}} \max_{\lambda \geq 0, \mu} L(\mathbf{w}, \lambda, \mu) \geq \min_{\mathbf{w}} \max_{\lambda \geq 0, \mu} L(\mathbf{w}, \lambda, \mu) \tag{4.3}
\]

Now we are ready to prove the theorem:

\[
\min_{\mathbf{w}} \max_{\lambda \geq 0, \mu} L(\mathbf{w}, \lambda, \mu) \geq \max_{\lambda \geq 0, \mu} \min_{\mathbf{w}} L(\mathbf{w}, \lambda, \mu) \quad \text{(by max-min inequality)}
\]

\[
= \min_{\mathbf{w}: A\mathbf{w} \geq \mathbf{b}, C\mathbf{w} = \mathbf{d}} l(\mathbf{w}) \quad \text{(by strong duality)} \tag{4.4}
\]

\[
\geq \min_{\mathbf{w}} \max_{\lambda \geq 0, \mu} L(\mathbf{w}, \lambda, \mu) \quad \text{(by the lemma 4.3)}
\]

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. \( \square \)

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

- if \( \lambda_i^* > 0 \), then \( A_i \mathbf{w}^* = \mathbf{b}_i \);
- if \( A_i \mathbf{w}^* > \mathbf{b}_i \), then \( \lambda_i^* = 0 \).

To see this, note that:

\[
l(\mathbf{w}^*) = \max_{\lambda \geq 0, \mu} \min_{\mathbf{w}} L(\mathbf{w}, \lambda, \mu) = \text{D}(\lambda^*, \mu^*) = \min_{\mathbf{w}} L(\mathbf{w}, \lambda^*, \mu^*) \leq L(\mathbf{w}^*, \lambda^*, \mu^*) \leq l(\mathbf{w}^*)
\]

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

Now, back to our question on finding \( \mathbf{w}^* \) given \( \lambda^* \) and \( \mu^* \). With the help of complementary slackness property, we can find \( \mathbf{w}^* \) such that \( A\mathbf{w}^* = \mathbf{b}, C\mathbf{w}^* = \mathbf{d} \), where \( A, \mathbf{b} \) 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):

$$\min_{w, b} \frac{\beta}{2} \|w\|^2 + \sum_{i=1}^{T} \max \left(0, 1 - y_i (w^\top x_i - b) \right)$$

which, in matrix form, is equivalent to

$$\min_{w, b, \xi} \frac{\beta}{2} \|w\|^2 + \xi^\top 1_t$$

s.t. $\xi \succeq 1_t - \Delta(y)(Xw - b1_t)$

$\xi \succeq 0_t,$

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

$$L(w, b, \xi, \lambda, \mu) = \frac{\beta}{2} w^\top w + \xi^\top 1_t + \mu^\top (0_t - \xi) + \lambda^\top [1_t - \Delta(y)(Xw - b1_t)] - \xi$$

$$= \frac{\beta}{2} w^\top w + \xi^\top (1_t - \mu - \lambda) + \lambda^\top [1_t - \Delta(y)(Xw - b1_t)].$$

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

$$\frac{\partial L}{\partial \xi} = 1_t - \mu - \lambda = 0$$

$$\implies \mu = 1_t - \lambda$$

$$\implies \lambda \succeq 1_t \quad \text{(as } \mu \succeq 0_t)$$

This eliminates $\xi$ and $\mu$ simultaneously:

$$L(w, b, \lambda) = \frac{\beta}{2} w^\top w + \lambda^\top [1_t - \Delta(y)(Xw - b1_t)] \quad \text{for } 0_t \succeq \lambda \succeq 1_t.$$  

Next, we eliminate $b$:

$$\frac{\partial L}{\partial b} = \lambda^\top y = 0,$$

$$\implies L(w, \lambda) = \frac{\beta}{2} w^\top w + \lambda^\top 1_t - \lambda^\top \Delta(y)Xw \quad \text{for } 0_t \succeq \lambda \succeq 1_t, \lambda^\top y = 0.$$

Finally, we eliminate $w$:

$$\frac{\partial L}{\partial w} = \beta w - X^\top \Delta(y)\lambda = 0$$

$$\implies w = \frac{1}{\beta} X^\top \Delta(y)\lambda,$$

$$\implies L(\lambda) = \lambda^\top 1_t - \frac{1}{2\beta} \lambda^\top \Delta(y)XX^\top \Delta(y)\lambda \quad \text{for } 0_t \succeq \lambda \succeq 1_t, \lambda^\top y = 0.$$  

Therefore, the dual problem of the regularized hinge loss is

$$\max_{\lambda} \quad \lambda^\top 1_t - \frac{1}{2\beta} \lambda^\top \Delta(y)XX^\top \Delta(y)\lambda$$

s.t. $0_t \succeq \lambda \succeq 1_t,$

$$\lambda^\top y = 0. \quad (4.5)$$

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

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

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

$$\lambda^*_t > 0 \implies 1 - y_i(X_i, w^* - b^*) - \xi^*_t = 0,$$

$$\lambda^*_t < 1 \implies \mu^*_t > 0 \implies \xi^*_t = 0.$$
Therefore,

\[ y_i (X_i \mathbf{w}^* - b^*) = 1 \]

\[ \implies b^* = X_i \mathbf{w}^* - y_i = \frac{1}{\beta} X_i \mathbf{X}^\top \Delta(y) \mathbf{\lambda}^* - y_i. \]

For test point \( x \),

\[ \hat{y} = \text{sign}(\mathbf{w}^\top \mathbf{x} - b^*) \]

\[ = \text{sign} \left( \frac{1}{\beta} \mathbf{x}^\top \mathbf{X}^\top \Delta(y) \mathbf{\lambda}^* - b^* \right) \]

Note that the above classification rule and problem (4.5) can be kernelized. All we need is the inner product \( \mathbf{x}_i^\top \mathbf{x}_j \).
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 \( y_i \in \mathbb{R}^k \)? Consider the linear prediction \( y_i = w^\top x_i \), because of multiple outputs, the linear model \( w \) becomes a matrix \( W \in \mathbb{R}^{n \times k} \), whose rows correspond to features and columns correspond to outputs. That is \( y_i = x_i^\top W \). The matrix form is

\[
\hat{Y} = XW
\]

where \( X \in \mathbb{R}^{t \times n} \), \( \hat{Y} \in \mathbb{R}^{t \times k} \).

\[ \text{Square loss with multiple outputs} \]

\[
\min_W \sum_{i=1}^t \sum_{j=1}^k (X_i W_{ij} - Y_{ij})^2 \\
\Leftrightarrow \min_W \text{tr}[(XW - Y)^\top (XW - Y)] \\
\Leftrightarrow \min_W \text{tr}(W^\top X^\top XW) - 2\text{tr}(Y^\top XW) + \text{tr}(Y^\top Y).
\]

Setting its derivative with respect to \( W \) to zero:

\[
\frac{d\text{Obj}}{dW} = 2X^\top X - 2X^\top Y = 0 \implies X^\top XW = X^\top Y.
\]

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: \( W = (X'X) \backslash (X'Y) \).)

\[ \text{Trace of matrix} \]

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

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

The derivative \( a \) of matrix trace:

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

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

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

Analogize to the $L_2$ regularizer, for matrix $W$, $R(W) = \sum_{ij} W_{ij}^2 = \text{tr}(W^T W) = \|W\|_F^2$, where $\| \cdot \|_F$ is the Frobenius norm. The representer theorem holds: for Frobenius norm $\min_W \beta \frac{1}{2} \text{tr}(W^T W) + L(XW, Y)$, the optimal $W^* = X^T A$ for some $A \in \mathbb{R}^{t \times k}$. Therefore, the adjoint form is

$$\min_A \frac{\beta}{2} \text{tr}(A^T XX^T A) + L(XX^T A, Y).$$

To predict the output of a new point $x$,

$$\hat{y}^T = x^T W^* = x^T X^T A^*.$$

### 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 $0 \leq \alpha \leq 1$,

$$\|\alpha X + (1 - \alpha)Y\| \leq \|\alpha X\| + \|(1 - \alpha)Y\|$$

$$= |\alpha| \|X\| + |1 - \alpha| \|Y\|$$

$$= \alpha \|X\| + (1 - \alpha) \|Y\|.$$

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.
5.1. MULTIVARIATE PREDICTION

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

$$\|W\|_p = \max_{x \neq 0} \frac{\|Wx\|_p}{\|x\|_p}.$$  

**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.

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

is the biggest singular value of $W$.

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

**Schatten $p$-norm examples**

- **Trace norm:**

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

- **Spectral norm:**

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

- **Frobenius norm:**

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

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

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

which looks complicated. In fact, assume $W \in \mathbb{R}^{n \times m}$, let $v \in \mathbb{R}^n$, whose entries are $v_i = \|W_{i,:}\|_q$, then $\|W\|_{p,q} = \|v\|_p$. 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

$$\min_W \frac{\beta}{2}\|W\|_{2,1}^2 + L(XW,Y).$$
Table 5.1: Matrix norm concrete examples.

<table>
<thead>
<tr>
<th>Norm</th>
<th>U</th>
<th>V</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>∥·∥_1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>∥·∥_∞</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>∥·∥_2</td>
<td>√2</td>
<td>√2</td>
<td>1</td>
</tr>
<tr>
<td>∥·∥_tr</td>
<td>√2</td>
<td>√2</td>
<td>2</td>
</tr>
<tr>
<td>∥·∥_F</td>
<td>√2</td>
<td>√2</td>
<td>√2</td>
</tr>
<tr>
<td>∥·∥_2,1</td>
<td>√2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

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 \( y_i \) 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 \( ∥·∥_{tr} \) is our choice according to Table 5.1, because \( U \) and \( V \) are penalized less compared to \( W \). Then we optimize

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

### 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 \( \hat{z}^\top = x^\top W \) is pre-prediction, then \( \hat{y} = f(\hat{z}) \) is post-prediction.

- For \( y \succeq 0_k \), \( f(z) = e^z \);
- For \( y \succeq 0_k, y^\top 1_k = 1 \), \( f(z) = e^z / (1_k^\top e^z) \), which is sometimes called “softmax” transform in machine learning literature;
- For \( y = 1_c = [0,0,\cdots,1,0,0,\cdots,0]^\top \), which is a class indicator that has 1 in the \( c \)th location and 0s everywhere else, \( f(z) = \text{indmax}(z) \), 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 \( f(z) = \nabla F(z) \). Then define loss function as

\[
L(\hat{y}, y) = L(f(\hat{z}), f(z))
= F(\hat{z}) - F(z) - f^\top(z)(\hat{z} - z)
\]

Because of strict convexity, we have

\[
F(\hat{z}) \geq F(z) + f^\top(z)(\hat{z} - z).
\]

Hence, \( L(\hat{y}, y) \geq 0 \) and

\[
L(\hat{y}, y) = 0 \iff f(\hat{z}) = f(z).
\]

For previous examples:

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

\[
L(\hat{y}, y) = F(\hat{z}) - F(z) - f^\top(z)(\hat{z} - z)
= 1_k^\top e^z - 1_k^\top e^z - (e^z)^\top(\hat{z} - z)
= 1_k^\top \hat{y} - 1_k^\top y - y^\top(\ln \hat{y} - \ln y)
= y^\top(\ln y - \ln \hat{y}) + 1_k^\top(\hat{y} - y)
\]

which is known as un-normalized entropy.
5.2. STRUCTURED OUTPUT

- \( y \succeq 0, y^\top 1_k = 1 \). \( f(z) = e^z / (1_k^\top e^z), F(z) = \ln (1_k^\top e^z) \). So,
  \[
  L(\hat{y}, y) = F(\hat{z}) - F(z) - f^\top (\hat{z} - z)
  \]
  \[
  = \ln (1_k^\top e^{\hat{z}}) - \ln (1_k^\top e^z) - (e^z / 1_k^\top e^z)^\top (\hat{z} - z)
  \]
  \[
  = 1_k^\top (e^z / 1_k^\top e^z) \ln (1_k^\top e^{\hat{z}}) - 1_k^\top (e^z / 1_k^\top e^z) \ln (1_k^\top e^z) - (e^z / 1_k^\top e^z)^\top (\ln e^{\hat{z}} - \ln e^z)
  \]
  \[
  = (e^z / 1_k^\top e^z)^\top (1_k \ln (1_k^\top e^{\hat{z}}) - 1_k \ln (1_k^\top e^z) - \ln e^{\hat{z}} + \ln e^z)
  \]
  \[
  = (e^z / 1_k^\top e^z)^\top (\ln e^{\hat{z}} - \ln e^z)
  \]
  which is known as KL-divergence.

5.1.5 Surrogate Loss for Classification

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

\[
L(\hat{y}, y) = I(y \neq \hat{y}).
\]

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

- Multinomial deviance (logistic margin loss).
  \[
  L(\hat{z}, y) = \ln (1_k^\top e^{\hat{z}}) - y^\top \hat{z}.
  \]

- Multiclass SVM loss (analogized to hinge loss in univariate case).
  \[
  L(\hat{z}, y) = \max(\ 1_k - y + \hat{z} - y^\top \hat{z} 1_k )
  \]

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

Therefore, the multiclass SVM is

\[
\min \frac{\beta}{2} ||W||_F^2 + \sum_i \max(1_k^\top - Y_i + X_i W - X_i W Y_i^\top 1_k^\top )
\]

\[
\iff \min \frac{\beta}{2} ||W||_F^2 + 1_k^\top \xi
\]

\[
s.t. \ \xi 1_k^\top \succeq 1, 1_k^\top - Y + XW - \delta(W Y Y^\top) 1_k^\top
\]

where \( \delta(A) \) is the column vector of the diagonal of \( A \) (opposite of \( \Delta(A) \)) 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 \( x_0 \),

\[
\hat{y}_0 = \text{indmax}(x_0^\top W)
\]

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 red apple”. 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”, “c” is much more likely than “q”; “a” is much more likely than “apple”.

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

$$\tilde{\varphi}(x_i, c_i) = \sum_{j=1}^{l-1} \varphi(x_i, c_{ij}, c_{i(j+1)}).$$

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

$$\hat{z}_i = \tilde{w}^\top \tilde{\varphi}(x_i, c_i) = \sum_{j=1}^{l-1} w^\top \varphi(x_i, c_{ij}, c_{i(j+1)}).$$

This decouple approach maintains (at least 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 $1_k^\top \epsilon_{\hat{z}_i} = \sum e_{\hat{z}_i}$ is hard when $k$, the number of class, is really large. However, with the help of decouple formulation, we have

$$\sum e_{\hat{z}_i} = \sum_{\{c_1, \ldots, c_l\}} \exp \left( \sum_{j=1}^{l-1} w^\top \varphi(x_i, c_{ij}, c_{i(j+1)}) \right) = \sum_{\{c_1, \ldots, c_l\}} \prod_{j=1}^{l-1} \exp (w^\top \varphi(x_i, c_{ij}, c_{i(j+1)})),
$$

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,

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

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

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

$$\max \hat{z}_i = \max_{\{c_1, \ldots, c_l\}} \left( \sum_{j=1}^{l-1} w^\top \varphi(x_i, c_{ij}, c_{i(j+1)}) \right),$$

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

“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,

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

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

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 \(h : \mathcal{X} \rightarrow \mathcal{Y}\) for specific \(Y\)? 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 \(Y_{1 \times k} = 1_t\).
- **Soft clustering**: \(Y \in \mathbb{R}^{t \times k}\), such that \(Y_{1 \times k} = 1_t\).

### 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 \(Y\),

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

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 \(\Sigma_X\) 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 \(Y\),

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

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

\[
X^\top XW = X^\top Y = U^\top Y^\top Y \\
\Rightarrow W = (X^\top X)^{-1}U^\top (Y^\top Y), \\
U = (Y^\top Y)^{-1}W^\top (X^\top X)
\]

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

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

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

\[
\min_{Y, U} \frac{1}{2} \|X - YU\|_F^2. \tag{6.1}
\]
Fix $Y$, we know that $U^* = Y^+X$. Plug in the reverse formulation, we have
\[
\begin{align*}
\min_Y & \quad \frac{1}{2} \|X - YY^+X\|_F^2 \\
\implies \min_Y & \quad \frac{1}{2} \text{tr} \left[(I_t - YY^+)XX^T(I_t - YY^+)^T\right] \\
\implies \min_Y & \quad \frac{1}{2} \text{tr} \left[(I_t - YY^+)K\right] \\
\max_Y & \quad \text{tr}(YY^+K),
\end{align*}
\]
(6.2)
where $K = XX^T$ is the kernel matrix. The second derivation above is based on the fact that $YY^+$ is symmetric and $\text{tr}(YY^+XX^TYY^+) = \text{tr}(YY^+XX^T) = \text{tr}(YY^+XX^T)$. Let the SVD of $Y$ be $Y = Q_Y \Sigma_Y V_Y^T$, then $Y^+ = V_Y \Sigma_Y^{-1} Q_Y^T$, $YY^+ = Q_Y Q_Y^T$. With such notation, the reverse formulation becomes
\[
\begin{align*}
\max_{Q_Y} & \quad \text{tr}(Q_Y Q_Y^T K), \quad \text{s.t.} \quad Q_Y^T Q_Y = I_{\text{rank}(Y)} \\
\implies \max_{Q_Y} & \quad \text{tr}(Q_Y^T K Q_Y), \quad \text{s.t.} \quad Q_Y^T Q_Y = I_{\text{rank}(Y)} \\
\implies \max_{Q} & \quad \sum_{i=1}^{\text{rank}(Y)} q_i^T K q_i,
\end{align*}
\]
where $q_i$ are mutually orthogonal with unit length. Assume $\text{rank}(Y) = k$, it is intuitive to see that by setting the $q_i$ 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 $K = XX^T$, with $X = Q_X \Sigma_X V_X^T$, then it is equivalent to setting $q_i$ to be the top $k$ left singular vectors of $X$.

To conclude, optimal solutions are given by $Y^* = Q_{X;1:k}$, $U^* = Q_{X;1:k}^T$. The original data matrix $X$ is approximated by $X \approx Q_{X;1:k} Q_{X;1:k}^T X$. 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
\[
W = (X^T X)^{-1} U^T (Y^T Y) = X^+ Q_{X;1:k} = V_{X;1:k} \Sigma_{X;1:k}^{-1}.
\]

To this point, we have done dimensionality reduction.

### Notes

**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:
\[
x_i \leftarrow x_i - \frac{1}{t} \sum_{j=1}^t x_j.
\]
(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 $X$ 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 $\hat{y}^T = \text{indmax}(x^T W)$. The regular supervised problem with square surrogate loss is
\[
\min_{W,Y \in \mathcal{Y}} \frac{1}{2} \|XW - Y\|_F^2,
\]
where $\mathcal{Y} = \{Y \in \{0,1\}^{t \times k}, Y \mathbf{1}_k = \mathbf{1}_t\}$. Its reverse formulation is
\[
\min_{U,Y \in \mathcal{Y}} \frac{1}{2} \|X - YU\|_F^2.
\]
6.3. KERNELIZATION

- Given fixed \( Y \), we already know that the optimal \( U \) is given by \( U^* = Y^+X = (Y^TY)^{-1}Y^TX \). Note that \( Y^TY = \Delta([t_{c_1}, t_{c_2}, \ldots, t_{c_d}]^\top) \), where \( t_{c_j} \) is the number of points in the \( j \)th cluster \( c_j \), i.e., \( Y^TY \) is a diagonal matrix with numbers of points of each cluster on the diagonal. Also note that \( Y^TX = [\sum_{x \in c_1} x; \sum_{x \in c_2} x; \ldots; \sum_{x \in c_d} x]^\top \).

  To combine, \( (Y^TY)^{-1}Y^TX = Y^+X = [m_1, m_2, \ldots, m_d]^\top \), where \( m_j \) 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

\[
\|X - YU\|^2_F = \sum_i \|X_i - Y_i U\|^2_2.
\]

Due to the fact that \( Y_i = [0, 0, \ldots, 1, \ldots, 0] \), it only has a single one indicating that the \( i \)th point belongs to the \( j \)th cluster. Obviously, the best strategy is to assign the \( i \)th data point to the \( j \)th 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 \( (x - u)^2 = x^2 + u^2 - 2xu \))

\[
D(X, U) = \delta(XX^\top)1_k^\top + 1_t \delta(UU^\top) - 2XU^\top.
\]

Then we have \( Y_i^* \) equals to \([0, 0, \ldots, 1, \ldots, 0] \), where the single one is in the \( j \)th entry and \( j = \text{argmin}_j D_{ij} \).

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

\[
\min_{Y \in \mathcal{Y}} \frac{1}{2} \|X - Y(Y^TY)^{-1}Y^TX\|^2_F.
\]

after plugging in \( U^* = Y^+X = (Y^TY)^{-1}Y^TX \). 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

\[
\begin{align*}
\min_W & \quad \frac{\beta}{2} \|W\|^2_F + \frac{1}{2} \|XW - Y\|^2_F \\
\iff \min_A & \quad \frac{\beta}{2} \text{tr}(A^TKA) + \frac{1}{2} \|KA - Y\|^2_F
\end{align*}
\]

which is based on \( W^* = X^\top A^* \) and the representer Theorem. Its derivative

\[
\frac{d\text{Obj}}{dA} = \beta KA + K(KA - Y) = 0
\]

\[
\iff K(K + \beta I_t)A = KY
\]

\[
\iff (K + \beta I_t)A = Y
\]

\[
\iff A = (K + \beta I_t)^{-1}Y.
\]

\[\text{(6.4)}\]

\[\text{(6.5)}\]
The reverse formulation is
\[
\min_U \frac{1}{2} \|X - YU\|^2_F.
\]
It is easy to see that \(U^* = Y^+ X = BX\) for some particular \(B \in \mathbb{R}^{k \times t}\). Therefore,
\[
\min_U \frac{1}{2} \|X - YU\|^2_F \iff \min_B \frac{1}{2} \|X - YBX\|^2_F \iff \min_B \frac{1}{2} \text{tr} \left( (I - YB)K(I - YB)^\top \right)
\]
We take the derivative
\[
\frac{d \text{Obj}}{dB} = \beta Y^\top (YB - I_t)K = 0 \\
\iff Y^\top YBK = Y^\top K \\
\iff Y^\top YB = Y^\top \\
\iff B = (Y^\top Y)^{-1} Y^\top = Y^+.
\]
Here we may assume that \(Y^\top Y\) is invertible (\(\text{rank}(Y) = k, k < t\)). From Eq. (6.5) and Eq. (6.6), we know that
\[
(K + \beta I_t)A = Y = B^\top Y^\top Y \\
A = (K + \beta I_t)^{-1} B^\top Y^\top Y.
\]
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.,
\[
\min_{B,Y} \frac{1}{2} \text{tr} \left( (I_t - YB)K(I_t - YB)^\top \right).
\]
With \(B^* = Y^+\), we have
\[
\min_Y \frac{1}{2} \text{tr} \left( (I_t - YY^+)K(I_t - YY^+)^\top \right).
\]
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 \(Y = \{Y \in \{0,1\}^{t \times k}, Y1_k = 1_t\}\). With square surrogate loss, we have
\[
\min_{B,Y \in Y} \frac{1}{2} \text{tr} \left( (I_t - YB)K(I_t - YB)^\top \right).
\]
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
\[
D(X, U) = \delta(XX^\top)1_k^\top + 1_t^\top \delta(UU^\top) - 2XU^\top \\
= \delta(K)1_k^\top + 1_t^\top \delta(BKB^\top) - 2KB^\top.
\]
This is analogized to the Euclidean distance matrix of each point \( x_i \) to the cluster center \( U_j \). Now given \( B \), we assign \( Y_j \) to be \([0, 0, \cdots, 1, \cdots, 0]\), where the single one is in the \( j \)th entry and \( j = \text{argmin}_j \ D_{ij} \). 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 \( x_0 \), its cluster index is computed as

\[
\hat{y}_0^\top = \text{indmax}(x_0^\top W)
= \text{indmax}(x_0^\top X^\top A)
= \text{indmax}(k_0^\top A),
\]

where \( k_0^\top = \varphi(x_0)^\top \Phi^\top \) is the inner product of (feature-extended) new point \( x_0 \) and (feature-extended) training set \( \Phi \).

### 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

\[
\Lambda = \Delta(K1_t)
\tilde{K} = \Lambda^{\frac{1}{2}} K \Lambda^{-\frac{1}{2}}
\tilde{A} = \Lambda^{\frac{1}{2}} A
\tilde{Y} = A^\frac{1}{2} Y
\tilde{B} = B \Lambda^{\frac{1}{2}}.
\]

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

\[
\min_\Lambda \frac{\beta}{2} \text{tr}(A^\top \Lambda^{-1} K \Lambda^{-1} A) + \frac{1}{2} \text{tr}(\Lambda(\Lambda^{-1} K \Lambda^{-1} A - Y)(\Lambda^{-1} K \Lambda^{-1} A - Y)^\top). \quad (6.8)
\]

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 \( \Lambda = \Delta(K1_t) \). Moreover, the square loss is also rescaled according to the instance importance \( \Lambda \). It is equivalent to

\[
\min_\Lambda \frac{\beta}{2} \text{tr}(\tilde{A}^\top \Lambda \tilde{A}) + \frac{1}{2} \text{tr}
\left((\tilde{K} \Lambda \tilde{A} - \tilde{Y})(\tilde{K} \Lambda \tilde{A} - \tilde{Y})^\top\right).
\quad (6.9)
\]

After taking derivative w.r.t. \( \tilde{A} \) and setting it to zero, we have

\[
A^* = \Lambda^\frac{1}{2}(\tilde{K} + \beta I_t)^{-1} \Lambda^\frac{1}{2} Y. \quad (6.10)
\]

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

\[
Y \approx \Lambda^{-1} K \Lambda^{-1} A^* = \Lambda^{-1} X X^\top \Lambda^{-1} A^* = \Lambda^{-1} X W^*.
\]

This can be seen as to rescale input \( X \) then predict via projection \( W^* \). For a new data point \( x \), define \( \lambda = x^\top X^\top \), as normalization constant. To predict its label,

\[
\hat{y} = \lambda^{-1} x^\top W^* = \frac{k^\top \Lambda^{-1} A^*}{k^\top 1_t}.
\]

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

\[
\min_{\tilde{B}, \tilde{Y}} \frac{1}{2} \text{tr} \left((I_t - \tilde{Y} \tilde{B}) \tilde{K}(I_t - \tilde{Y} \tilde{B})^\top\right)
\Rightarrow \min_{\tilde{Y}} \frac{1}{2} \text{tr} \left((I_t - \tilde{Y} \tilde{Y}^+) \tilde{K}(I_t - \tilde{Y} \tilde{Y}^+)\right),
\quad (6.11)
\]

with

\[
B^* = \tilde{Y}^+ = (Y^\top \Lambda Y)^{-1} Y^\top.
\quad (6.12)
\]
Similar to KPCA, the optimal \( \tilde{Y}^* \) is the top \( k \) eigenvectors of \( \tilde{K} \). Combining (6.12) and (6.10), we have

\[
\Lambda^{-1/2}(\tilde{K} + \beta I_t)\Lambda^{-1/2}A = Y = B^\top (Y^{\top} \Lambda Y)
\]

\[
A = \Lambda^{1/2}(\tilde{K} + \beta I_t)^{-1}\Lambda^{1/2}B^\top (Y^{\top} \Lambda Y).
\]

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 \( Y_1 = 1 \), That is, the first component of the new representation \( y_i \) is always 1 for very point \( i \). To see this, let \( \Lambda^{1/2} = \Lambda^{1/2}1_t \), then

\[
\tilde{K}\Lambda^{1/2} = \Lambda^{-1/2}K\Lambda^{-1/2}\Lambda^{1/2}1_t
\]

\[
= \Lambda^{-1/2}K1_t
\]

\[
= \Lambda^{-1/2}\Lambda1_t \quad \text{since} \quad \Lambda = \Delta(K1_t) \Rightarrow \Lambda1_t = K1_t
\]

\[
= \Lambda^{1/2}1_t = \lambda^{1/2}
\]

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

\[
Y_1 = \Lambda_{-1/2}\tilde{Y}_1 = \Lambda_{-1/2}\lambda^{1/2} = \Lambda_{-1/2}\lambda^{1/2} = 1_t.
\]

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

\[
X \leftarrow \left( I_t - \frac{1}{t}1_t1_t^\top \right) X.
\]

\( H = I_t - \frac{1}{t}1_t1_t^\top \) 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 \( \Phi \) 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),

\[
\min_{B,Y \in \mathcal{Y}} \frac{1}{2} \text{tr}\left( (I_t - \tilde{Y}\tilde{B})\tilde{K}(I_t - \tilde{Y}\tilde{B})^\top \right)
\]

\[
\iff \min_{B,Y \in \mathcal{Y}} \frac{1}{2} \text{tr}\left( \Lambda(\Lambda^{-1} - YB)K(\Lambda^{-1} - YB)^\top \right)
\]

where \( \mathcal{Y} = \{ Y \in \{0,1\}^{t \times k}, Y1_k = 1 \} \) 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

\[
D(\Lambda^{-1}X, BX) = \delta(\Lambda^{-1}XX^\top \Lambda^{-1})1_k^\top + 1_t^\top \delta(BXX^\top B^\top)^\top - 2\Lambda^{-1}XX^\top B^\top
\]

\[
= \delta(\Lambda^{-1}K\Lambda^{-1})1_k^\top + 1_t^\top \delta(BKB^\top)^\top - 2\Lambda^{-1}KB^\top.
\]

Then \( Y^*_j \) equals to \([0, 0, \ldots, 1, \ldots, 0]\), where the single one is in the \( j \)th entry and \( j = \arg\min_j D_{ij} \).

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

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:

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

where

- $\mathcal{Y} = \{Y \in \{0, 1\}^{t \times k}, Y \mathbf{1}_k = \mathbf{1}_t\}$;
- $\text{cut}(\text{class}_j, \overline{\text{class}_j})$ is the cost of cutting class $j$ from the rest of the graph:
  $$\text{cut}(\text{class}_j, \overline{\text{class}_j}) = \frac{1}{2} \sum_{i,l : Y_{ij} \neq Y_{lj}} K_{il},$$
  which is the sum of edges being cut;
- $\text{vol}(\text{class}_j)$ is considered to be the volume of class $j$:
  $$\text{vol}(\text{class}_j) = \sum_{i : Y_{ij} = 1} K_{il},$$
  which measures the association of class $j$ and the rest of the graph.

Note that

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

$$= \frac{1}{2} \sum_{i,l} K_{il}(Y_{ij}^2 + Y_{lj}^2 - 2Y_{ij}Y_{lj})$$

$$= Y_{ij}^T \Delta(K \mathbf{1}_t) - K)Y_{ij} = Y_{ij}^T (\Lambda - K)Y_{ij},$$

$$\text{vol}(\text{class}_j) = \sum_{i : Y_{ij} = 1} K_{il} = \sum_{i} K_{il}Y_{ij}^2 = \sum_{i} \left( \sum_{l} K_{il} \right) Y_{ij}^2$$

$$= Y_{ij}^T \Delta(K \mathbf{1}_t)Y_{ij} = Y_{ij}^T \Lambda Y_{ij},$$

Therefore, the problem becomes

$$\min_{Y \in \mathcal{Y}} \text{tr} \left( (Y^T \Lambda Y)^{-1} Y^T (\Lambda - K)Y \right)$$

$$\iff \min_{Y \in \mathcal{Y}} \text{tr} \left( I - Y(Y^T \Lambda Y)^{-1} Y^T K \right)$$

$$\iff \min_{Y \in \mathcal{Y}} \text{tr} \left( \Lambda^{-1}K - Y(Y^T \Lambda Y)^{-1} Y^T K \right)$$

$$\iff \min_{Y \in \mathcal{Y}} \text{tr} \left( \Lambda(\Lambda^{-1} - Y(Y^T \Lambda Y)^{-1} Y^T)K(\Lambda^{-1} - Y(Y^T \Lambda Y)^{-1} Y^T) \right)$$

$$\iff \min_{Y \in \mathcal{Y}, B} \text{tr} \left( \Lambda(\Lambda^{-1} - YB)K(\Lambda^{-1} - YB) \right),$$

which is equivalent to what we have seen before.