A NEW ALGORITHM FOR FINDING THE MINIMUM DISTANCE BETWEEN TWO CONVEX HULLS

Dougsoo Kaown, B.Sc., M.Sc.

Dissertation Prepared for the Degree of

DOCTOR OF PHILOSOPHY

UNIVERSITY OF NORTH TEXAS

May 2009

APPROVED:

Jianguo Liu, Major Professor
John Neuberger, Committee Member
Xiaohui Yuan, Committee Member
Matthew Douglass, Chair of the Department of Mathematics
Michael Monticino, Interim Dean of the Robert B. Toulouse School of Graduate Studies

Kaown, Dougsoo. A New Algorithm for Finding the Minimum Distance between Two Convex Hulls. Doctor of Philosophy (Mathematics), May 2009, 66 pp., 24 illustrations, bibliography, 27 titles.

The problem of computing the minimum distance between two convex hulls has applications to many areas including robotics, computer graphics and path planning. Moreover, determining the minimum distance between two convex hulls plays a significant role in support vector machines (SVM). In this study, a new algorithm for finding the minimum distance between two convex hulls is proposed and investigated. A convergence of the algorithm is proved and applicability of the algorithm to support vector machines is demostrated. The performance of the new algorithm is compared with the performance of one of the most popular algorithms, the sequential minimal optimization (SMO) method. The new algorithm is simple to understand, easy to implement, and can be more efficient than the SMO method for many SVM problems.

ACKNOWLEDGEMENTS

I thank my advisor Dr. Liu, Jianguo who gave me endless support and guidance. There are no words to describe his support and guidance. I thank my PhD committee members Dr. Neuberger and Dr. Yuan. I thank my father. Even though he already left our family, I always think about him. Without my father, my mother raised me with her heart with endless love. I thank my mother. She always supports me. I also thank my wife and my daughter who always love me and are with me. Without them, I could not finish this pleasant and wonderful journey. I thank the Department of Mathematics for giving me this opportunity and supporting me on this journey. I thank all staff and faculty members in the department. They are my life time teachers.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1. Introduction and Discussion of the Problem

Finding the minimum distance between two convex hulls is a popular research topic with important real-world applications. The fields of robotics, animation, computer graphics and path planning all use this distance calculation. For example, in robotics, the distance between two convex hulls is calculated to determine the path of the robot so that it can avoid collisions.

However, computing the minimum distance between two convex hulls plays its most significant role in support vector machines (SVM) [6], [7], [10], [15]. SVM is a very useful classification tool. SVM gets its popularity because of its impressive performance in many real world problems. Many fast iterative methods are introduced to solve the SVM problems. The popular sequential minimal optimization (SMO) method is one of them. There are also many algorithms for finding the minimum distance between two convex hulls, including the Gilbert algorithm and Mitchell-Dem'yanov-Malozemov (MDM) algorithm. These two algorithms are the most basic algorithms for solving the SVM problems geometrically. Furthermore there have been many published papers based on these two algorithms. Keerthi's paper [2] is an important example of this. In this work, he uses these two algorithms to produce a new hybrid algorithm, and he shows that the SVM problems can be transformed by computing the minimum distance between two convex hulls.

Definition 1.1

Let $\{x_1, ..., x_m\}$ be a set of vectors in R^n. Then the convex combination of $\{x_1, ..., x_m\}$ is given by

$$\sum_i \alpha_i x_i \text{ where } \sum_i \alpha_i = 1 \ \textit{and} \ \alpha_i \geq 0 \tag{1}$$

Definition 1.2

For a given set $X = \{x_1, ..., x_m\}$, coX is the convex hull of X. coX is the set of all convex combinations of elements of X.

$$coX = \{\sum_i \alpha_i x_i | \sum_i \alpha_i = 1, \alpha_i \geq 0\} \tag{2}$$

Gilbert's algorithm and the MDM algorithm are easy to understand and implement. However, these two algorithms are designed for finding the nearest point to the origin from the given convex hull. Let $X = \{x_1, ..., x_m\}$ and $Y = \{y_1, ..., y_s\}$ be finite point sets in R^n. U and V denote convex hulls that are generated by X and Y, respectively.

$$U = \{u = \sum_i \alpha_i x_i | \sum_i \alpha_i = 1, \alpha_i \geq 0\}$$

$$V = \{v = \sum_j \beta_j y_j | \sum_j \beta_j = 1, \beta_j \geq 0\}$$

Then the Gilbert and the MDM algorithms find the solution for the following problem :

$$\min \|u\|$$

$$\text{subject to } u = \sum_i \alpha_i x_i \ , \ \sum_{i=1} \alpha_i = 1, \ \alpha_i \geq 0$$

This problem is called the minimal norm problem (MNP). The solution for the above MNP is the nearest point to the origin from U. For $x, y \in R^n$, the *Euclidean* norm can be defined as follows. $\|x\| = \sqrt{(x, x)}$ where the inner product is defined by $(x, y) = x^t y = x_1 y_1 + ... + x_n y_n$. The minimum distance between two convex hulls can be found by solving an optimization problem. This problem is called the nearest point problem (NPP). A NPP problem can be seen in Figure 1.1. By using Gilbert's algorithm or the MDM algorithm, a user of these algorithms can find the minimum distance between two convex hulls by finding the nearest point to the origin from the convex hull that is generated by the difference of vectors.

$$\min \|u - v\| \quad \text{Nearest Point Problem (NPP)}$$

$$\text{subject to } \; u = \sum_{i=1}^{m} \alpha_i x_i, \quad \sum_{i=1}^{m} \alpha_i = 1, \quad \alpha_i \geq 0$$

$$v = \sum_{j=1}^{s} \beta_j y_j, \quad \sum_{j=1}^{s} \beta_j = 1, \quad \beta_j \geq 0$$

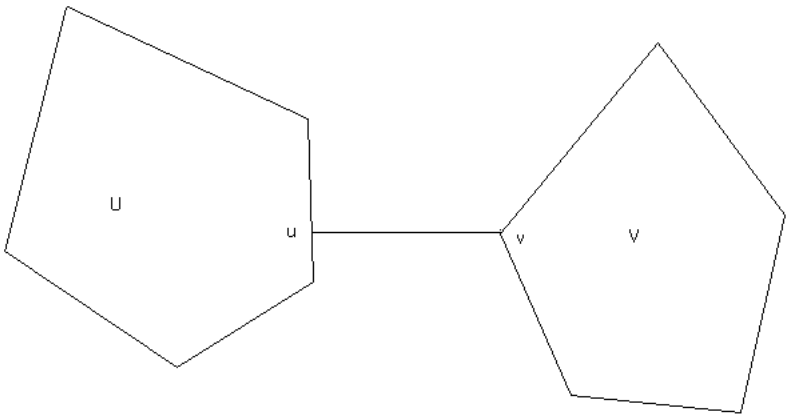

FIGURE 1.1. Nearest Point Problem

Set $Z = \{x_i - y_j | x_i \in X \text{ and } y_j \in Y\}$. Let $W = coZ$.

$$W = \{w = \sum_{l} \gamma_l z_l \mid \sum_{l} \gamma_l = 1, \ \gamma_l \geq 0\}$$

Then the minimum distance between two convex hulls can be found by solving the following MNP.

$$\min \|w\|$$

$$\text{subject to} \quad w = \sum_{l} \gamma_l z_l \quad \sum_{l=1} \gamma_l = 1 \quad , \gamma_l \geq 0$$

However, computing the minimum distance between two convex hulls by using such algorithms requires big memory storage, because the convex hull W has $m \times s$ vertices. For example, let $X = \{x_1, x_2\}$ and $Y = \{y_1, y_2, y_3\}$,

then $Z = X - Y = \{x_1 - y_1, x_1 - y_2, x_1 - y_3, x_2 - y_1, x_2 - y_2, x_2 - y_3\}$. This means that if one set has 1000 points and the other set has 1000 points, then 1,000,000 points will be used in the Gilbert or the MDM method. If these methods are used to find the minimum distance between two convex hulls, the calculation would be too expensive. Thus, a direct method without forming the difference of vectors is desired. This dissertation study presents a direct method of computing the minimum distance between two convex hulls. Chapter 2 proposes a new algorithm for finding the solution for NPP. Convergence is proved in chapter 2. Chapter 3 introduces Support Vector Machines (SVM). Chapter 4 discusses solving SVM by using the new algorithm. Chapter 5 shows the experiment results comparing SMO and the new algorithm. Finally, the conclusion and the future work for the new algorithm is discussed in chapter 6.

1.2. Previous Works

In this section, two important iterative algorithms for finding the nearest point to the origin from a given convex hull are discussed beginning with Gilbert's algorithm [5]. Gilbert's algorithm was the first algorithm developed for finding the minimum distance to the origin from a convex hull. Gilbert's algorithm is a geometric method used to find the minimum distance. Additionally, Gilbert's algorithm can be viewed geometrically. Gilbert's Algorithm can be explained by using convex hull representation.

First, let $X = \{x_1, ..., x_m\}$ and $U = \{u = \sum_i \alpha_i x_i | \sum_i \alpha_i = 1, \alpha_i \geq 0\}$.

Definition 1.3

For $u \in U$, define

$$\delta_{MDM}(u) = (u, u) - \min(x_i, u) \tag{3}$$

Theorem 1 in [1] states that $\delta_{MDM}(u) = 0$ if and only if u is the nearest to the origin.

Gilbert's algorithm

Step 1 Choose $u_k \in U$.

Step 2 Find $(\overline{x}_k, u_k) = \min(x_i, u_k)$.

Step 3 If $\delta_{MDM}(u_k) = 0$ then stop. Otherwise, find u_{k+1} where u_{k+1} is the point of the line segment joining u_k and $\overline{x}_k$ that has the minimum norm.

Step 4 Go to **Step 2**.

In Step 3, the minimum norm of line segment of u_k and $\overline{x_k}$ can be found in the following way. The minimum norm of line segment of u_k and $\overline{x_k}$ is

u_k if $(-u_k, \overline{x_k} - u_k) \leq 0$

$\overline{x_k}$ if $\|b - a\|^2 \leq (-u_k, \overline{x_k} - u_k)$

$u_k + \frac{(-u_k, \overline{x_k} - u_k)}{\|b-a\|^2}$ otherwise

Gilbert's Algorithm tends to converge fast at the beginning, but it is very slow as it approaches the solution, because Gilbert's Algorithm remains zigzagging until it finds the solution. Using the fast convergence at the beginning, Keerthi [2] used it when he made a hybrid algorithm. After Gilbert's algorithm, Mitchell-Dem'yanov-Malozemov (MDM) suggested a geometric algorithm to find the nearest point to the origin. Like Gilbert's algorithm, the MDM algorithm is an iterative method to find the nearest point from the origin. Let X and U be the same as the previous page. Let u^* be the nearest to the origin. Define $(x', u) = \max_{\alpha_i > 0}(x_i, u)$ where $\alpha_[i$ is the coefficient of x_i in the convex hull representation and $(\overline{x}, u) = \min(x_i, u)$.

Definition 1.4 Define

$$\Delta_{MDM}(u) = (x', u) - (\overline{x}, u) \tag{4}$$

Note that

$$(x', u) - (\overline{x}, u) = (x' - \overline{x}, u) \tag{5}$$

Lemma 2 in [1] states that $\alpha' \Delta_{MDM}(u) \leq \delta_{MDM}(u) \leq \Delta_{MDM}(u)$ where α' is the coefficient of x' in the convex hull representation.

Then by theorem 1 and lemma 2 in [1], u is the nearest point to the origin if and only if $\triangle_{MDM}(u) = 0$. Then the algorithm for finding u^* is the following.

The MDM Algorithm

Step 1 Choose $u_k \in U$

Step 2 Find x'_k and $\overline{x}_k$

Step 3 Set $u_k(t) = u_k + t\alpha'_k(\overline{x}_k - x'_k)$ where α' has the same index of x'_k and $t \in [0, 1]$

Step 4 Set $u_{k+1} = u_k + t_k\alpha'_k(\overline{x}_k - x'_k)$ where $(u_k(t_k), u_k(t_k)) = \min_{t\in[0,1]}(u_k(t), u_k(t))$

Step 5 If $\Delta_{MDM,k}(u_k) = (x'_k - \overline{x}_k, u_k) = 0$ then stop. Otherwise go to **Step 2**.

The convergence was proved in [1]. This algorithm is also very simple to implement. In this algorithm, u_k needs to stay in the convex hull. In order to be inside of the convex hull, the algorithm adds $t\alpha'_k\overline{x}$ and subtracts $t\alpha'_k x'_k$ to the current point so that the sum of α_i is 1. Here, it may take some time to comprehend why only a nonzero α is used while trying to find x' in the convex hull representation. The MDM algorithm has a reason to use x' when the current point u_k moves. If $\max(x_i, u)$ were used instead of $\max_{\alpha_i>0}(x_i, u)$, u_k may converge slowly or may not converge to u^*. $t_k\alpha'_k$ will decide how far the current point can move and $\overline{x}_k - x'_k$ will give the direction for u_k. Note that in each iteration, u_k updates two coefficients in the representation. By taking this method, four or more coefficients are updated at a time. The method will be shown in the Appendix A.

CHAPTER 2

A NEW ALGORITHM FOR FINDING THE MINIMUM DISTANCE BETWEEN TWO CONVEX HULLS

2.1. A New Algorithm for Finding the Minimum Distance of Two Convex Hulls

This chapter introduces a new algorithm for finding the minimum distance of two convex hulls and a proof for its convergence of the algorithm. This algorithm is an iterative method for computing the minimum distance between two convex hulls. Moreover, this algorithm has a good application to SVM.

Let X, Y, Z, U, V and W be the same as in chapter 1. As mentioned in Chapter 1, finding the nearest point to the origin of W is equivalent to solving NPP. Let w^* be the nearest point to the origin and $u^* - v^*$ be the solution for NPP, then $w^* = u^* - v^*$.

Note that for any $w \in W$,

$$w = u - v = \sum_i \alpha_i x_i - \sum_j \beta_j y_j$$

$$\delta_{MDM}(w) = (w, w) - \min_l (z_l, w) = (w, w) - \min_{i,j} (x_i - y_j, w)$$

$$\Delta_{MDM}(w) = \max_{\gamma_l > 0} (z_l, w) - \min_l (z_l, w) = \max_{\alpha_i > 0, \beta_j > 0} (x_i - y_j, w) - \min_{i,j} (x_i - y_j, w)$$

Mitchell-Dem'yanov-Malozemov defined $\Delta_{MDM}(w)$ and δ_{MDM} in their paper [1].

Definition 2.1

For $u \in U$ and $v \in V$, define

$$\Delta_x(u - v) = \max_{\alpha_i > 0} (x_i, u - v) - \min_i (x_i, u - v) \tag{6}$$

$$\Delta_y(v - u) = \max_{\beta_j > 0} (y_j, v - u) - \min_j (y_j, v - u) \tag{7}$$

Actually, $\Delta_{MDM}(u-v) = \Delta_x(u-v) + \Delta_y(v-u)$ in order to show this. Lemma 2.1 is needed.

Lemma 2.1

$$\max(x_i - y_j, u - v) = \max(x_i, u - v) - \min(y_j, u - v) \tag{8}$$

$$\min(x_i - y_j, u - v) = \min(x_i, u - v) - \max(y_j, u - v) \tag{9}$$

$$\max(y_j, u - v) = -\min(y_j, v - u) \tag{10}$$

(Proof) (8) can be shown by inequalities ($\leq$). Let $(x_M - y_M, u - v) = \max(x_i - y_j, u - v)$,

then $(x_M - y_M, u - v) = (x_M, u - v) - (y_M, u - v)$.

$(x_M, u - v) \leq \max(x_i, u - v)$ and $(y_M, u - v) \geq \min(y_j, u - v)$.

Thus $(x_M - y_M, u - v) = (x_M, u - v) - (y_M, u - v) \leq \max(x_i, u - v) - \min(y_j, u - v)$.

$\Longrightarrow \max(x_i - y_j, u - v) \leq \max(x_i, u - v) - \min(y_j, u - v)$.

($\geq$) Let $(x_m, u - v) = \max(x_i, u - v)$ and $(y_m, u - v) = \min(y_j, u - v)$,

then $(x_m, u - v) - (y_m, u - v) = (x_m - y_m, u - v)$.

$\max(x_i - y_j, u - v) \geq (x_m - y_m, u - v)$

$\Longrightarrow \max(x_i - y_j, u - v) \geq \max(x_i, u - v) - \min(y_j, u - v)$

Proof of (9) is similar to (8).

Proof of (10) can be shown by inequalities.■

Let $(y_M, u - v) = \max(y_j, u - v)$, then $(y_M, u - v) = -(y_M, v - u)$.

$\Longrightarrow \max(y_j, u - v) = -(y_M, v - u) \leq -\min(y_j, v - u)$

Let $(y_{min}, v - u) = \min(y_j, v - u)$, then $(y_{min}, v - u) = -(y_{min}, u - v)$.

$\Longrightarrow -\min(y_j, v - u) = (y_{min}, u - v) \leq \max(y_j, u - v)$.■

Lemma 2.2

$$\Delta_{MDM}(u - v) = \Delta_x(u - v) + \Delta_y(v - u) \tag{11}$$

(Proof) Since $u - v \in W$,

$$\begin{aligned}
\Delta_{MDM}(u - v) &= \max_{\alpha_i > 0, \beta_j > 0}(x_i - y_j, u - v) - \min_{i,j}(x_i - y_j, u - v) \\
&= \max_{\alpha_i > 0}(x_i, u - v) - \min_{\beta_j > 0}(y_j, u - v) - \min_{i}(x_i, u - v) + \max_{j}(y_j, u - v) \\
&= \max_{\alpha_i > 0}(x_i, u - v) - \min_{i}(x_i, u - v) - \min_{\beta_j > 0}(y_j, u - v) + \max_{j}(y_j, u - v)
\end{aligned}$$

$$= \max_{\alpha_i>0}(x_i, u-v) - \min_i(x_i, u-v) + \max_{\beta_j>0}(y_j, v-u) - \min_j(y_j, v-u)$$
$$= \Delta_x(u-v) + \Delta_y(v-u).\blacksquare$$

Using (8), (9) and (10), lemma 2.2 was shown. A new algorithm for finding the minimum distance between two convex hulls uses the iterative method. In the new algorithm, the following notations are used.

Let $(x'_k, u_k - v_k) = \max_{\alpha_{i_k}>0}(x_{i_k}, u_k - v_k)$ and $(\overline{x}_k, u_k - v_k) = \min_i(x_{i_k}, u_k - v_k)$.

$$\Delta_x(u_k - v_k) = (x'_k - \overline{x}_k, u_k - v_k) \tag{12}$$

$$\Delta_y(v_k - u_k) = (y'_k - \overline{y}_k, v_k - u_k) \tag{13}$$

Algorithm 2.1

Step 1 Choose $u_k \in U$ and $v_k \in V$

where $u_k = \sum_i \alpha_{i_k} x_i$ and $v_k = \sum_j \beta_{j_k} y_j$.

Step 2 Find $\overline{x}_{i_k}$ and x'_{i_k}

where $(\overline{x}_{i_k}, u_k - v_k) = \min_i(x_{i_k}, u_k - v_k)$ and $(x'_{i_k}, u_k - v_k) = \max_{\alpha_{i_k}>0}(x_{i_k}, u_k - v_k)$.

Step 3 $u_{k+1} = u_k + t_k \alpha_{i'_k}(\overline{x}_{i_k} - x_{i'_k})$

where $0 \le t_k \le 1$ and $t_k = \min\{1, \frac{\Delta_x(u_k - v_k)}{\alpha_{i'} \left\| \overline{x}_{i_k} - x'_{i_k} \right\|^2}\}$.

Note t_k is defined by

$$(u_k(t_k) - v_k, u_k(t_k) - v_k) = \min_{t\in[0,1]}(u_k(t) - v_k, u_k(t) - v_k)$$

where $u_k(t) = u_k + t\alpha_{i'_k}(\overline{x}_{i_k} - x'_{i_k})$ and

$$(u_k(t) - v_k, u_k(t) - v_k) = \alpha_{i'_k} \left\| \overline{x}_{i_k} - x'_{i_k} \right\|^2 t^2 - 2\alpha_{i'_k}\Delta_x(u_k - v_k)t + (u_k - v_k, u_k - v_k).$$

Step 4 Find $\overline{y}_{j_k}$ and y'_{j_k}

where $(\overline{y}_{j_k}, v_k - u_{k+1}) = \min_j(y_{j_k}, v_k - u_{k+1})$

and $(y'_{j_k}, v_k - u_{k+1}) = \max_{\beta_{j_k}>0}(y_{j_k}, v_k - u_{k+1})$.

Step 5 $v_{k+1} = v_k + t'_k \beta_{j'_k}(\overline{y}_{j_k} - y'_{j_k})$

where $0 \le t'_k \le 1$ and $t'_k = \min\{1, \frac{\Delta_y(v_k - u_{k+1})}{\beta_{j'_k} \left\| \overline{y}_{j_k} - y'_{j_k} \right\|^2}\}$.

Note t'_k is defined by

$$(v_k(t'_k) - u_{k+1}, v_k(t'_k) - u_{k+1}) = \min_{t' \in [0,1]} (v_k(t') - u_{k+1}, v_k(t') - u_{k+1})$$

where $v_k(t') = v_k + t'\beta_{j'_k}(\overline{y}_{j_k} - y'_{j_k})$ and

$$(v_k(t') - u_{k+1}, v_k(t') - u_{k+1}) = \beta_{j'_k} \left\| \overline{y}_{j_k} - y'_{j_k} \right\|^2 t'^2 - 2t'\beta_{j'_k} \Delta_y (v_k - u_{k+1}) + (v_k - u_{k+1}, v_k - u_{k+1}).$$

Step 6. Iterating this way, a sequence $\{u_k - v_k\}$ is obtained such that $\|u_{k+1} - v_{k+1}\| \leq \|u_k - v_k\|$ since $\|u_{k+1} - v_k\| \leq \|u_k - v_k\|$ and $\|u_{k+1} - v_{k+1}\| \leq \|u_{k+1} - v_k\|$.

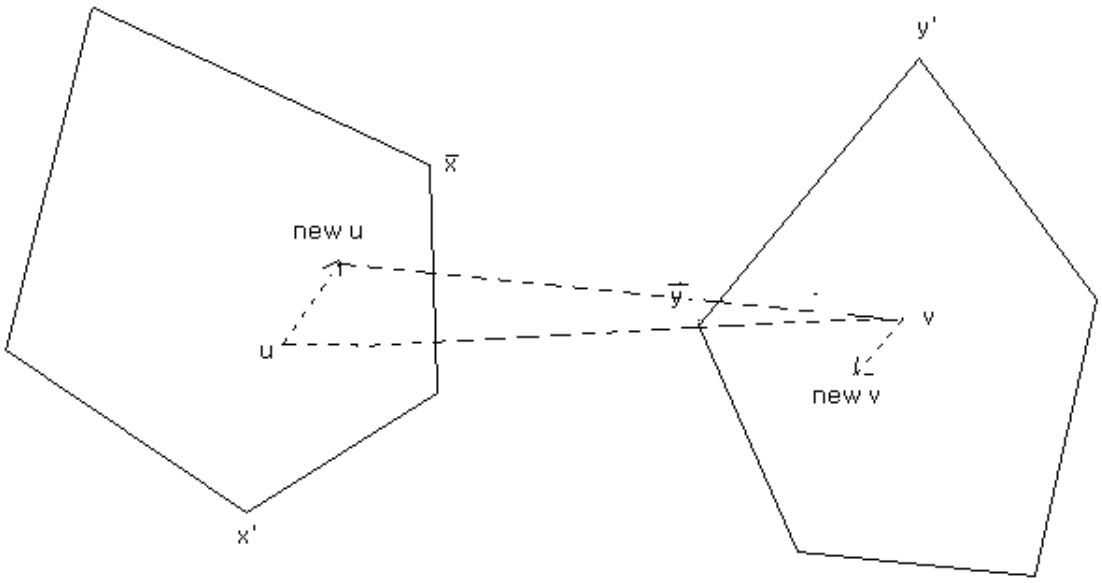

FIGURE 2.1. Algorithm 2.1

Figure 2.1 describes how to find *new u* and *new v*. First find *new u* by using *old u*, *old v*, then find *new v* by using *old v* and *new u*. Iterated this way, this algorithm finds the solution $u^* - v^*$ for NPP.

2.2. Convergence for Algorithm 2.1

Before proving the convergence for Algorithm 2.1, let's define the following.

Definition 2.2

For $u \in U$ and $v \in V$,

define

$$\delta_x(u-v) = (u, u-v) - \min_i(x_i, u-v) \tag{14}$$

$$\delta_y(v-u) = (v, v-u) - \min_j(y_j, v-u) \tag{15}$$

Lemma 2.3 Show that $\|u-v-(u^*-v^*)\| \leq \sqrt{\delta_x(u-v)+\delta_y(v-u)}$.

(Proof) By lemma 2 in [1] states that for any $w \in W, \|w-w^*\| \leq \sqrt{\delta_{MDM}(w)}$

Note that since $u-v \in W$,

$$\begin{aligned}\delta_{MDM}(u-v) &= (u-v, u-v) - \min_{i,j}(x_i - y_j, u-v)\\ &= (u, u-v) - (v, u-v) - \min_i(x_i, u-v) - \min_j(y_j, v-u)\\ &= (u, u-v) - \min_i(x_i, u-v) + (v, v-u) - \min_j(y_j, v-u)\\ &= \delta_x(u-v) + \delta_y(v-u).\end{aligned}$$

Then $\|u-v-(u^*-v^*)\| \leq \sqrt{\delta_{MDM}(u-v)} = \sqrt{\delta_x(u-v)+\delta_y(v-u)}$. ■

Lemma 2.4 Suppose $u_k \in U$, $v_k \in V, k = 1, 2...$ is a sequence of points such that $\|u_{k+1}-v_{k+1}\| \leq \|u_k - v_k\|$ and there is a subsequence such that $\delta_x(u_{k_j}-v_{k_j}) \to 0$ and $\delta_y(v_{k_j}-u_{k_j}) \to 0$ then $u_k - v_k \to u^* - v^* = w^*$.

(Proof) Since $\|u_k - v_k\|$ is a nonincreasing sequence bounded below, it has a limit

$$\|u_k - v_k\| \to \mu.$$

By lemma 2.3, $\left\|u_{k_j} - v_{k_j} - (u^*-v^*)\right\| \to 0$ since $\delta_x(u_{k_j}-v_{k_j}) \to 0$ and $\delta_y(v_{k_j}-u_{k_j}) \to 0$.

This implies $\left\|u_{k_j} - v_{k_j}\right\| \to \|u^*-v^*\| = \|w^*\|$.

Hence $\|u_k - v_k\| \to \|u^*-v^*\| = \|w^*\|$.

Any convergent subsequence of $u_k - v_k$ must converge to $w^* = u^* - v^*$ because of uniqueness of $w^* = u^* - v^*$. ■

Lemma 2.3 and 2.4 are important lemmas to prove the convergence of Algorithm 2.1. After the following theorem, support vectors will be defined. Support vectors are significant concept in this dissertation and support vector machines (SVM).

Theorem 1 Show

$$\min_i(x_i, u^* - v^*) = (u^*, u^* - v^*) \tag{16}$$

$$\min_j(y_j, v^* - u^*) = (v^*, v^* - u^*) \tag{17}$$

(proof) Let $u^* = \sum_i \alpha_i^* x_i^*$ and $v^* = \sum_j \beta_j^* y_j^*$.

Note $\min_i(x_i - v^*, u^* - v^*) \geq (u^* - v^*, u^* - v^*)$

and $\min_j(y_j - u^*, v^* - u^*) \geq (v^* - u^*, v^* - u^*)$.

Then $\min_i(x_i, u^* - v^*) - (v^*, u^* - v^*) \geq (u^*, u^* - v^*) - (v^*, u^* - v^*)$.

$\Longrightarrow \quad \min_i(x_i, u^* - v^*) \geq (u^*, u^* - v^*) = (\sum_i \alpha_i^* x_i^*, u^* - v^*)$

but $\min_i(x_i, u^* - v^*) \leq \sum_i \alpha_i^*(x_i^*, u^* - v^*)$.

Thus $\min_i(x_i, u^* - v^*) = (u^*, u^* - v^*)$.

Also $\min_j(y_j, v^* - u^*) - (u^*, v^* - u^*) \geq (v^*, v^* - u^*) - (u^*, v^* - u^*)$.

$\Longrightarrow \quad \min_j(y_j, v^* - u^*) \geq (v^*, v^* - u^*) = (\sum_j \beta_j^* y_j^*, v^* - u^*)$

but $\min_j(y_j, v^* - u^*) \leq \sum_j \beta_j^*(y_j^*, v^* - u^*)$.

Thus $\min_j(y_j, v^* - u^*) = (v^*, v^* - u^*)$. ■

Definition 2.3

Let the solution of NPP be $u^* - v^*$ then those x_i^*'s corresponding to $\alpha_i^* > 0$ and y_j^*'s corresponding to $\beta_j^* > 0$ are called support vectors.

As seen in Figure 2.1, u^* and v^* can be represented by support vectors x_1, x_2, y_1 and y_6. Subtract $(v^*, u^* - v^*)$ on both sides from (16), then

$$\min_i(x_i - v^*, u^* - v^*) = (u^* - v^*, u^* - v^*).$$

Similarly from (17),

$$\min_j(y_j - u^*, v^* - u^*) = (v^* - u^*, v^* - u^*).$$

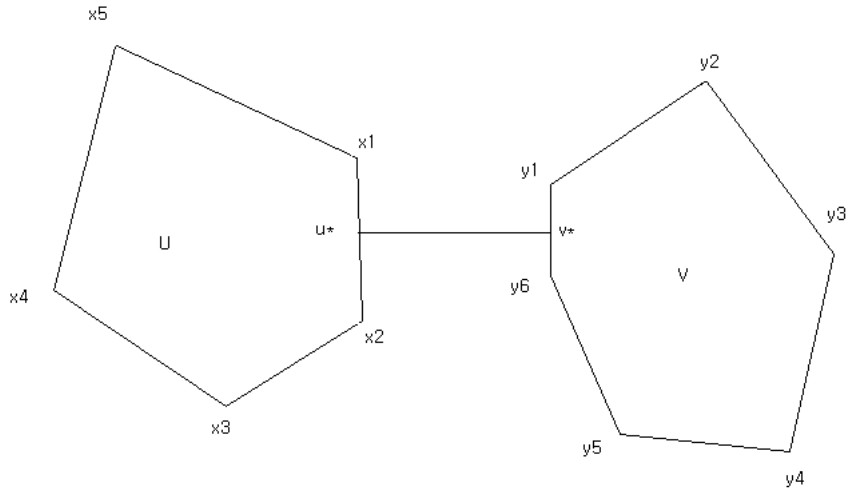

FIGURE 2.2. Support Vectors

Lemma 2.5 For any vector $u - v$, $u \in U$, $v \in V$

$$\alpha_{i'}\Delta_x(u-v) \le \delta_x(u-v) \le \Delta_x(u-v) \tag{18}$$

where $\alpha_{i'} > 0$ and $(x_{i'}, u-v) = \max_{\alpha_i>0}(x_i, u-v)$.

$$\beta_{j'}\Delta_y(v-u) \le \delta_y(v-u) \le \Delta_y(v-u) \tag{19}$$

where $\beta_{j'} > 0$ and $(y_{j'}, v-v) = \max_{\beta_j>0}(y_j, v-u)$.

(Proof of (18)) Let $u = \sum_{i=1}^{m} \alpha_i x_i$, then

$$(u-v, u-v) = \sum_{i=1}^{m} \alpha_i (x_i - v, u-v) \le \max_{\alpha_i>0}(x_i - v, u-v).$$

$$\Rightarrow \delta_x(u-v) \le \Delta_x(u-v)$$

Let $(x_{i''}, u-v) = \min_i (x_i, u-v)$ and $(x_{i'}, u-v) = \max_{\alpha_i > 0} (x_i, u-v)$

Then $\Delta_x(u-v) = (x_{i'} - x_{i''}, u-v)$. Set $\overline{A} = (\overline{\alpha}_1, ..., \overline{\alpha}_m)$ where

$\overline{\alpha}_i = \alpha_i$ if $i \neq i', i''$

$\overline{\alpha}_i = 0$ if $i = i'$

$\overline{\alpha}_i = \alpha_{i'} + \alpha_{i''}$ if $i = i''$.

$$\overline{u} = u + \alpha_{i'}(x_{i''} - x_{i'})$$

$$(\overline{u} - v, u - v) = (u + \alpha_{i'}(x_{i''} - x_{i'}) - v, u - v)$$

$$= (u, u-v) + \alpha_{i'}((x_{i''} - x_{i'}) - v, u - v)$$

$$= (u,u) - (u,v) + \alpha_{i'}((x_{i''} - x_{i'}), u - v) - (v, u - v)$$

$$= (u,u) - (u,v) - \alpha_{i'}\Delta_x(u-v) - (v,u) + (v,v)$$

$$= (u - v, u - v) - \alpha_{i'}\Delta_x(u-v).$$

so

$$\delta_x(u-v) = (u-v, u-v) - \min_i (x_i - v, u - v) \geq (u-v, u-v) - (\overline{u} - v, u - v).$$

But the right hand side of inequity is equal to

$$(u-v, u-v) - (u-v, u-v) + \alpha_{i'}\Delta_x(u-v).$$

$$\Rightarrow \delta_x(u-v) \geq \alpha_{i'}\Delta_x(u-v)$$

Proof of (19) is similar to the proof of (18). ■

Lemma 2.6

(20) $$\lim_{k\to\infty} \alpha'_k \Delta_x(u_k - v_k) = 0$$

(21) $$\lim_{k\to\infty} \beta'_k \Delta_y(v_k - u_k) = 0$$

(Proof of (20)) First, observe that $u_k(t) = u_k + t\alpha'_k(\overline{x}_k - x'_k)$, $v_k(t') = v_k + t'\beta'_k(\overline{y}_k - y'_k)$ and $\|u_{k+1} - v_{k+1}\| \leq \|u_k - v_k\|$.

Note that

$$(u_k(t) - v_k, u_k(t) - v_k) = (u_k - v_k, u_k - v_k) - 2t\alpha'_{i'}\Delta_x(u_k - v_k) + t^2(\alpha'_{i'}\|\overline{x}_k - x'_k\|).$$

Suppose the assertion false. Then there is a subsequence $u_{k_j} - v_{k_j}$ for which $\alpha'_{k_j}\Delta_x(u_{k_j} - v_{k_j}) \geq \epsilon > 0$. Then $(u_{k_j}(t) - v_{k_j}, u_{k_j}(t) - v_{k_j}) \leq (u_{k_j} - v_{k_j}, u_{k_j} - v_{k_j}) - 2t\epsilon + t^2d^2$ where $d = \max_{l,p}\|x_l - x_p\|$. Because right hand side of inequality is minimized at $t_\circ = \min\{\frac{\varepsilon}{d^2}, 1\}$

(2.6.1) If $t_\circ = \frac{\varepsilon}{d^2}$,

$$\begin{aligned}(u_{k_j}(t) - v_{k_j}, u_{k_j}(t) - v_{k_j}) &\leq (u_{k_j} - v_{k_j}, u_{k_j} - v_{k_j}) - 2\tfrac{\varepsilon}{d^2}\varepsilon + (\tfrac{\varepsilon}{d^2})^2d^2 \\ &= (u_{k_j} - v_{k_j}, u_{k_j} - v_{k_j}) - \tfrac{\varepsilon^2}{d^2} \\ &= (u_{k_j} - v_{k_j}, u_{k_j} - v_{k_j}) - t_\circ\varepsilon.\end{aligned}$$

(2.6.2) If $t_\circ = 1 (\Rightarrow \frac{\varepsilon}{d^2} > 1 \Rightarrow \varepsilon > t_\circ d^2)$

$$\begin{aligned}(u_{k_j}(t) - v_{k_j}, u_{k_j}(t) - v_{k_j}) &\leq (u_{k_j} - v_{k_j}, u_{k_j} - v_{k_j}) - 2t_\circ\varepsilon + t_\circ^2d^2 \\ &\leq (u_{k_j} - v_{k_j}, u_{k_j} - v_{k_j}) - 2t_\circ\varepsilon + \varepsilon t_\circ \\ &= (u_{k_j} - v_{k_j}, u_{k_j} - v_{k_j}) - t_\circ\varepsilon.\end{aligned}$$

By (2.6.1) and(2.6.2)

$$(u_{k_j}(t) - v_{k_j}, u_{k_j}(t) - v_{k_j}) \leq (u_{k_j} - v_{k_j}, u_{k_j} - v_{k_j}) - t_\circ\varepsilon.$$

This contradicts $\|u_{k+1} - v_{k+1}\| \leq \|u_k - v_k\|$.

Proof of (21) is similar to (20). ■

Lemma 2.7

$$\underline{\lim}\Delta_x(u_k - v_k) = 0 \tag{22}$$

$$\underline{\lim}\Delta_y(v_k - u_k) = 0 \tag{23}$$

(proof) Suppose $\underline{\lim}\Delta_x(u_k - v_k) \neq 0$ and $\underline{\lim}\Delta_y(v_k - u_k) \neq 0$

then there is $\Delta'_x(u' - v')$ such that $\underline{\lim}\Delta_x(u_k - v_k) = \Delta'_x(u' - v')$.

Then for sufficiently large $k > K_1$

$$\Delta_x(u_k - v_k) \geq \frac{\Delta'_x(u' - v')}{2}. \tag{24}$$

And there is $\Delta''_y(v'' - u'')$ such that $\underline{\lim}\Delta_y(v_k - u_k) = \Delta'_y(v'' - u'')$. Then for sufficiently large $k > K_2$

$$\Delta_y(v_k - u_k) \geq \frac{\Delta''_y(v'' - u'')}{2}. \tag{25}$$

By lemma 2.6, $\alpha'_k \to 0$

Let $\overline{t_k}$ be a point that $(u_k(t) - v_k, u_k(t) - v_k)$ assumes its global minimum.

Then

$$\begin{aligned}
&(u_k(t) - v_k, u_k(t) - v_k) \\
&= (u_k + t\alpha_{i'}(\overline{x}_k - x'_k) - v_k, u_k + t\alpha_{i'}(\overline{x}_k - x'_k) - v_k) \\
&= (u_k - v_k + t\alpha_{i'}(\overline{x}_k - x'_k), u_k - v_k + t\alpha_{i'}(\overline{x}_k - x'_k)) \\
&= (u_k - v_k, u_k - v_k) - 2t\alpha_{i'}(u_k - v_k, (x'_k - \overline{x}_k)) + t^2(\alpha_{i'}\|\overline{x}_k - x'_k\|)^2 \\
&= (u_k - v_k, u_k - v_k) - 2t\alpha_{i'}\Delta_x(u_k - v_k) + t^2(\alpha_{i'}\|\overline{x}_k - x'_k\|)^2.
\end{aligned}$$

$$\Rightarrow \overline{t_k} = \frac{\Delta_x(u_k - v_k)}{\alpha_{i'}\|\overline{x}_k - x'_k\|^2} \tag{26}$$

By $\Delta_x(u_k - v_k) \geq \frac{\Delta'_x(u'-v')}{2}$ and $\alpha'_k \to 0$, $\overline{t_k} \to \infty$.

Again by lemma 2.6 $\beta'_k \to 0$ and let $\overline{t'_k}$ be a point that that $(v_k(t') - u_k, v_k(t') - u_k)$ assumes its global minimum. Then

$$\overline{t'_k} = \frac{\Delta_y(v_k - u_k)}{\beta_{i'}\|\overline{y_k} - y'_k\|} \tag{27}$$

Again by $\Delta_y(v_k - u_k) \geq \frac{\Delta''_y(v''-u'')}{2}$ and $\beta'_k \to 0$, $\overline{t'_k} \to \infty$.

Hence, for large $k > K > K_1$, the minimum of $(u_k(t) - v_k, u_k(t) - v_k)$ on the segment $0 \leq t \leq 1$ is obtained at $t_k = 1$ so that for such values k

$$u_{k+1} = u_k + \alpha'_k(\overline{x}_k - x'_k). \tag{28}$$

Also, for large $k > K > K_2$, the minimum of $(v_k(t') - u_k, v_k(t') - u_k)$ on the segment $0 \leq t' \leq 1$ obtained at $t'_k = 1$ so that for such values k

$$v_{k+1} = v_k + \beta'_k(\overline{y}_k - y'_k). \tag{29}$$

Let $u_{k_j} - v_{k_j}$ be a subsequence such that

$$\Delta_x(u_{k_j} - v_{k_j}) \to \Delta'_x(u' - v'). \tag{30}$$

By discarding terms from the sequence $u_{k_j} - v_{k_j}$, the following conditions are satisfied.

$$\alpha_i^{k_j} \to \alpha_i^* \text{ and } \beta_i^{k_j} \to \beta_i^* \tag{31}$$

where α_i^* and β_i^* are coefficient for u' and v' respectively.

$$\overline{x}_{k_j} = \overline{x} \in X \tag{32}$$

This means that $\min(x_i, u_{k_j} - v_{k_j})$ is obtained for all k_j at the same value of i

$$(\overline{x}, u_{k_j} - v_{k_j}) = \min_i(x_i, u_{k_j} - v_{k_j})$$

$$x'_{k_j} = x' \in X \tag{33}$$

$$(x', u_{k_j} - v_{k_j}) = \max_{\alpha_i > 0}(x_i, u_{k_j} - v_{k_j}) \tag{34}$$

Using (30), (31), (32) and (33), the following is obtained $(x' - \overline{x}, u' - v') = \Delta'_x(u' - v')$

Now introduce the new notations,

$$\rho_x^* = \min(x_i, u' - v') \tag{35}$$

$$X_1^* = \{x_i \in X \mid (x_i, u' - v) = \rho_x^*\} \tag{36}$$

$$X_2^* = X \setminus X_1^* \tag{37}$$

Note that

$$x_i \in X_2^* \iff (x_i, u' - v') > \rho_x^* \tag{38}$$

(39) $$\overline{x} \in X_1^*$$

and

$$x' \in X_2^*$$

Set

(40) $$\rho'_x = \min_{x_i \in X_2^*} (x_i, u' - v')$$

and

(41) $$\rho'_x > \rho^*_x$$

then

(42) $$\tau_x = \min\{\Delta'_x(u' - v'), \rho'_x - \rho^*_x\}.$$

Note that for $x_i \in X_2^*$, $(x_i, u' - v') \geq \rho^*_x + \tau_x$.

Choose δ_{o_x} such that whenever $\|(u - v) - (u' - v')\| < \delta_{o_x}$,

(43) $$\max_i |(x_i, u - v) - (x_i, u' - v')| < \frac{\tau_x}{4}$$

(**Claim**) Let k_j be such that $\|u_{k_j} - v_{k_j} - (u' - v')\| < \delta_{o_x}$. Then $X_1(u_{k_j} - v_{k_j}) \subset X_1^*$

where $X_1(u_{k_j} - v_{k_j}) = \{x_i \in X \mid (x_i, u_{k_j} - v_{k_j}) = \min(x_i, u_{k_j} - v_{k_j})\}$.

(proof of claim) Let $x_i \in X_1(u_{k_j} - v_{k_j})$, then

$$(x_i, u' - v') = (x_i, u_{k_j} - v_{k_j}) + (x_i, (u' - v') - (u_{k_j} - v_{k_j}))$$

$$= \min_i (x_i, u_{k_j} - v_{k_j}) + (x_i, (u' - v') - (u_{k_j} - v_{k_j}))$$

$$= \min_i (x_i, u_{k_j} - v_{k_j}) + (x_i, u' - v') - (x_i, (u_{k_j} - v_{k_j}))$$

$$= \rho^*_x - \rho^*_x + \min_i (x_i, u_{k_j} - v_{k_j}) + (x_i, u' - v') - (x_i, u_{k_j} - v_{k_j})$$

$$= \rho^*_x + (x_i, (u' - v') - (u_{k_j} - v_{k_j})) + [-\min_i (x_i, u' - v') + \min_i (x_i, u_{k_j} - v_{k_j})]$$

$$\leq \rho^*_x + (x_i, (u^* - v^*) - (u_{k_j} - v_{k_j})) + \max_i |(x_i, u_{k_j} - v_{k_j}) - (x_i, u' - v')|$$

$$\leq \rho^*_x + \tau_x/4 + \tau_x/4 = \rho^*_x + \tau_x/2.$$

Since δ_{O_x} was chosen arbitrarily and $(x_i, u' - v') \leq \rho_x^* + \tau_x/2$.

$$\Rightarrow x_i \in X_1^*$$

(end of claim)

Let $k_j > K$ be such that the following conditions hold.

$$\Delta_x(u_k - v_k) \geq \Delta'_x(u' - v') - \tau_x/4 \tag{44}$$

$$\|u_{k_j} - v_{k_j} - (u' - v')\| < \delta_{o_x}/2 \tag{45}$$

$$\sum_{l=1}^{p} \alpha'_{k_j+l-1} < \delta_{o_x}/4d \quad \text{since} \quad \alpha'_k \to 0 \tag{46}$$

where $\quad d_x = \max_{\forall l,r} \|x_l - x_r\| > 0$, $d_y = \max_{\forall l,r} \|y_l - y_r\|$ and $d = \{d_x, d_y\}$

$$\sum_{l=1}^{p} \beta'_{k_j+l-1} < \delta_{o_x}/4d \quad \text{since} \quad \beta'_k \to 0 \tag{47}$$

By (28) and (46),

$$u_{k_j+p} = u_{k_j} + \sum_{l=1}^{p} \alpha'_{k_j+l-1}(\overline{x}_{k_j+l-1} - x'_{k_j+l-1}) \tag{48}$$

$$\|u_{k_j+p} - u_{k_j}\| < \frac{\delta_{o_x}}{4d} d = \delta_{o_x}/4 \tag{49}$$

By (29) and (47),

$$v_{k_j+p} = v_{k_j} + \sum_{l=1}^{p} \beta'_{k_j+l-1}(\overline{y}_{k_j+l-1} - y'_{k_j+l-1})$$

$$\|v_{k_j+p} - v_{k_j}\| < \frac{\delta_{o_x}}{4d} d = \delta_{o_x}/4.$$

Then by (46)

$$\|u_{k_j+p} - v_{k_j+p} - (u' - v')\| \leq \|u_{k_j+p} - v_{k_j+p} - (u_{k_j} - v_{k_j})\| + \|u_{k_j} - v_{k_j} - (u' - v')\| < \delta_{o_x}$$

Hence $X_1(u_{k_j+p} - v_{k_j+p}) \subset X_1^*$.

Next for all $x_i \in X_1^*$ and $p \in [1:m]$, the followings are obtained

$$(x_i, u_{k_j+p} - v_{k_j+p}) = (x_i, u' - v') + (x_i, u_{k_j+p} - v_{k_j+p} - (u' - v'))$$

$$= (x_i, u' - v') + (x_i, u_{k_j+p} - v_{k_j+p}) - (x_i, u' - v')$$

$$\leq \rho_x^* + \tau_x/4 \leq \Delta_x'(u' - v') - \tau_x + \rho_x^* + \tau/4 = \Delta_x'(u' - v') + \rho_x^* - \frac{3\tau_x}{4}$$

$$(x_i, u_{k_j+p} - v_{k_j+p}) \leq \Delta_x'(u' - v') + \rho_x^* - \frac{3\tau_x}{4}. \tag{50}$$

Now show that for some vector $u_{k_j+p} - v_{k_j+p}$, $p \in [1:m]$

$$\max_{\{i|\alpha_i^{k_j+p}>0\}} (x_i, u_{k_j+p} - v_{k_j+p}) \leq \Delta_x'(u' - v') + \rho_x^* - \frac{3\tau_x}{4}.$$

If this inequality fails to hold for some p, then $x'_{k_j+p} \in X_2^*$.

Since $X_1(u_{k_j+p} - v_{k_j+p}) \subset X_1^*$, $u_{k_j+p+1} = u_{k_j+p} + \alpha'_{k_j+p}(\overline{x}_{k_j+p} - x'_{k_j+p})$ involves the vector x'_{k_j+P} with zero coefficients, while the newly introduced vector $\overline{x}_{k_j+p}$ satisfies (50). Since X_2^* contains at most m-1 vectors, there is $p \in [1:m]$ such that all vectors appearing in the representation hold (50).

$$\Rightarrow \max_{\alpha_i>0}(x_i, u_{k_j+p} - v_{k_j+p}) \leq \Delta_x'(u' - v') + \rho_x^* - \frac{3\tau_x}{4}$$

On the other hand,

$$\min(x_i, u_{k_j+p} - v_{k_j+p}) \geq \rho_x^* - \tau_x/4. \tag{51}$$

Thus

$$\Delta_x(u_{k_j+p} - v_{k_j+p}) = \max_{\alpha_i>0}(x_i, u_{k_j+p} - v_{k_j+p}) - \min(x_i, u_{k_j+p} - v_{k_j+p}) \leq \Delta_x'(u' - v') - \tau_x/2.$$

This contradicts (44), Thus $\underline{\lim}\Delta_x(u_k - v_k) = 0$.

Similarly, $\underline{\lim}\Delta_y(v_k - u_k) = 0$ can be shown. ■

Theorem 2 $\Delta_x(u - v) + \Delta_y(v - u) = 0$ if and only if $u - v = u^* - v^*$.

(proof) By lemma 2.2, $\Delta_x(u-v)+\Delta_y(v-u)=\Delta_{MDM}(u-v)$. And by theorem 1 and lemma 2 in [1], $\Delta_{MDM}(u-v)=0$ if and only if $u-v=u^*-v^*$.

Thus, theorem has been proved.■

Theorem 3. The sequence u_k-v_k converges to u^*-v^*.

(proof) By Lemma 2.7, there are subsequence $u_{k_j}-v_{k_j}$ and $v_{k_j}-u_{k_j}$ such that $\Delta_x(u_{k_j}-v_{k_j})\to 0$ and $\Delta_y(v_{k_j}-u_{k_j})\to 0$. Then by Lemma 2.3, 2.4 and 2.5 u_k-v_k converges to u^*-v^*. ■

Theorem 4. Let

$$G=\{u=\sum_i \alpha_i x_i \mid (u,u^*-v^*)=(u^*,u^*-v^*)\} \tag{52}$$

$$G'=\{v=\sum_j \beta_j x_j \mid (v,v^*-u^*)=(v^*,v^*-u^*)\} \tag{53}$$

Then for large k, $u_k\in G$ and $v_k\in G'$

(proof) By theorem 1, $\min_i(x_i,u^*-v^*)=(u^*,u^*-v^*)$

and $\min_j(y_j,v^*-u^*)=(v^*,v^*-u^*)$.

Set

$$X_1=\{x_i\in X \mid (x_i,u^*-v^*)=(u^*,u^*-v^*)\},$$

$$X_2=X\setminus X_1=\{x_i\in X \mid (x_i,u^*-v^*)>(u^*,u^*-v^*)\},$$

$$Y_1=\{y_j\in Y \mid (y_j,v^*-u^*)=(v^*,v^*-u^*)\}$$

and

$$Y_2=\{y_j\in Y \mid (y_j,v^*-u^*)>(v^*,v^*-u^*)\}.$$

If $X_2=\varnothing$ then $u_k\in G$, $^\forall k$.

If $Y_2=\varnothing$ then $v_k\in G'$, $^\forall k$.

Now suppose $X_2\neq\varnothing, Y_2\neq\varnothing$

Let

$$\tau=\min_{x_i\in X_2}(x_i,u^*-v^*)-(u^*,u^*-v^*)>0 \tag{54}$$

and

$$\tau' = \min_{y_j \in Y_2}(y_j, v^* - u^*) - (v^*, v^* - u^*) > 0 \tag{55}$$

Note that $(x_i - v_k, u_k - v_k) \to (x_i - v_k, u^* - v^*)$ and $(y_j - u_k, v_k - u_k) \to (y_j - u_k, v^* - u^*)$ since $u_k - v_k \to v^* - u^*$, this follows that for large $k > K_1$

$$\max_i |(x_i - v_k, u_k - v_k) - (x_i - v_k, u^* - v^*)| < \frac{\tau}{4} \tag{56}$$

$$\max_j |(y_j - u_k, v_k - u_k) - (y_j - u_k, v^* - u^*)| < \frac{\tau'}{4} \tag{57}$$

Then by the claim in lemma 2.7 for large $k > K_1$

$X_1(u_k - v_k) \subset X_1$,

where $X_1(u_k - v_k) = \{x_i \in X \mid (x_i, u_k - v_k) = \min_i(x_i, u_k - v_k)\}$

and $Y_1(v_k - u_k) \subset Y_1$,

where $Y_1(v_k - u_k) = \{y_j \in Y \mid (y_j, v_k - u_k) = \min_j(x_j, v_k - u_k)\}$.

Now let $x \in X_1$ then

$$(x, u^* - v^*) = (u^*, u^* - v^*) \tag{58}$$

By (56), $\max_i |(x_i, u_k - v_k) - (x_i, u^* - v^*)| < \frac{\tau}{4}$. Then

$$|(x, u_k - v_k) - (x, u^* - v^*)| < \frac{\tau}{4} \tag{59}$$

By (58), $(x, u_k - v_k) - (u^*, u^* - v^*) < \frac{\tau}{4}$. Then

$$(x_i, u_k - v_k) < (u^*, u^* - v^*) + \frac{\tau}{4} \tag{60}$$

Now let $x \in X_2$, then by (54)

$$(x, u^* - v^*) - (u^*, u^* - v^*) \geq \tau \tag{61}$$

and by (56)

$$-\frac{\tau}{4} < (x, u_k - v_k) - (x, u^* - v^*) < \frac{\tau}{4} \tag{62}$$

Add $(x, u_k - v_k) - (x, u^* - v^*)$ to (61) on both sides,

Then

$(x, u_k - v_k) - (x, u^* - v^*) + (x, u^* - v^*) - (u^*, u^* - v^*) \geq \tau + (x, u_k - v_k) - (x, u^* - v^*)$.

And by (62),

$$(x, u_k - v_k) \geq (u^*, u^* - v^*) + \frac{3\tau}{4}. \tag{63}$$

Similarly,

$$(y_j, v_k - u_k) \leq (v^*, v^* - u^*) + \frac{\tau'}{4} \text{ if } y_j \in Y_1 \tag{64}$$

and

$$(y_j, v_k - u_k) \geq (v^*, v^* - u^*) + \frac{3\tau'}{4} \text{ if } y_j \in Y_2. \tag{65}$$

are obtained.

For u_k, $k > K_1$, if the convex hull representation for u_k involves $x_i \in X_2$ with a nonzero coefficient, then $\Delta_x(u_k - v_k) \geq \frac{\tau}{2}$ by (60), (63) and $X_1(u_k - v_k) \subset X_1$.

Similarly, for $v_{k'}$, $k' > K_1'$, involves $y_j \in Y_2$ with a nonzero coefficient, then

$$\Delta_y(u_{k'} - v_{k'}) \geq \frac{\tau'}{2}. \tag{66}$$

Then for large $k > K_1$ and $\dot{k} > K_1'$, u_k and v_k can be written as below

$$u_k = \sum_{\{i|x_i \in X_1\}} \alpha_i^{(k)} x_i + \sum_{\{i|x_i \in X_2\}} \alpha_i^{(k)} x_i \tag{67}$$

$$v_k = \sum_{\{j|y_j \in Y_1\}} \beta_j^{(k)} y_j + \sum_{\{j|y_j \in Y_2\}} \beta_j^{(k)} y_j \tag{68}$$

Consider

$$\begin{aligned}
&(u_k - u^*, u^* - v^*) \\
&= (u_k, u^* - v^*) - (u^*, u^* - v^*) \\
&= (\sum_{\{i|x_i \in X_1\}} \alpha_i^{(k)} x_i + \sum_{\{i|x_i \in X_2\}} \alpha_i^{(k)} x_i, u^* - v^*) - (u^*, u^* - v^*) \\
&= (\sum_{\{i|x_i \in X_1\}} \alpha_i^{(k)} x_i, u^* - v^*) + (\sum_{\{i|x_i \in X_2\}} \alpha_i^{(k)} x_i, u^* - v^*) - (u^*, u^* - v^*) \\
&= \sum_{\{i|x_i \in X_1\}} \alpha_i^{(k)} (x_i, u^* - v^*) + \sum_{\{i|x_i \in X_2\}} \alpha_i^{(k)} (x_i, u^* - v^*) \\
&- \sum_{\{i|x_i \in X_1\}} \alpha_i^{(k)} (u^*, u^* - v^*) - \sum_{\{i|x_i \in X_2\}} \alpha_i^{(k)} (u^*, u^* - v^*) \\
&= \sum_{\{i|x_i \in X_2\}} \alpha_i^{(k)} (x_i - u^*, u^* - v^*) \geq \tau \sum_{\{i|x_i \in X_2\}} \alpha_i^{(k)}
\end{aligned}$$

since $(x_i, u^* - v^*) = (u^*, u^* - v^*)$ if $x_i \in X_1$ and $(x_i - u^*, u^* - v^*) \geq \tau$ if $x_i \in X_2$.

By the convergence of $u_k - v_k$, the left hand side of this inequality tends to zero as $k \to \infty$

This means that $\sum_{\{i|x_i \in X_2\}} \alpha_i^{(k)} \to 0$.

Similarly,

$(v - u^*, v^* - u^*)$

$= \sum_{\{j|y_j \in Y_2\}} \beta_j^{(k)} (y_j - u^*, v^* - u^*) \geq \tau' \sum_{\{j|y_j \in Y_2\}} \beta_j^{(k)}$

Also, $\sum_{\{j|y_j \in Y_2\}} \beta_j^{(k)} \to 0$ can be seen.

Choose $K > K_1$ so large and for $k \geq K$,

$$\sum_{\{i_i \in X_2\}} \alpha_i^{(k)} \leq \frac{\tau}{2d^2} \tag{69}$$

$$\text{where } d = \max_{x_i \in X_1, x_j \in X_2} \|x_i - x_j\| > 0.$$

Let $\overline{t_k}$ be a point that $(u_k(t) - v_k, u_k(t) - v_k)$ assumes a global minimum.

If the representation of u_k, $k \geq K$, involves $x_i \in X_2$ with a nonzero coefficient then

$\overline{t_k} = \frac{\Delta_x(u_k - v_k)}{\alpha'_k \|\overline{x_k} - x\prime_k\|^2} \geq \frac{\tau}{2\ (\sum_{\{i|x_i \in X_2\}} \alpha_i^{(k)}) d^2} \geq 1$

by $\Delta_x(u_k - v_k) \geq \frac{\tau}{2}$. Hence $u_{k+1} = u_k + \alpha'_k(\overline{x_k} - x'_k)$.

Similarly, choose $K' > K'_1$ so large and for $k \geq K'$ then

$v_{k+1} = v_k + \beta'_k(\overline{y_k} - y'_k)$.

Now, for u_{k+1}, $\overline{x_k} \in X_1$ meanwhile x'_k will be disappeared in the representation of u_{k+1}.

(i.e x'_k has zero coefficient α'_k in u_{k+1})

Then there is $l \in \{1, ..., m\}$ such that u_{k+l} does not involve the vector $x_i \in X_2$

so there $k > K + l$ such that $u_k \in G$.

Similarly, there is some k' such that $v'_k \in G'$. ■

CHAPTER 3

SUPPORT VECTOR MACHINES

3.1. Introduction for Support Vector Machines (SVM)

Support vector machines (SVM) is a family of learning algorithms used for the classification of objects into two classes. The theory is developed by Vapnik and Chervonenkis. SVM became very popular in the early 90's. SVM has been broadly applied to all kinds of classifications from handwritten recognition to face detection in image. SVM has gained popularity because it can be applied to a wide range of real world applications and computational efficiency. Moreover, it is theoretically robust.

In linearly separable data sets, there are many decision boundaries. In order to have a good generalization for a new decision, a good decision boundary is needed. The decision boundaries of two sets should be as far away from two classes as possible in order to make an effective generalization.

The shortest distance between two boundaries is called a margin. SVM maximizes the margin so that the maximized margin results in less errors for a future test, while other methods only reduce classification errors. In this view, SVM enables more generalization than other methods. To illustrate this, consider the Figure 3.1.

Line (1) and (2) have zero errors for classifying the two groups, but the goal of this classification is to classify future inputs. If test point T belongs to O group , then (1) and (2) are not the same any more. This means that (2) is a better generalization than (1).

Maximal margin can be obtained by solving a quadratic programming (QP) optimization problem. The optimal hyperplane which has the maximal margin can be obtained by solving the quadratic programming optimization problem. The SVM QP optimization problem will be introduced in Section 3.2. A classifier will be defined in Section 3.2 as well. In 1992, Bernhard Boser, Isabelle Guyon and Vapnik added a kernel trick which will be introduced

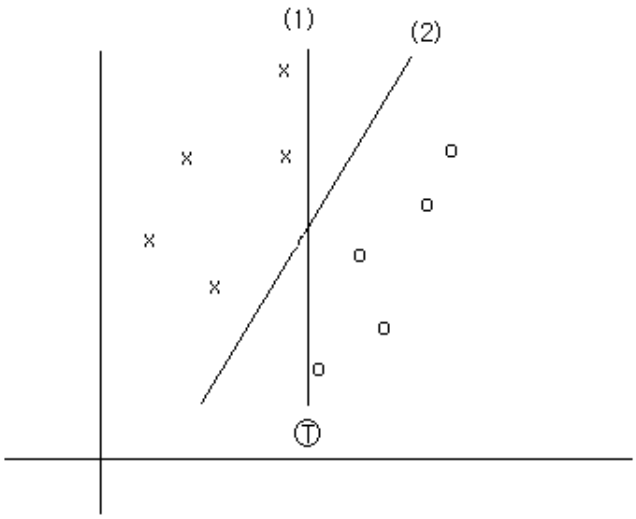

FIGURE 3.1. SVM Generalization

in Section 3.5 to linear classifier. This kernel trick makes linear classification possible in a feature space that is usually bigger than the original space.

3.2. SVM for Linearly Separable Sets

This section will introduce the most basic SVM problem: SVM with linearly separable sets. Let S be a training of points $x_i \in R^m$ with $t_i \in \{-1, 1\}$ for $i = 1, ..., N$, then

$$S = \{(x_1, t_1), ..., (x_N, t_N)\}$$

i.e If $t_i = +1$ then x_i belongs to the first class.

If $t_i = -1$ then x_i belongs to the second class.

Let S be linearly separable, then there is a hyperplane $wx + b = 0$ where w in R^m and $wx + b = 0$ correctly classifies all in S.

Define classifier $f(x_i) = wx_i + b$, $f(x_i) = 1$ if $t_i = 1$ and $f(x_i) = -1$ if $t_i = -1$.

i.e $wx_i + b > 0$ if $t_i = +1$

$wx_i + b < 0$ if $t_i = -1$

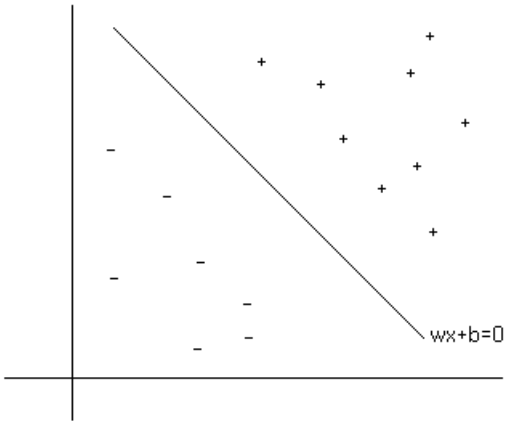

FIGURE 3.2. Linear Classifier

Even if there is a classifier that separates the training set correctly, a maximal margin is needed. In order to separate the two sets correctly and a have better generalization, it is necessary to maximize the margin. To compute an optimal hyperplane, consider the following inequalities for any $i = 1, ..., N$

$$wx_i + b > 1 \quad \text{if } t_i = +1$$

$$wx_i + b < -1 \text{ if } t_i = -1$$

These two inequalities are decision boundaries. To compute the margin, consider the following two hyperplanes :

$$wx_1 + b = 1$$

$$wx_2 + b = 0$$

where distance from x_1 to x_2 is the shortest distance of two hyperplanes. This means that the direction $x_2 - x_1$ is parallel to w.

Subtracting the first from the second equality, $w(x_2 - x_1) = 1$ is obtained, since w and $x_2 - x_1$ have the same direction, $\|x_2 - x_1\| = \frac{1}{\|w\|}$. In order to minimize the error, $\|x_2 - x_1\|$ should be maximized. Maximizing $\|x_2 - x_1\|$ is equivalent to minimizing $\|w\|$.

Thus, the optimal hyperplane can be found by solving the following optimization problem.

$$\min \frac{1}{2} \|w\|^2 \tag{70}$$

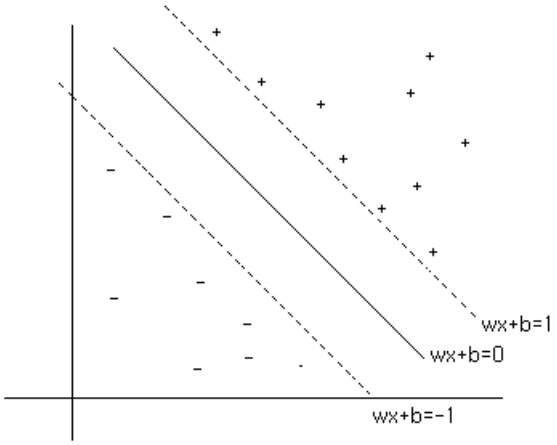

FIGURE 3.3. Classifier with Margin

$$\text{subjec to} \quad t_i(w^T x_i + b) \geq 1$$

3.3. SVM Dual Formulation

This section will introduce the dual formulation of SVM optimization problem. The dual formulation is usually efficient in implementation. Also, in dual formulation, a non-separable set will be more generalized. The most important aspect of dual formulation is the kernel trick. The kernel trick can be used in the dual formulation. The last part of this section will show the dual formulation using KKT condition in optimization theory.

Let's consider the Lagrangian function from (70),

$$L(w, b, \lambda) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{N} \lambda_i [t_i(w^T x_i + b) - 1] \tag{71}$$

$$\text{where} \quad \lambda = (\lambda_1, ...\lambda_N)$$

and differentiate with respect to the original variables.

$$\frac{\partial L}{\partial w}(w, b, \lambda) = w - \sum_{i=1}^{N} \lambda_i t_i x_i = 0. \tag{72}$$

$$\frac{\partial L}{\partial b}(w, b, \lambda) = -\sum_{i=1}^{N} \lambda_i t_i = 0. \tag{73}$$

From (72),

$$w = \sum_{i=1}^{N} \lambda_i t_i x_i. \tag{74}$$

Substitute (74) into (71)

$$L(w, b, \lambda) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} t_i t_j \lambda_i \lambda_j x_i x_j - \sum_{i=1}^{N} \sum_{j=1}^{N} t_i t_j \lambda_i \lambda_j x_i x_j - b \sum_{i=1}^{N} \lambda t_i + \sum_{i=1}^{N} \lambda_i$$

$$= -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} t_i t_j \lambda_i \lambda_j x_i x_j + \sum_{i=1}^{N} \lambda_i.$$

Thus SVM-DUAL is the following.

$$\max -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} t_i t_j \lambda_i \lambda_j x_i x_j + \sum_{i=1}^{N} \lambda_i \tag{75}$$

$$s.t \quad \sum_{i=1}^{N} \lambda_i t_i = 0$$

$$\lambda_i \geq 0$$

Once λ is found, (w, b) can be found easily by using $w = \sum_{i=1}^{N} \lambda_i t_i x_i$ and $wx_i + b = 1$ for x_i in the first class.

3.4. Linearly Nonseparable Training Set

In the case of a linearly nonseparable training set, linear classifier $wx + b = 0$ cannot separate the training set correctly. This means that the linear classifier needs to allow some errors. The following three cases will happen with linear classifier $wx + b = 0$.

Case 1 : Satisfy $t_i(wx_i + b) \geq 1$

Case 2 : $0 \leq t_i(wx_i + b) < 1$

Case 3 : $t_i(wx_i + b) < 0$

Case 2 and case 3 have errors while Case 1 does not have an error. These errors can be measured using slack variable s_i. In Case 1, $s_i = 0$. In Case 2, $0 < s_i < 1$. And in Case 3, $s_i > 1$.

Figure 3.4 shows the case of a linearly nonseparable set.

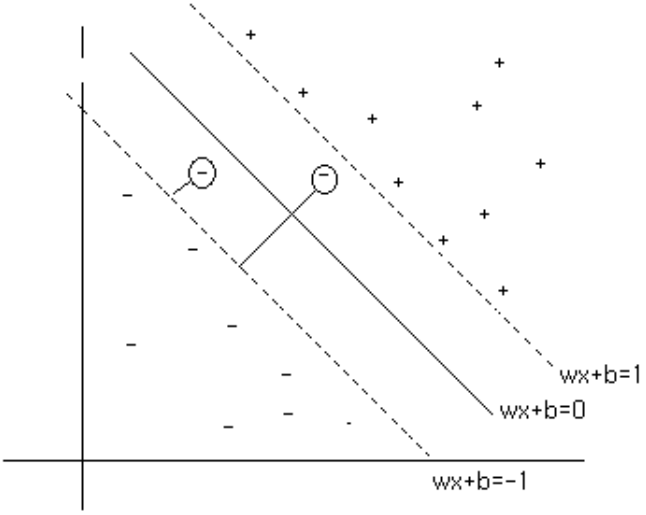

FIGURE 3.4. Linearly Nonseparable Set

In the case of a linearly nonseparable set, the margin is maximized while the number of $s_i > 0$ is minimized. So there are two goals. The first goal is to maximize the margin, and the second goal is to minimize the number of $s_i > 0$.

Now, consider

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} s_i \tag{76}$$

In (76), C was introduced. Let's think of two extreme cases. When $C = 0$, the second goal can be ignored. This is the case of a linearly separable set. When $C \to \infty$, the margin is ignored. C is a parameter that the user of SVM can choose to tune the trade off between the width of margin and the number of s_i. So the optimization problem for nonlinearly separable case is the following :

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} s_i \tag{77}$$

$$s.t \qquad t_i(wx_i + b) \geq 1 - s_i$$

$$s_i \geq 0$$

$$i = 1, 2, ..., N$$

Now let's consider Lagrangian function from (77).

$$L(w, b, s, \lambda, \mu) = (\frac{1}{2} \|w\|^2 + C \sum_{i=1}^{N} s_i) - [\sum_{i=1}^{N} \lambda_i (t_i (w^T x_i + b) - 1 + s_i) + \sum_{i=1}^{N} \mu_i s_i] \tag{78}$$

And differentiate with respect to the original variables.

$$\frac{\partial L}{\partial w}(w, b, s, \lambda, \mu) = w - \sum_{i=1}^{N} \lambda_i t_i x_i = 0 \tag{79}$$

$$\frac{\partial L}{\partial b}(w, b, s, \lambda, \mu) = -\sum_{i=1}^{N} \lambda_i t_i = 0 \tag{80}$$

$$\frac{\partial L}{\partial s}(w, b, s, \lambda, \mu) = C - \lambda - \mu = 0 \tag{81}$$

From (78),

$$w = \sum_{i=1}^{N} \lambda_i t_i x_i \tag{82}$$

Substitute (82) into (78),

$L(w, b, s, \lambda, \mu)$
$= \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} t_i t_j \lambda_i \lambda_j x_i x_j + C \sum_{i=1}^{N} s_i - \sum_{i=1}^{N} \sum_{j=1}^{N} t_i t_j \lambda_i \lambda_j x_i x_j - b \sum_{i=1}^{N} \lambda t_i - \sum_{i=1}^{N} \lambda_i s_i - \sum_{i=1}^{N} \mu s_i + \sum_{i=1}^{N} \lambda_i$
$= -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} t_i t_j \lambda_i \lambda_j x_i x_j + \sum_{i=1}^{N} \lambda_i + C \sum_{i=1}^{N} s_i - \sum_{i=1}^{N} \lambda_i s_i - \sum_{i=1}^{N} \mu s_i$
By (81)
$= -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} t_i t_j \lambda_i \lambda_j x_i x_j + \sum_{i=1}^{N} \lambda_i$
Thus, Daul-SVM linearly separable is obtained as follows

$$\max -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} t_i t_j \lambda_i \lambda_j x_i x_j + \sum_{i=1}^{N} \lambda_i \tag{83}$$

$$s.t \quad \sum_{i=1}^{N} \lambda_i t_i = 0$$

$$0 \leq \lambda_i \leq C$$

$$i = 1, 2, ..., N$$

3.5. Kernel SVM

In many problems, a linear classifier could not have both high performance and good generalization in the original space.

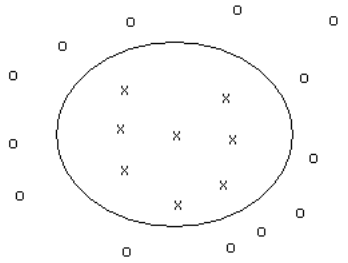

FIGURE 3.5. Nonlinear Classifier

In Fig 3.5, a linear classifier cannot separate two data sets correctly, so a nonlinear classifier must be considered. As a result, a nonlinear classifier produces an efficient generalization for the future test. As it is shown in the above Figure 3.5, the nonlinear classifier has a good performance (i.e., generalization.) Then how does SVM separate a nonlinear separable data set? Consider the training set $S = \{(x_1, t_1), ..., (x_N, t_N)\}$ which is not linearly separable. However S can be linearly separable in a different space which is usually a higher dimension than the original space using the set of real function $\phi_1, ..., \phi_M$. Here, assume ϕ to be unknown. These functions are called features . Using ϕ, $x = (x_1, .., x_m)$ can be mapped to $\phi(x) = (\phi_1(x), ..., \phi_M(x))$. After mapping to the feature space, the new training set $S' = \{(\phi(x_1), t_1), ..., (\phi(x_N), t_N)\}$ is obtained.

To see this clearly, consider the following example. $S_1 = \{(a, 1), (b, -1), (c, -1), (d, 1)\}$ where $a = (0, 0), b = (1, 0), c = (0, 1)$ and $d = (1, 1)$. S_1 cannot be linearly separable without making any error.

Now let $\phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$ where $x = (x_1, x_2)$. By the feature function $\phi(x)$,

$a = (0,0) \rightarrow \phi(a) = a' = (0,0,0)$

$b = (1,0) \rightarrow \phi(b) = b' = (1,0,0)$

$c = (0,1) \rightarrow \phi(c) = c' = (0,0,1)$

$d = (1,1) \rightarrow \phi(d) = d' = (1,\sqrt{2},1)$

First, check that $S_1' = \{(a',1),(b',-1),(c',-1),(d',1)\}$ is linearly separable in R^3, so $\phi(x)$ can be seen. However, as mentioned earlier, ϕ usually is unknown if the kernel function is used. Let $x = (x_1, x_2)$ and $y = (y_1, y_2)$.

Then define $K(x,y) = (x,y)^2 = x_1^2y_1^2 + 2x_1y_1x_2y_2 + x_2^2y_2^2$. We can easily check $K(a,b) = \phi(a)\phi(b)$

$K(c,d) = \phi(c)\phi(d)$.

SVM uses a kernel function that the value of the kernel function is the same as the value of inner product in the feature space. Thus, using new training set S' and by (83), let's consider the following optimization problem :

$$\max -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} t_it_j\lambda_i\lambda_j\phi(x_i)\phi(x_j) + \sum_{i=1}^{N}\lambda_i \tag{84}$$

$$s.t \quad \sum_{i=1}^{N}\lambda_i t_i = 0$$

$$0 \leq \lambda_i \leq C$$

$$i = 1, 2, ..., N$$

Now, define the kernel $K(x_i, x_j) = \phi(x_i)\phi(x_j)$ for any inner product (x_i, x_j) in the original space. Then (84) can be written as the following :

$$\max -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} t_it_j\lambda_i\lambda_j K(x_i,x_j) + \sum_{i=1}^{N}\lambda_i \tag{85}$$

$$s.t \quad \sum_{i=1}^{N} \lambda_i t_i = 0$$

$$0 \leq \lambda_i \leq C$$

$$i = 1, 2, ..., N$$

By the definition of kernel, $K(x_i, x_j)$ is always corresponding to the inner product in some feature space. The kernel $K(x_i, x_j)$ can be computed without computing the image $\phi(x_i)$ and $\phi(x_j)$.

(85) is called kernel substitution. By using kernel substitution, inner product evaluations can be computed in the original space without knowing the feature space.

Now, the followings are popular kernel functions.

1. Polynomial kernels

   $K(x_i, x_j) = [(x_i, x_j) + c]^d$

   where d is the degree of the polynomial and c is a constant.

2. Radial basis function kernel

   $K(x_i, x_j) = e^{-\|x_i - x_j\|/2\gamma^2}$

   Where γ is a parameter.

3. Sigmoid kernel

   $K(x_i, x_j) = \tanh[\alpha(x_i, x_j) + \beta]$

   where α and β are parameters.

CHAPTER 4

SOLVING SUPPORT VECTOR MACHINES USING THE NEW ALGORITHM

4.1. Solving SVM Problems Using the New Algorithm

This chapter will show that Algorithm 2.1 can be used to solve SVM problems. Section 4.1 will show that solving SVM is equivalent to finding the minimum distance of two convex hulls. Finding the maximal margin of training data sets and finding the minimum distance of two convex hulls given by the training data sets seems to be different processes. Yet, after reformulating SVM-NV, SVM-NV can be recognized as a NPP.

Let $\{x_i\}_{i=1}^m$ be a training data set, Let I be the index set for the first class and J be the index set for the second class. Also, set $U = \{u \mid u = \sum \beta_i x_i, \sum \beta_i = 1, \beta_i \geq 0, i \in I\}$ and $V = \{v \mid v = \sum \beta_j x_j, \sum \beta_j = 1, \beta_j \geq 0, j \in J\}$

Reformulation of SVM-NV was done by Keerthi[2]. SVM Dual is equivalent to

$$\min \frac{1}{2} \sum_s \sum_t \beta_s \beta_t t_s t_t (x_s, x_t) \tag{86}$$

$$s.t \quad \beta_s \geq 0,$$

$$\sum_{i \in I} \beta_i = 1, \sum_{j \in J} \beta_j = 1$$

Now consider an NPP problem. In an NPP problem, $\|u - v\|^2$ needs to be minimized.

$$\|u - v\|^2$$

$$= \left\| \sum_i \beta_i x_i - \sum_j \beta_j x_j \right\|^2$$

$$= (\sum_i \beta_i x_i - \sum_j \beta_j x_j, \sum_i \beta_i x_i - \sum_j \beta_j x_j)$$

$$= (\sum_i \beta_i x_i, \sum_i \beta_i x_i) - (\sum_i \beta_i x_i, \sum_j \beta_j x_j) - (\sum_j \beta_j x_j, \sum_i \beta_i x_i) + (_j \beta_j x_j, \sum_j \beta_j x_j)$$

$$(87) \quad = \sum_i \sum_i \beta_i \beta_i (x_i, x_i) - \sum_i \sum_j \beta_i \beta_j (x_i, x_j) - \sum_i \sum_j \beta_i \beta_j (x_i, x_j) + \sum_j \sum_j \beta_j \beta_j (x_j, x_j).$$

Note that in (87), if the inner product of two vectors is from the same index set, then the sign of the inner product is positive. Otherwise, it is negative.

Thus, (87) can be written as the following:

$$(88) \qquad \sum_s \sum_t \beta_s \beta_t t_s t_t (x_s, x_t)$$

$$\text{where } t_s t_t = 1 \text{ if } x_s \text{ and } x_t \text{ are from the same index set}$$

$$t_s t_t = -1$$

The minimum distance between two convex hulls is nonzero if and only if two convex hulls do not intersect. That is, if two convex hulls intersect, then the minimum distance of two convex hulls is zero. As it is shown earlier in chapter 3, real SVM problems have some violations. If some x violates, then two convex hulls could intersect. However, having zeros for a minimum distance for two convex hulls is not desired. Here, Friess recommends using a sum of squared violations in the cost function.

$$\min \frac{1}{2} \|w\|^2 + \frac{\widetilde{C}}{2} \sum_k s_k^2$$

$$\text{s.t } \; wx_i + b > 1 - s_i$$

$$wx_i + b < -1 + s_i$$

This problem is called SVM Sum of Squared Violation (SVM-VQ). SVM-VQ can be converted to (89) by simple transformation. To see this, let's set

$$\widetilde{w} = \begin{pmatrix} w \\ \sqrt{\widetilde{C}} s \end{pmatrix}; \; b = \widetilde{b}; \; \widetilde{x_i} = \begin{pmatrix} x_i \\ \frac{1}{\sqrt{\widetilde{C}}} e_i \end{pmatrix}, \; i \in I \text{ and } \widetilde{x_j} = \begin{pmatrix} x_j \\ -\frac{1}{\sqrt{\widetilde{C}}} e_j \end{pmatrix}, j \in J$$

where e_i is the m dimensional vector in which the ith component is 1 and all are 0. Then

$$\frac{1}{2} \|\widetilde{w}\|^2 = \frac{1}{2} \|w\|^2 + \frac{\widetilde{C}}{2} \sum_k s_k^2$$

and

$$\widetilde{w}\widetilde{x_i} + \widetilde{b} = wx_i + b + s_i$$

$$\widetilde{w}\widetilde{x_j} + \widetilde{b} = wx_j + b - s_i$$

Thus, SVM-VQ is equivalent to

$$\min \frac{1}{2}\|\widetilde{w}\|^2 \tag{89}$$

$$\text{s.t} \quad \widetilde{w}\widetilde{x_i} + \widetilde{b} > 1$$

$$\widetilde{w}\widetilde{x_j} + \widetilde{b} < -1$$

Here, feasible space of SVM-VQ is non-empty because of slack variable s, and the result of (89) is automatically feasible.

By setting,
$\widetilde{x_i} = \begin{pmatrix} x_i \\ \frac{1}{\sqrt{\widetilde{C}}} e_i \end{pmatrix}$, $i \in I$ and $\widetilde{x_j} = \begin{pmatrix} x_j \\ -\frac{1}{\sqrt{\widetilde{C}}} e_j \end{pmatrix}, j \in J$. the new algorithm can be used as usual.

4.2. Stopping Method for the Algorithm

This section will show the stopping method for NPP using Algorithm 2.1. This section will use the KKT condition of optimization theory.

First, consider the general Quadratic Programming Problem.

(Quadratic Programming Problem)

$$\min \quad \frac{1}{2}x^tQx + cx + d$$

$$s.t \quad Ax - b = 0$$

$$x \geq 0.$$

KKT for the Quadratic programming is the following

$L(x, \lambda, \mu) = \frac{1}{2}x^tQx + cx + d - \lambda'(Ax - b) - \mu'x = 0$

1. $\frac{\partial L(x,\lambda,\mu)}{\partial x} = Qx + c - \lambda^t A - \mu = 0$
2. $\lambda_i(Ax - b)_i = 0, \mu^t x = 0$
3. $\lambda_i \geq 0, \mu_i \geq 0$

By KKT condition, $x_i \neq 0$ then $(Qx + c - \lambda^t A)_i = 0$. This means if nonzero index i is known, then $(Qx + c - \lambda^t A)_i = 0$. Using 2, the linear system can be solved to solve the Quadratic programming problem.

By (88), the NPP problem is equivalent to the following Quadratic programming problem.

$$\min \tfrac{1}{2} a^t A a \qquad \text{where } A = \begin{bmatrix} X^tX & -X^tY \\ -Y^tX & Y^tY \end{bmatrix} \text{ (NPP-QP)} \quad s.t \sum_{i=1} a_i = 2, a_i \geq 0$$

Consider lagrangian for NPP-QP

$$L(a, \lambda, u) = \frac{1}{2} a^t A a - \lambda(a^t e - 2) - \mu^t a$$

Then by KKT condition,

1. $Aa - \lambda e - \mu = 0$
2. $\lambda(a^t e - 2)_i = 0$
3. $\mu^t a = 0$
4. By 1, $\mu = Aa - \lambda e$
5. By 3, if $a_i \neq 0, (Aa - \lambda e)_i = 0$
6. By 3, if $a_i = 0, (Aa - \lambda e)_i \geq 0$

Then by the above, if the nonzero indices for a is known, only linear system 2 and 5 need to be solved. Finally, by using theorem 4 in chapter 2, it is noted that indices for u_k and v_k are not changed for large k. From this, it can be assumed that u_k and v_k are on G and G' respectively. Once u_k and v_k are on G on G' respectively, nonzero indices for a can be determined in the NPP-QP. After solving linear system 2 and 5, condition 6 must be checked to determine if the zero indices are satisfied.

CHAPTER 5

EXPERIMENTS

5.1. Comparison with Linearly Separable Sets

This section will compare performance between Algorithm 2.1 and sequential minimal optimization(SMO) using linearly separable sets. The MATLAB program was used to compare the performance. First, performance of Algorithm 2.1 and the MDM algorithm are compared in dimension 100. By increasing the number of points, the results of experiments can be seen in Figure 5.1. Second, twenty non-intersected pairs of convex hulls in dimensions 100, 200, 300, 500, 700, 1000, 1200, 2000, and 2500 are randomly generated. Then the minimum distances between two convex hulls is computed. In each dimension, the number of vectors is changed to see the performance of two algorithms. The following graphs show CPU times for Algorithm 2.1 and SMO in linear kernel with coefficient 0, linear kernel with coefficient 1, polynomial kernel with degree 3, and polynomial kernel with degree 5. In the graphs, linear kernel with coefficient 0 was denoted by Lin(0), linear kernel with coefficient 1 was denoted by Lin(1), polynomial kernel with degree 3 was denoted by P3, and polynomial kernel with degree 5 was denoted by P5. Also in the graphs, for example the expression 7000×100 indicates that 7000 vectors are in dimension 100. Figures 5.2 to 5.12 illustrate the results for the experiments in section 5.1.

5.2. Comparison with Real World Problems

In this section, the performance between algorithm 2.1 and sequential minimal optimization (SMO) is compared with real world problems. Four kernel functions are used to compare the performance of Algorithm 2.1 and SMO. Those kernels are polynomial kernel with degree 3, polynomial kernel with degree 5, radical basis with gamma 1 and radical basis with gamma 10. Polynomial kernel with degree 3 , polynomial kernel with degree 5 , radical basis

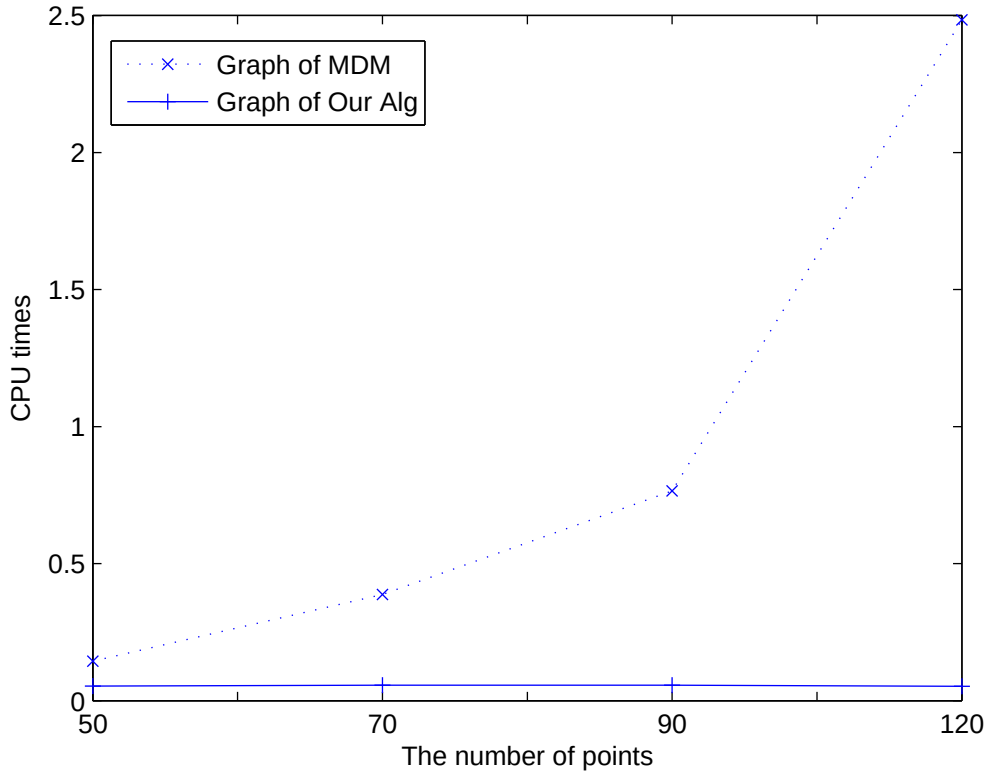

FIGURE 5.1. Algorithm 2.1 and the MDM Algorithm

with gamma 1 and radical basis with gamma 10 are denoted d3,d5, g1 and g10 respectively. Figures 5.13 to 5.16 illustrate the results for the experiments in section 5.2.

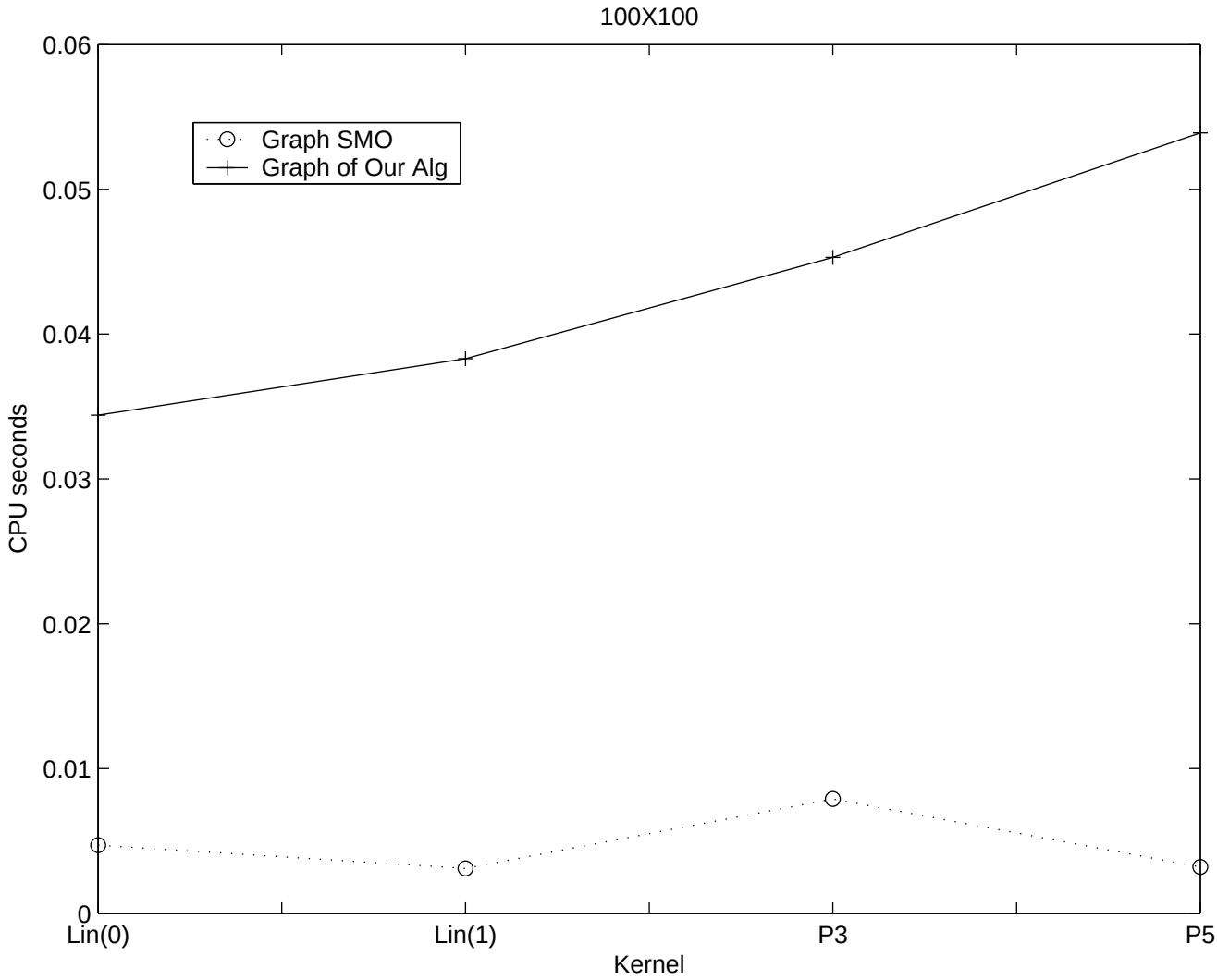

FIGURE 5.2. 100 Points in Dimension 100

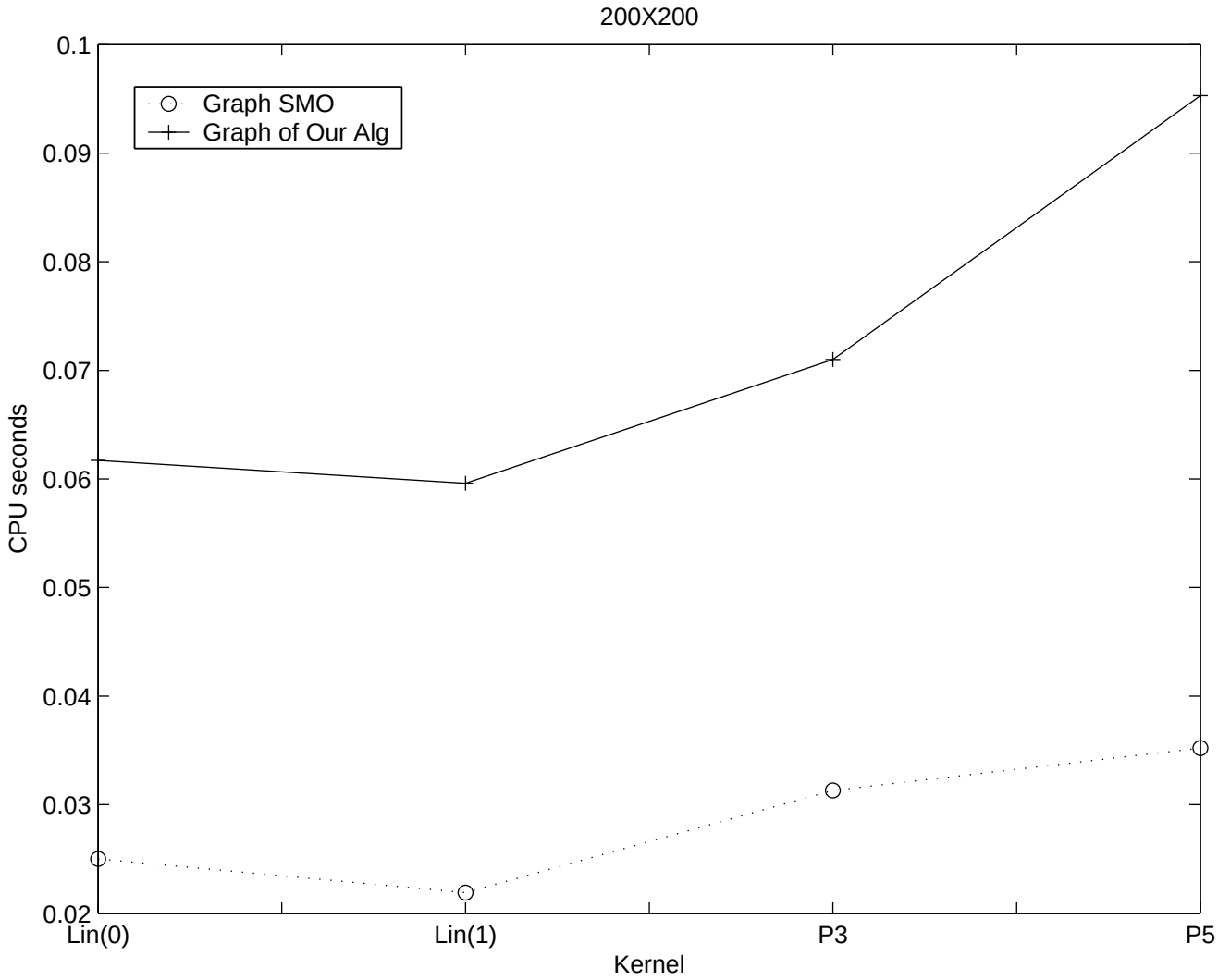

FIGURE 5.3. 200 Points in Dimension 200

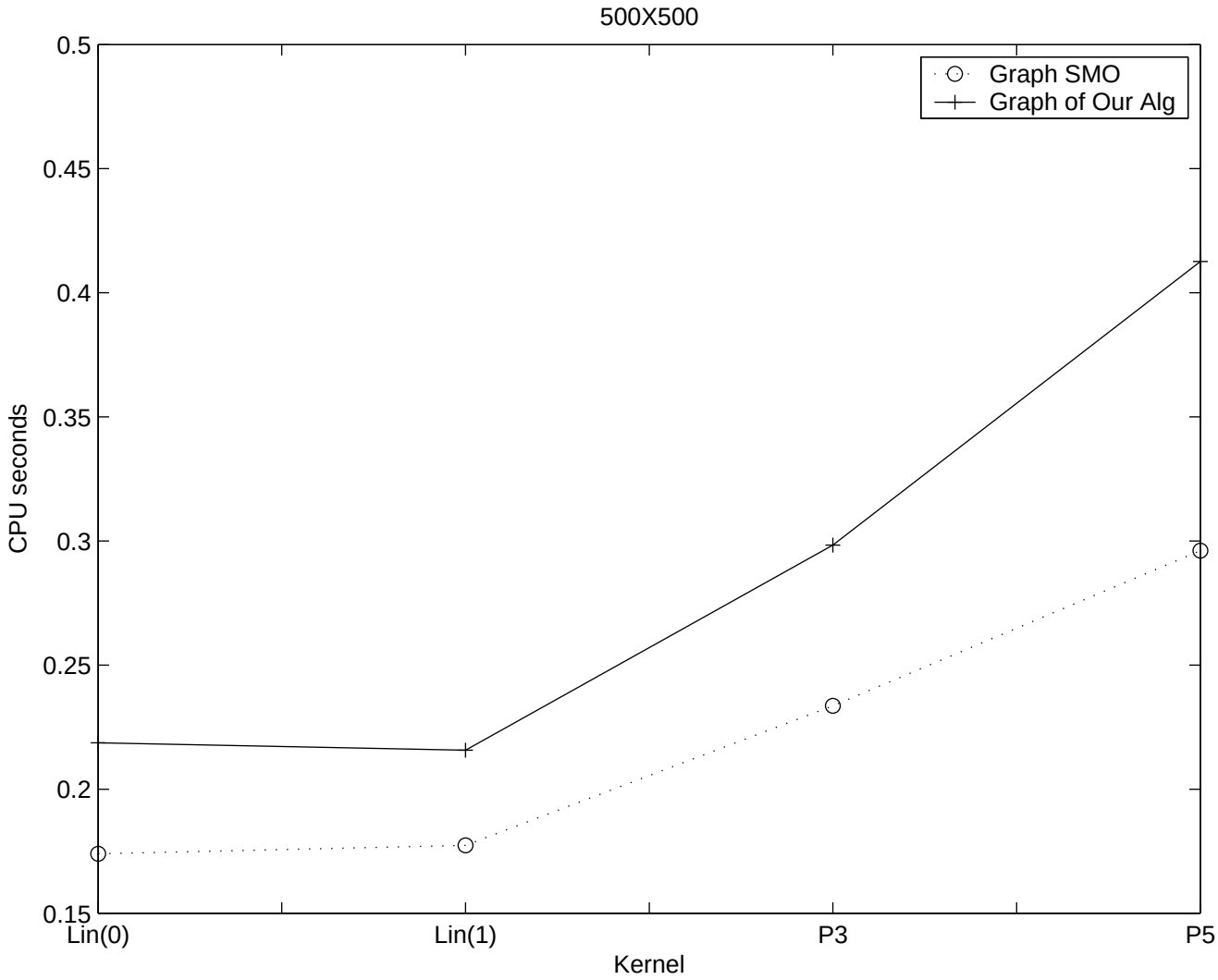

FIGURE 5.4. 500 Points in Dimension 500

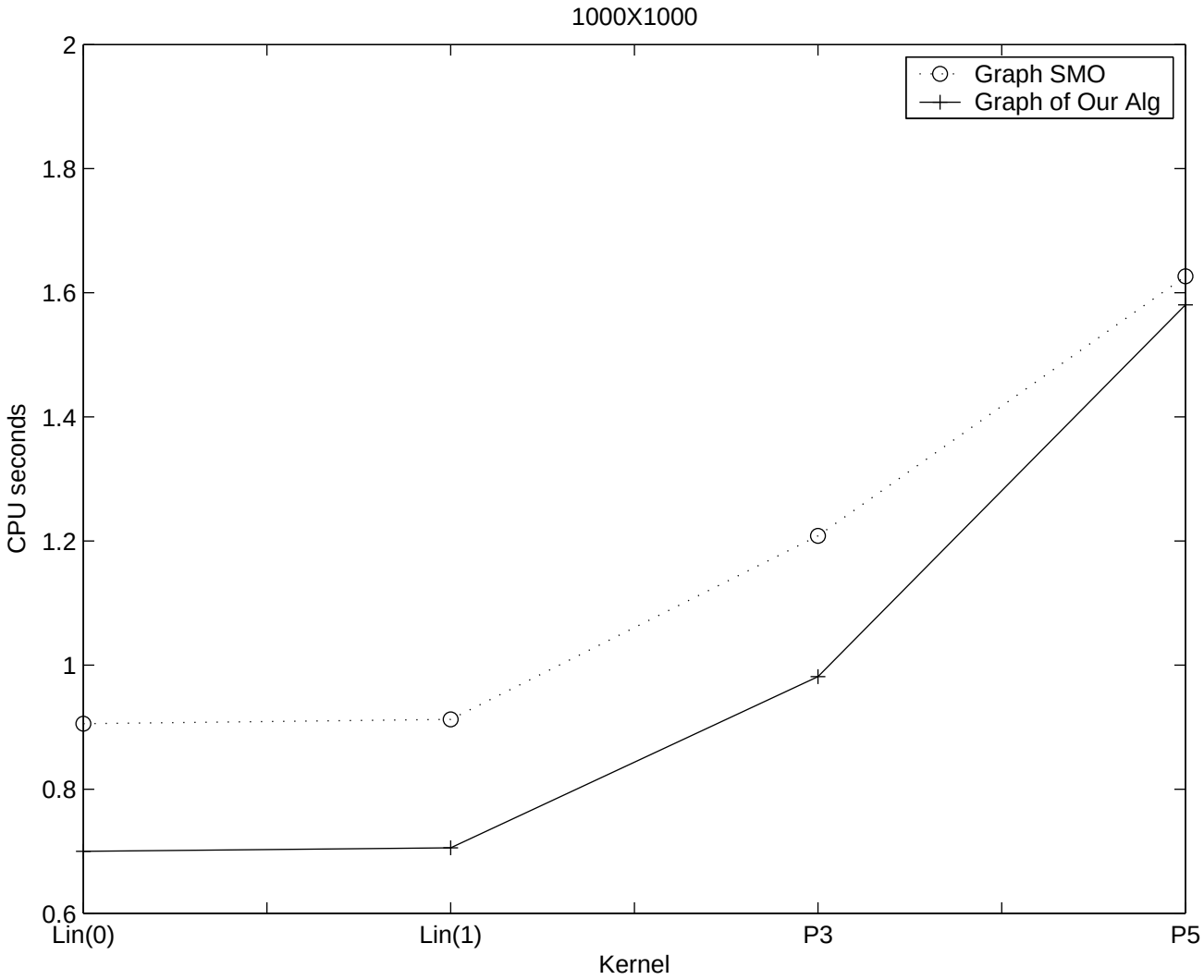

FIGURE 5.5. 1000 Points in Dimension 1000

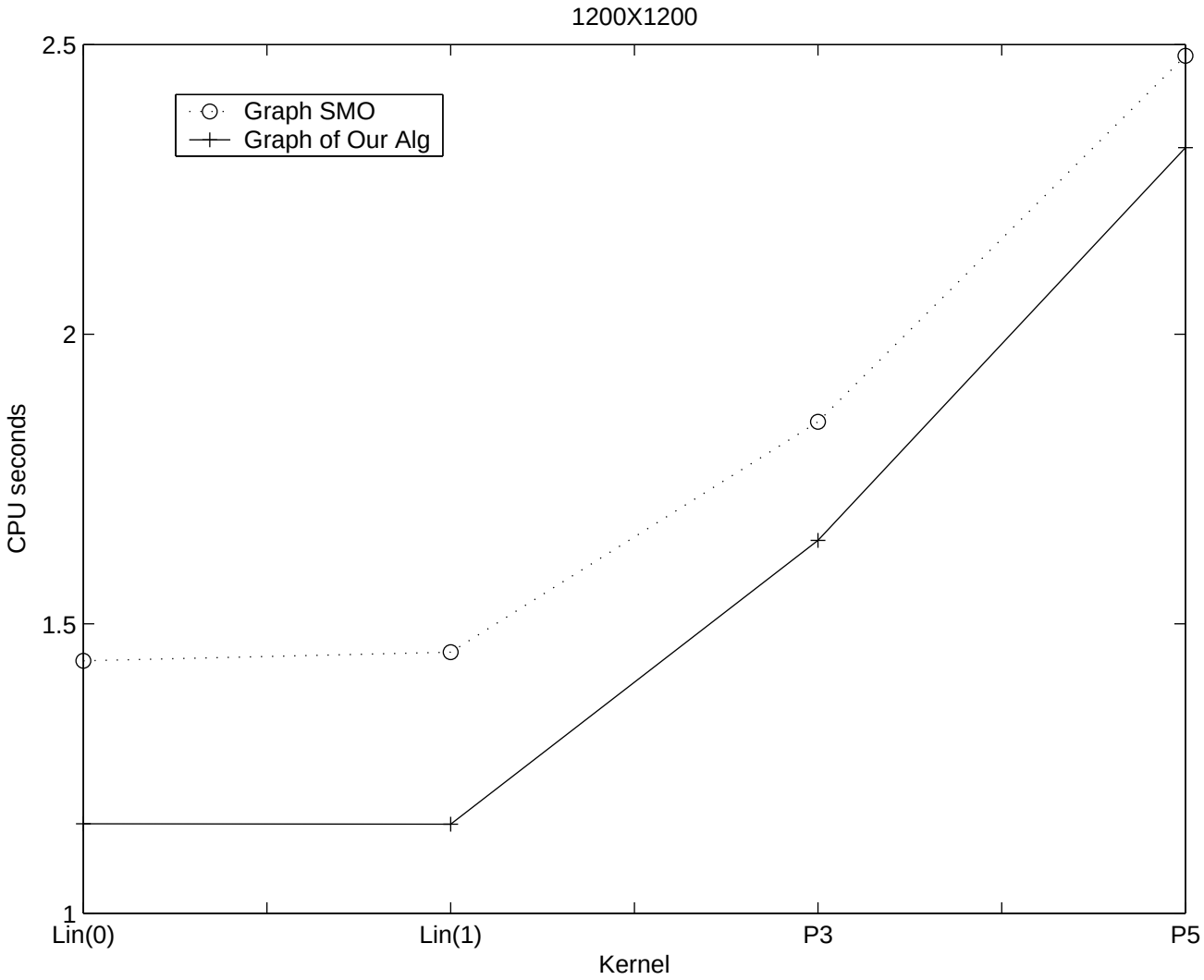

FIGURE 5.6. 1200 Points in Dimension 1200

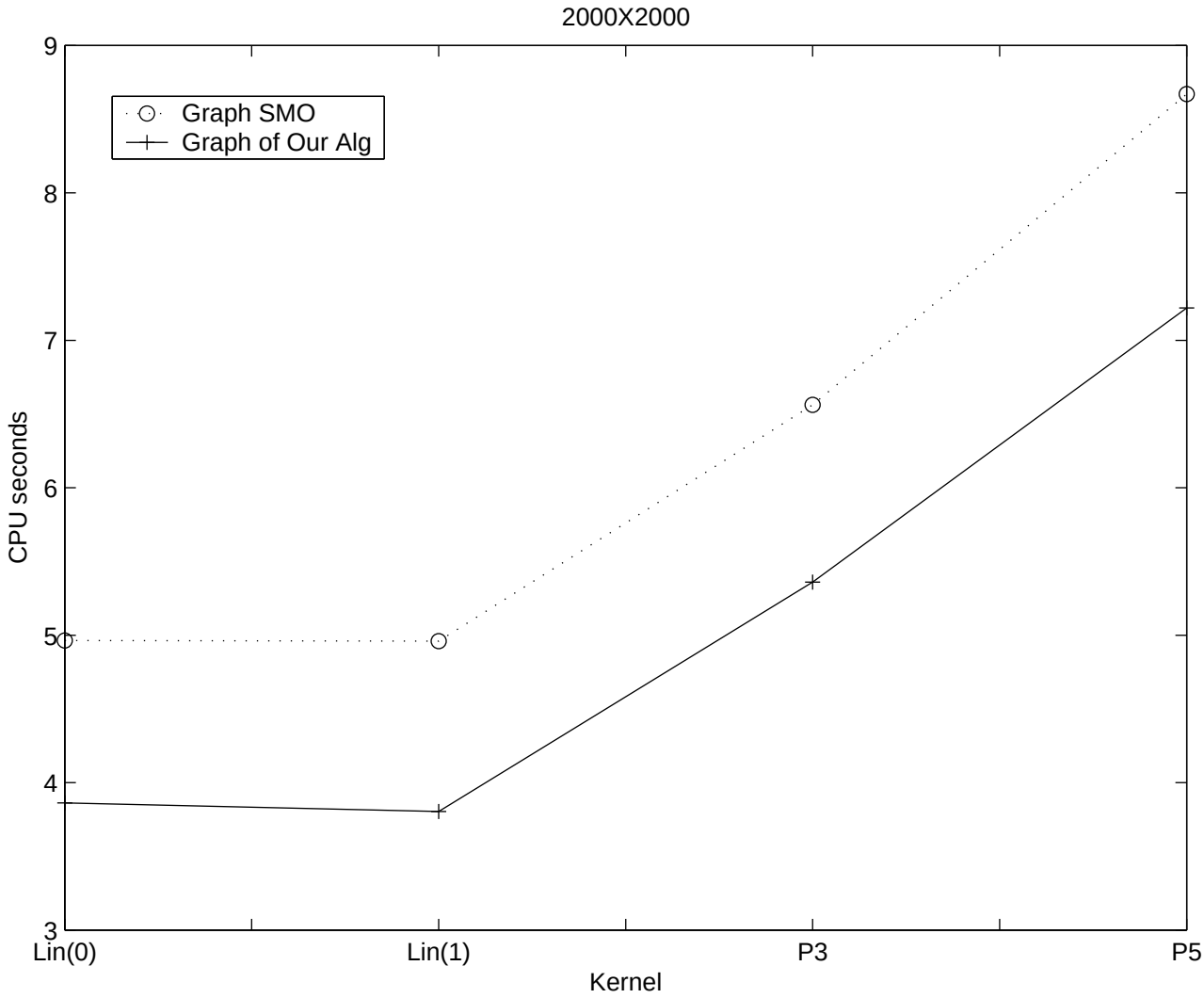

FIGURE 5.7. 2000 Points in Dimension 2000

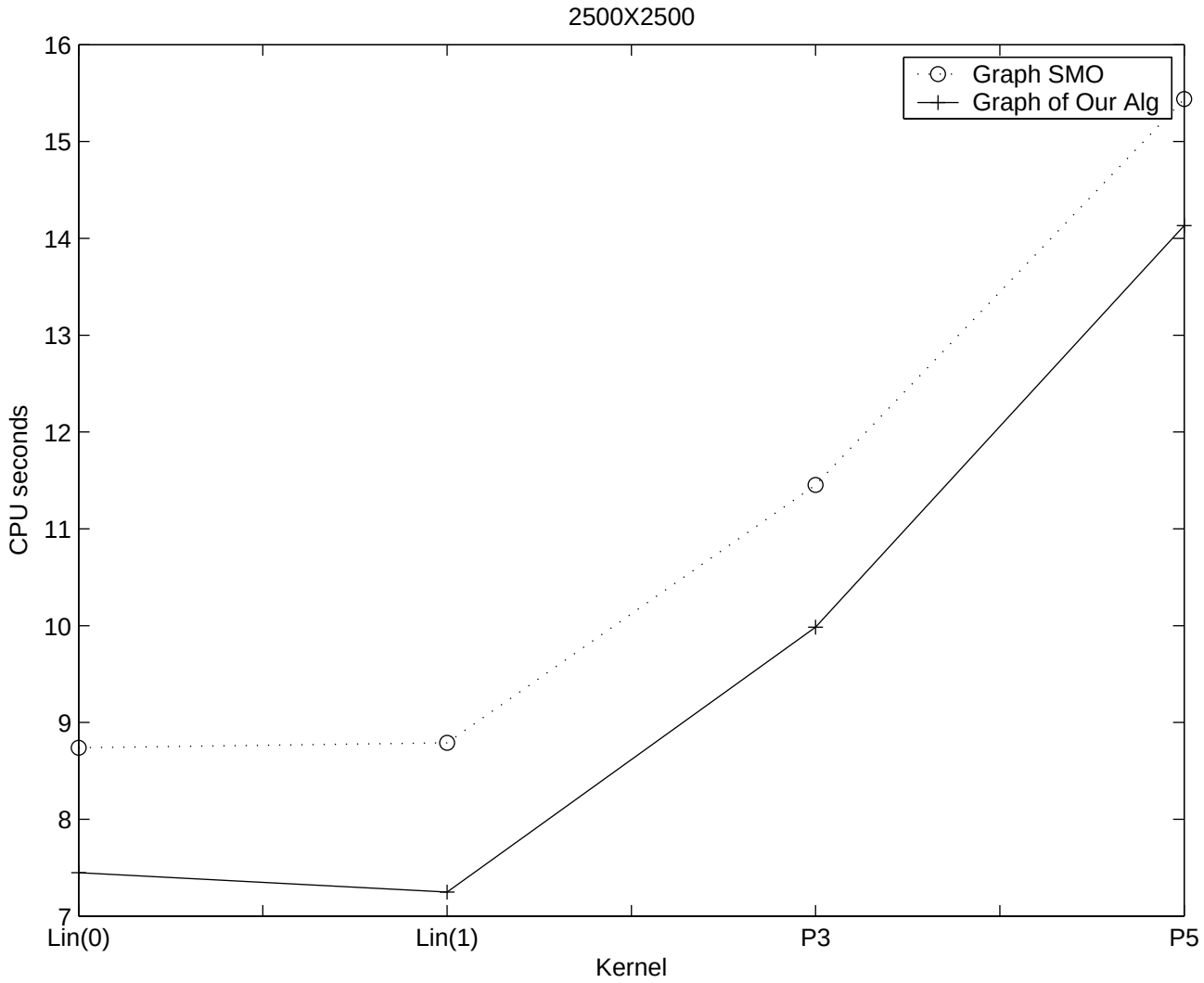

FIGURE 5.8. 2500 Points in Dimension 2500

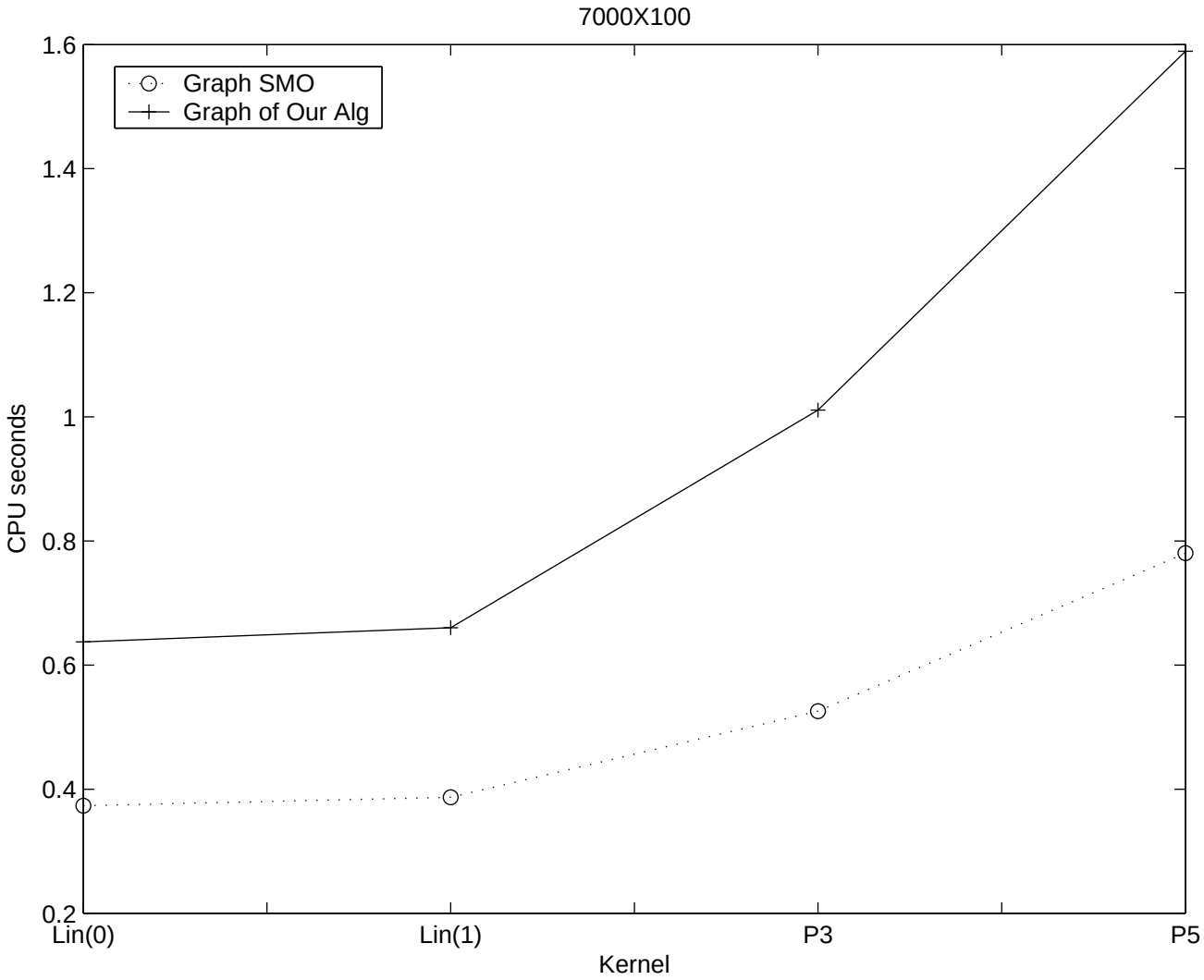

FIGURE 5.9. 7000 Points in Dimension 100

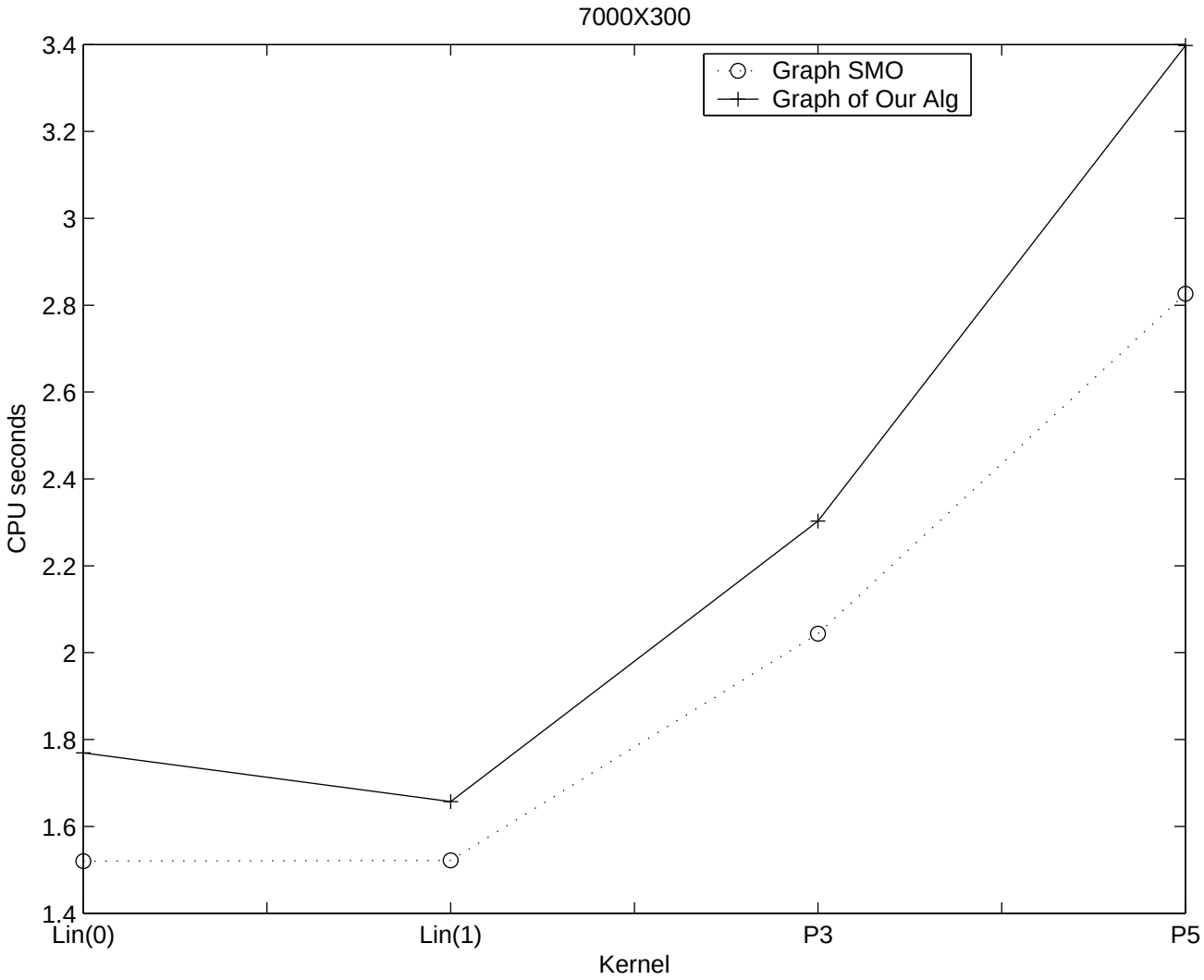

FIGURE 5.10. 7000 Points in Dimension 300

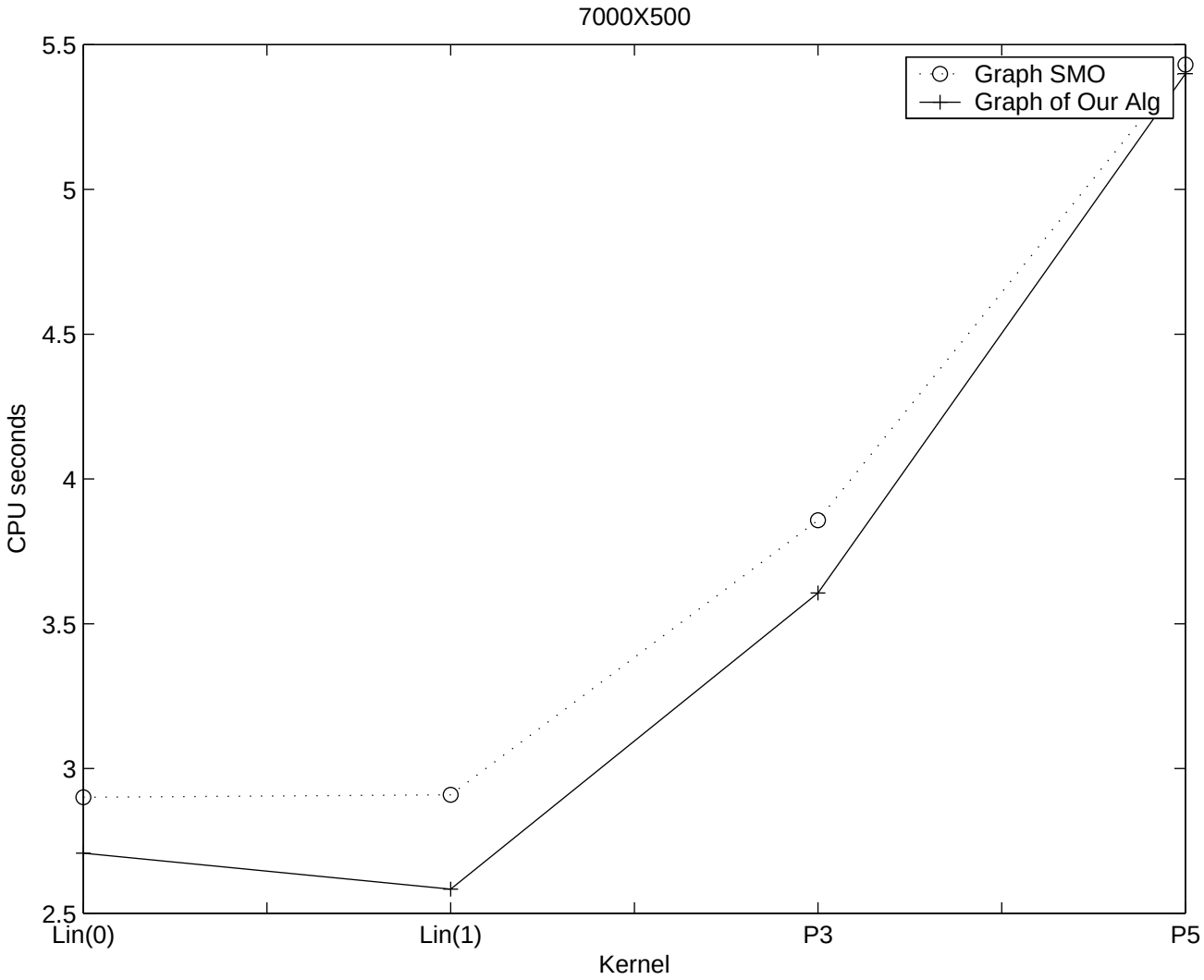

FIGURE 5.11. 7000 Points in Dimension 500

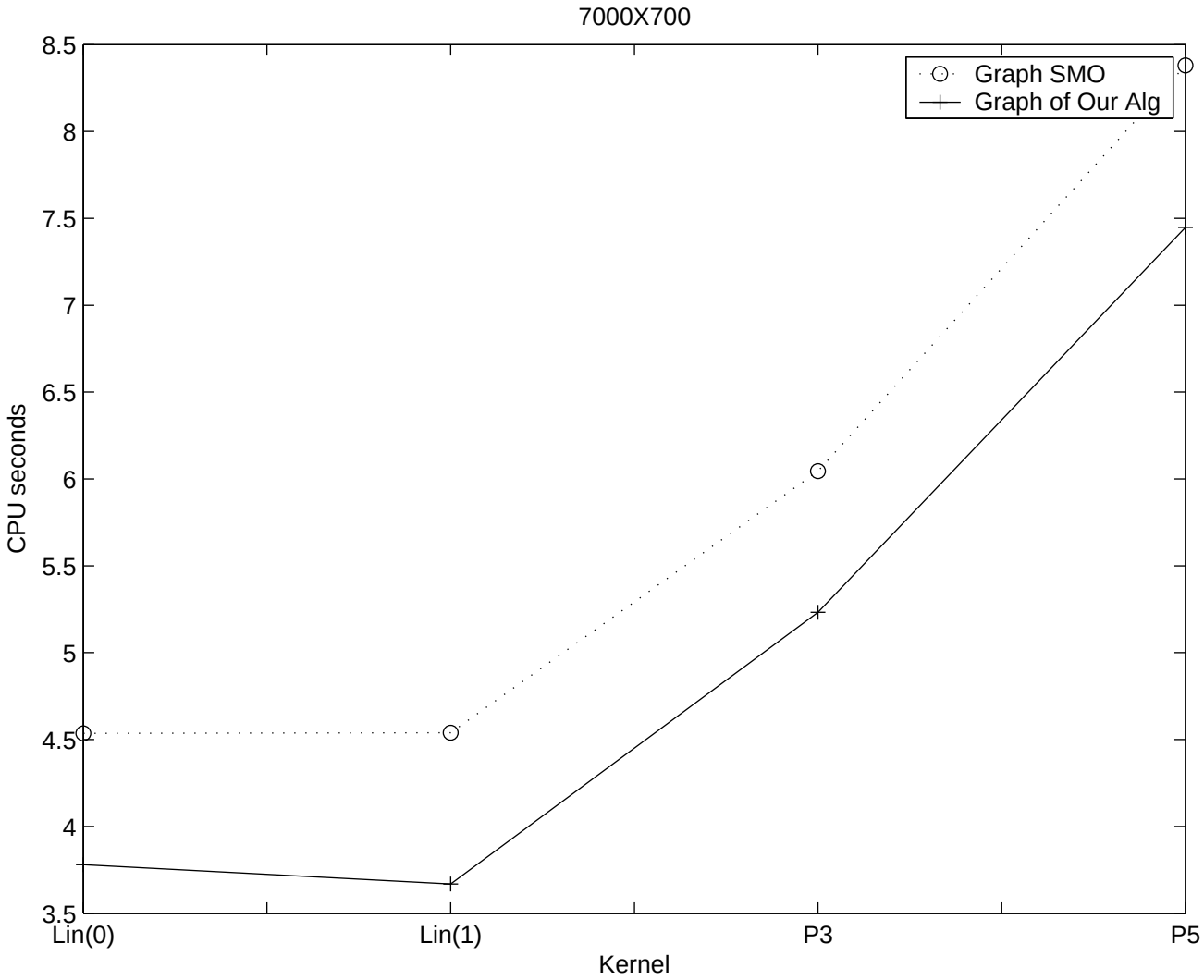

FIGURE 5.12. 7000 Points in Dimension 700

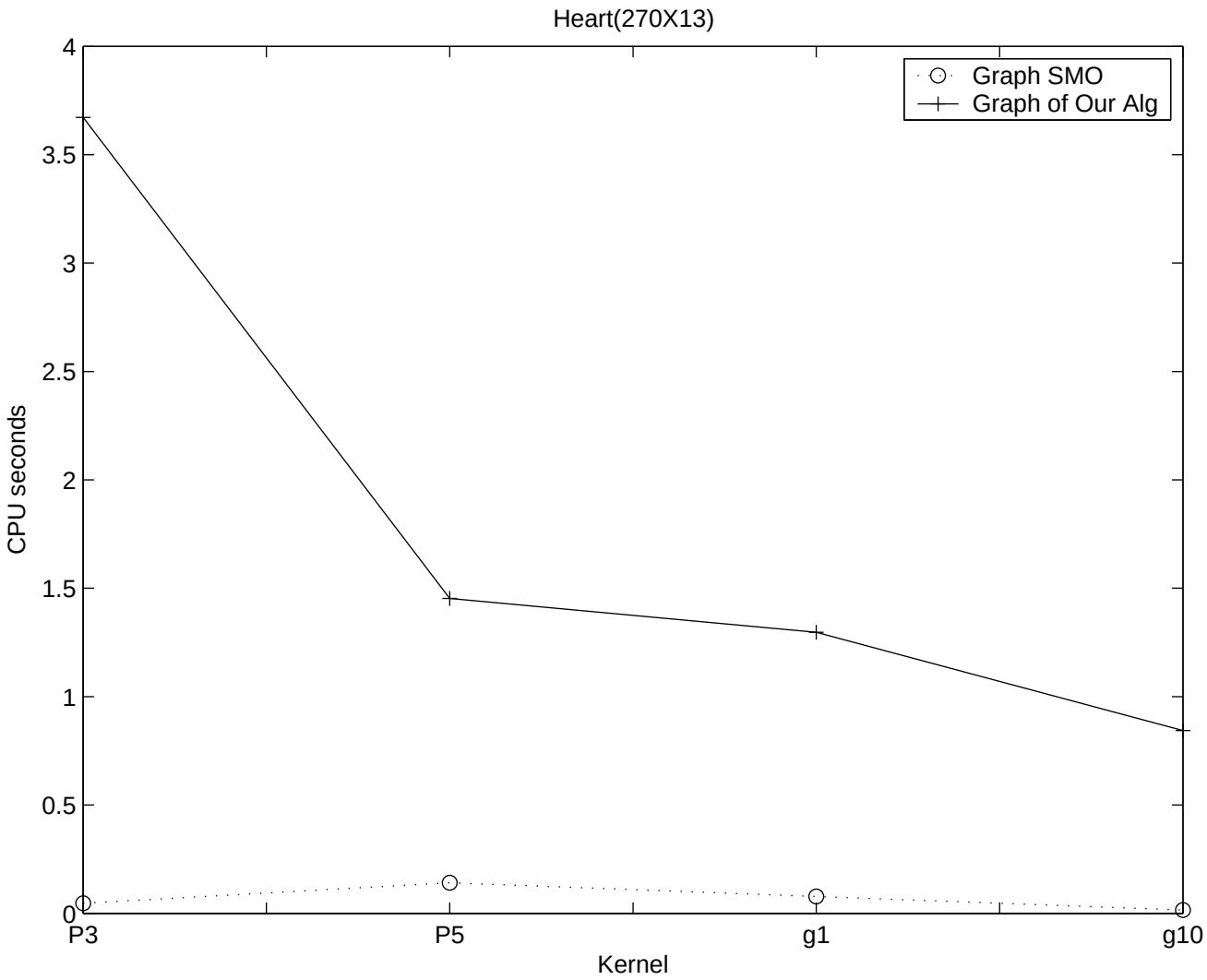

FIGURE 5.13. Heart

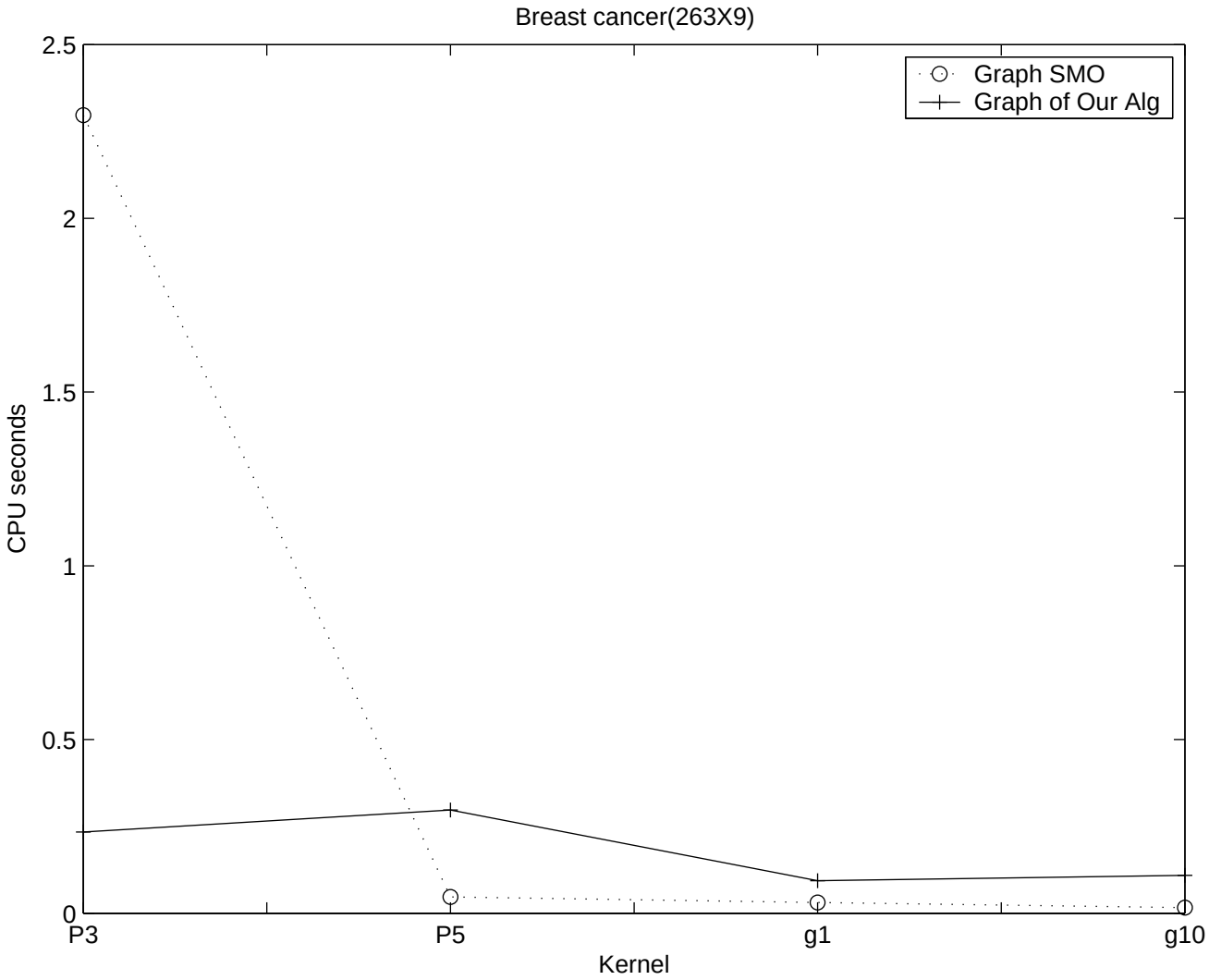

FIGURE 5.14. Breast Cancer

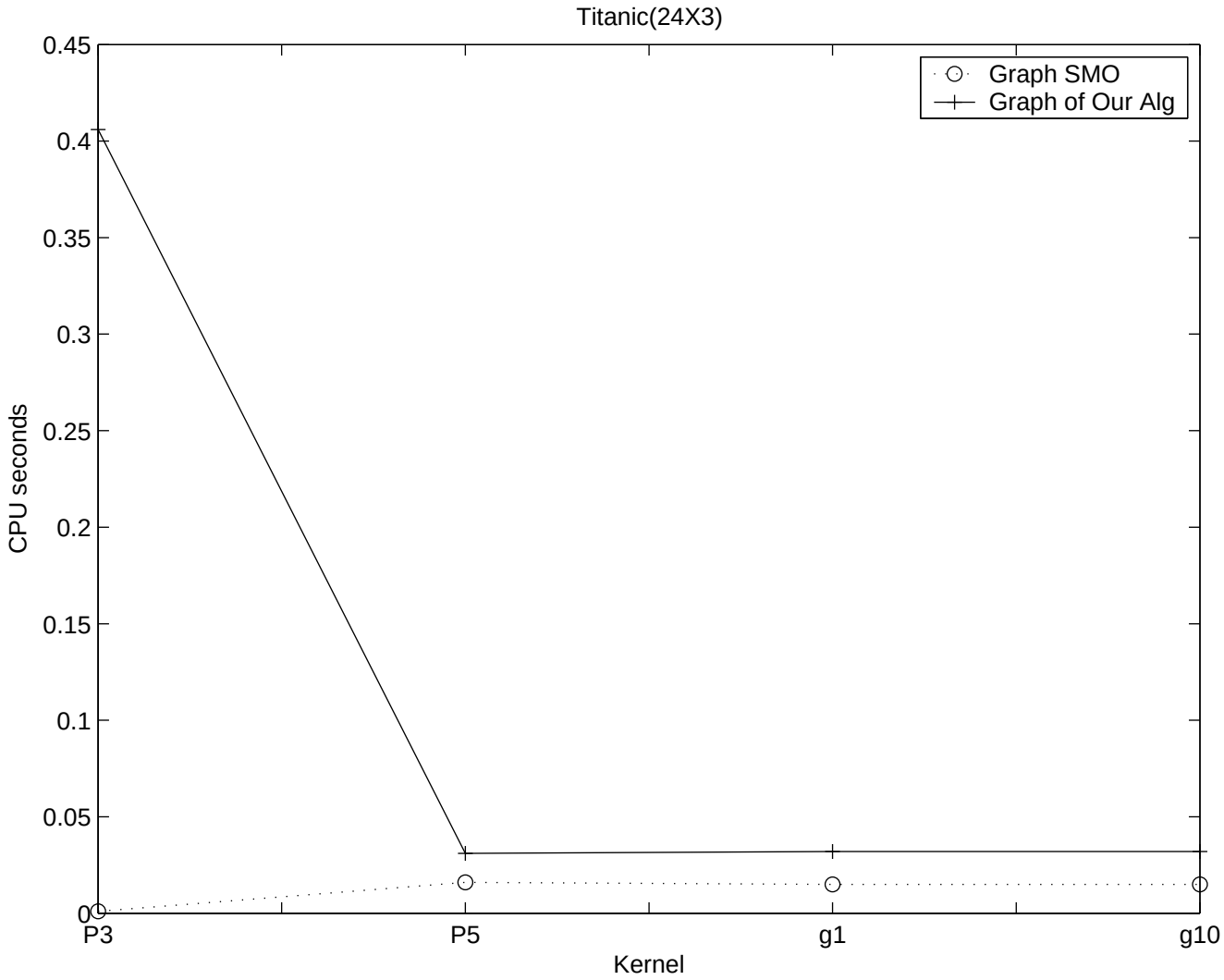

FIGURE 5.15. Titanic

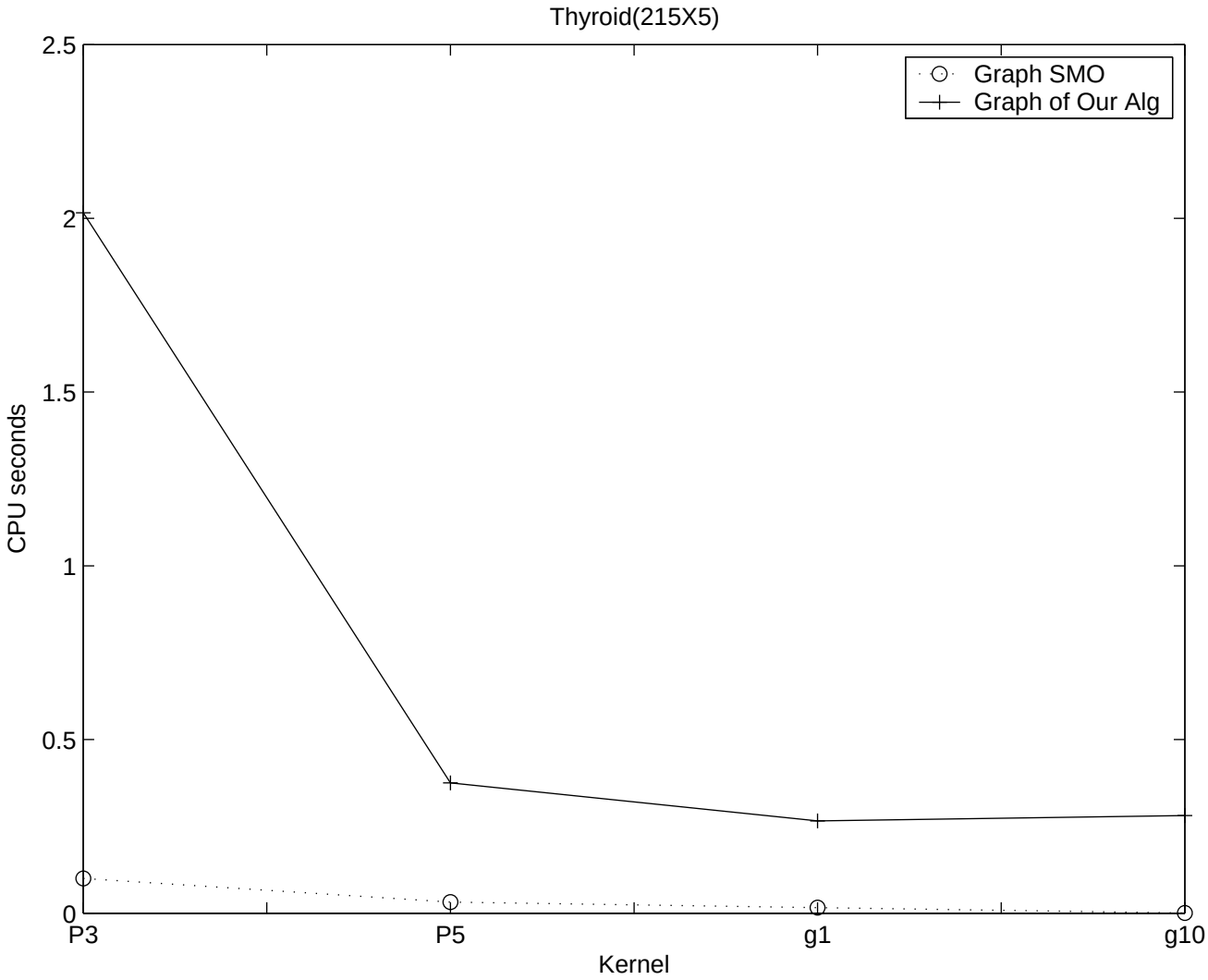

FIGURE 5.16. Thyroid

CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1. Conclusion

As shown previously in section 1 of chapter 5, the performance of SMO is better than Algorithm 2.1 with 100×100, 200×200, and 500×500. The performance with 1000×1000 or with higher dimensions, Algorithm 2.1 works better than SMO. Algorithm 2.1 performs better for higher dimensions. To show this, fix the number of points and increase the dimension from dimension 100 to 300. In the dimensions of both 100 and 300 with 7000 points respectively, SMO performs faster than Algorithm 2.1. However, with data set randomly generated by the MATLAB, CPU time for Algorithm is shorter than SMO in the dimensions of both 500 and 700 with 7000 points respectively. Therefore, in randomly generated linearly separable sets with a higher dimension, Algorithm 2.1 performs better than SMO. Although many application problems have a small number of support vectors, data sets that are generated by the MATLAB usually have a small number of support vectors. That is, if the data sets have a large number of support vectors, then Algorithm 2.1 may perform slower than SMO. Unfortunately, most of the real world data sets are in a low dimension format. Since the performance of SMO is better than Algorithm 2.1 with linearly separable sets, SMO performed better with those data sets. But as shown in section 2 in chapter 5, Algorithm 2.1 is still comparable with SMO. By the theorem 4 in chapter 2 and stopping method for the algorithm in section 2 in chapter 4, Algorithm 2.1 can find the solution for the NPP in a finite number of iterations.

6.2. Future Work

In Algorithm 2.1, two coefficients of u_k in representation at each iteration are updated, but note that, in fact, four or more coefficients of u_k can be updated. Instead of using

$u_{k+1} = u_k + \alpha'_k(\overline{x_k} - x'_k)$, $u_{k+1} = u_k + t_k\alpha_k(x_{\min,av} - x_{\max,av})$ can be used. In the new update, $0 \leq t_k \leq 1$, $\alpha_k = \min\{\alpha_{i'k}, \alpha_{2,i'k}\}$,

$x_{\min,av} = \frac{x_{i''k}+x_{2,i''k}}{2}$, $x_{\max,av} = \frac{x_{i'k}+x_{2,i'k}}{2}$

where $(x_{i''k}, u_k - v_k) = \min(x_i, u_k - v_k)$, $(x_{2,i''k}, u_k - v_k) = \min_{i \neq i''}(x_i, u_k - v_k)$, $(x_{i'k}, u_k - v_k) = \max_{a_i > 0}(x_i, u_k - v_k)$ and $(x_{2,i'k}, u_k - v_k) = \max_{a_i > 0 \text{ and} a_i \neq \alpha_{i'}}(x_i, u_k - v_k)$. With this expression, four coefficients in u_{k+1} are updated. The precise method of updating four coefficients at a time will be shown in Appendix B. When Algorithm 2.1 and the improved version of Algorithm 2.1 are compared, the number of iterations for the improved version of Algorithm 2.1 is always smaller than the one with Algorithm 2.1. By using the above method, CPU time for Algorithm 2.1 is improved. Moreover, improved Algorithm 2.1 could perform better than SMO in lower dimensions.

Appendix A

Improved MDM.

Define $u'_2 = \max_{\alpha_j>0,\alpha_j\neq\alpha'}(x_i, u)$ and $\overline{u}_2 = \min_{x_j\neq\overline{u}}(x_j, u)$

and $u'_{av} = \frac{u'+u'_2}{2}$ and $\overline{u}_{av} = \frac{\overline{u}+\overline{u}_2}{2}$

This section will show the improved MDM algorithm by using the average points u'_{av} and $\overline{u}_{av}$.

Improved MDM

Step 1 Choose $u_k \in U$ and find u'_k and $\overline{u}_k$

Step 2 Find u'_{2k} and $\overline{u}_{2k}$.

If u'_{2k} does not exist, then use the MDM.

If $(u'_{2k}, u'_{2k}) \leq (u_k, u_k)$, then use the MDM.

If $(\overline{u}_{2k}, \overline{u}_{2k}) > (u_k, u_k)$, then use the MDM.

If $\delta(u_k) > \Delta_k(u_k) = (u'_{avk} - \overline{u}_{avk}, u_k)$, then use the MDM.

Step 3 Otherwise, set $u_k(t) = u_k + t\alpha_k(\overline{u}_{avk} - u'_{avk})$ where $\alpha = \min\{\alpha', \alpha'_2\}$ and $t \in [0, 1]$

$\alpha = \min\{\alpha', \alpha'_2\}$ is used since each coefficient of α need to be greater than or equal to 0 in order to be in the convex hull.

Step 4 Set $u_{k+1} = u_k + t_k\alpha_k(\overline{u}_{avk} - u'_{avk})$ where $(u_k(t_k), u_k(t_k)) = \min_{t\in[0,1]}(u_k(t), u_k(t))$

Step 5 Iterate until $\Delta_k(u_k) = (u'_{avk} - \overline{u}_{avk}, u_k) \to 0$.

In the representation $u_k = \sum_{i=1}^{m} \alpha_{ik}x_i$, the improved MDM is updating 4 coefficient of the representation meanwhile the MDM is updating 2 coefficient of the representation. This makes the iteration faster. However, updating two many coefficient could make the iteration slow around the solution u^*. By **Step 4** $\|u_{k+1}\| \leq \|u_k\|$ is obtained.

The proof of the convergence

Assumption 1

(1) u'_{2k} exist

(2) $(u'_{2k}, u'_{2k}) > (u_k, u_k)$

(3) $(\overline{u}_{2k}, \overline{u}_{2k}) \leq (u_k, u_k)$

(4) $\delta(u_k) \leq \Delta_k(u_k)$

Define $\delta(u) = (u,u) - (\overline{u}_{av}, u)$. Then clearly $\delta(u) \geq 0$.By Corollay 2 to Lemma 2 in [1], If $\{u_k\}$, $u_k \in U$, $k = 0, 1, 2...$is a sequence of points such that $\|u_{k+1}\| \leq \|u_k\|$ and there is a subsequence $\left\{u_{k_j}\right\}$ such that $\delta(u_{k_j}) \underset{k_j \to \infty}{\to} 0$ then $u_k \to u^*$. Also, by the Assumption 1 for any $u = \sum_{i=1}^{m} \alpha_i x_i \in U$, $\delta(u) \leq \Delta(u)$. **Lemma A.1** $\lim_{k\to\infty} \alpha_k \Delta_k(u_k) = 0$ where $\alpha_k = \min\{\alpha', \alpha_2'\}$

(proof) Let $u_k(t) = u_k + 2t\alpha_k(\overline{u}_{avk} - u_k')$.

Then $(u_k(t), u_k(t)) = (u_k, u_k) - 4t\alpha_k\Delta_k(u_k) + (4\alpha^2)t^2 \left\|\overline{u}_{avk} - u_{av}'\right\|^2$.

Suppose $\lim_{k\to\infty} \alpha_k\Delta_k(u_k) \neq 0$. then there is a subsequnce u_{k_j} that $\alpha_{k_j}\Delta_{k_j} \geq \epsilon > 0$.

Then $\left(u_{k_j}(t), u_{k_j}(t)\right) \leq (u_{k_j}, u_{k_j}) - 4t\epsilon + 16t^2d^2$ where $d = \max \|x_l - x_p\|$.

$(u_{k_j}, u_{k_j}) - 4t\epsilon + 16t^2d^2$ is minimized at $\bar{t} = \min\{\frac{\varepsilon}{d^2}, 1\}$

If $\bar{t} = 1$, then $\varepsilon > d^2\bar{t}$ since $\frac{\varepsilon}{d^2} > \bar{t}$.

Then by $(u_k(t), u_k(t)) = (u_k, u_k) - 4t\alpha_k\Delta_k(u_k) + (4\alpha^2)t^2 \left\|\overline{u}_{avk} - u_{av}'\right\|^2$

$\left(u_{k_j}(t), u_{k_j}(t)\right) \leq (u_{k_j}, u_{k_j}) - 4\varepsilon + 4\varepsilon$

$= (u_{k_j}, u_{k_j})$

This is a contradiction.

If $\bar{t} = \frac{\varepsilon}{d^2}$ then $\left(u_{k_j}(t), u_{k_j}(t)\right) \leq (u_{k_j}, u_{k_j}) - 4\frac{\varepsilon}{d^2}\varepsilon + 4(\frac{\varepsilon}{d^2})^2d^2$

$= (u_{k_j}, u_{k_j})$

This is a contradiction. Thus $\lim_{k\to\infty} \alpha_k\Delta_k(u_k) = 0$ ■

A.2 $\underline{\lim}_{k\to\infty} \Delta_k(u_k) = 0$

(proof) Suppose $\underline{\lim}_{k\to\infty} \Delta_k(u_k) = \Delta^{\#}(u^{\#}) > 0$

Then for large $k > K$, $\Delta_k(u_k) \geq \frac{\Delta^{\#}(u^{\#})}{2}$.

By Lemma A.1 $\alpha_k \to 0$. Assume $(u_k(t), u_k(t))$ is minimized at $\bar{t}_k = \frac{\Delta_k(u_k)}{2\alpha_k\|\overline{u}_{avk} - u_{av}'\|^2}$.

Hence, for large $k > K \geq K_1$

$\bar{t}_k = 1$, thus $u_{k+1} = u_k + 2\alpha_k(\overline{u}_{avk} - u_{av}')$

Let $\left\{u_{k_j}\right\}$ be subsequence such that $\Delta_{k_j}(u_{k_j}) \to \Delta^{\#}(u^{\#})$

In the sequence $\left\{u_{k_j}\right\}$, some terms can be omitted if necessary. The sequence that satisfies the following 3 conditions hold can be found.

(1) $\alpha_{i,k_j} \to \alpha_i^{\#}$

$\Longrightarrow u_{k_j} \to u^{\#}$

(2) $\overline{u}_{av,k_j} = \overline{u}_{av} \in H^* = \{u_{av} \mid u_{av} = \frac{x_i + x_j}{2},\ for\ any\ i,j \in [1:m]\}$

(3) $u'_{av,k_j} = u' \in H^*$

Then $(u'_{av} - \overline{u}_{av}, u^{\#}) = \Delta^{\#}(u^{\#})$, and $\Delta^{\#}(u^{\#}) > 0$ is obtained.

Define $\rho^* = \min_i(x_i, u^{\#})$, $H_\rho = \{x_i \mid (x_i, u^{\#}) = \min(x_i, u^{\#})\}$

$H_1^* = \{u_{av} \in H^* \mid (u_{av}, u^{\#}) = \rho^*\}$, $H_2^* = H^* \setminus H_1^*$

$$\rho' = \min_{u_{av} \in H_2^*}(u_{av}, u^{\#})$$
$$\tau = \min\{\Delta^{\#}(u^{\#}), \rho' - \rho^*\}$$

Note $u_{av} \in H_2^*$,then $(u_{av}, u^{\#}) \geq \rho^* + \tau$.

Because if $\tau = \rho' - \rho^*$ then $\rho^* + \tau = \rho^* + \rho' - \rho^* = \rho'$

Since $u_{av} \in H_2^*$, $(u_{av}, u^{\#}) \geq \rho'$

Because if $\tau = \Delta^{\#}(u^{\#})$ then $\rho' - \rho^* \geq \Delta^{\#}(u^{\#}) \Longrightarrow \rho^* + \tau \leq \rho^* + \rho' - \rho^* = \rho'$

$\Longrightarrow (u_{av}, u^{\#}) \geq \rho' \geq \rho^* + \tau$.

Choose $\delta > 0$ such that whenever $\left\|u - u^{\#}\right\| < \delta$,

$$\max_i |(u_{av,i}, u) - (u_{av,i}, u^{\#})| < \frac{\tau}{4}$$

(Claim) Let k_j be such that $\left\|u_{k_j} - u^{\#}\right\| < \delta$ then $H_1(u_{k_j}) \subset H_1^*$

where $H_1(u_{k_j}) = \{u_{av} \in H^* \mid (u_{av}, u_{k_j}) = \min(u_{av}, u_{k_j})\}$

(proof of claim) Let $u_{av} \in H_1(u_{k_j})$.

$$\begin{aligned}(u_{av}, u^{\#}) &= (u_{av}, u_{k_j}) + (u_{av}, u^{\#} - u_{k_j}) \\ &= \min(u_{av}, u_{k_j}) + (u_{av}, u^{\#} - u_{k_j}) \\ &= \rho^* + (u_{av}, u^{\#} - u_{k_j}) + [\min(u_{av}, u_{k_j}) - \min(u_{av}, u^{\#})] \\ &\leq \rho^* + (u_{av}, u^{\#} - u_{k_j}) + \max|(u_{av}, u_{k_j}) - (u_{av}, u^{\#})| \leq \rho^* + \tfrac{\tau}{2}.\end{aligned}$$

Let $k_j > k$ be such that the following conditions hold

(a) $\Delta_{k''}(u_{k''}) \geq \Delta^{\#}(u^{\#}) - \frac{\tau}{4}$ for $k'' > k_j$

(b) $\left\|u_{k_j} - u^{\#}\right\| < \frac{\delta}{2}$

(c) $\sum_{l=1}^{s}\alpha_{k_j+l-1} < \frac{\delta}{2d}$ where $d = \max_{i,j}\left\|u_{av,i} - u_{av,j}\right\|$ and $\alpha_k \in \min\{\alpha'_k, \alpha'_{2,k}\}$

Then $u_{k_j+p} = u_{k_j} + \sum_{l=1}^{s}\alpha_{k_j+l-1}(\overline{u}_{av,k_j+l-1} - u'_{av,k_j+l-1})$ is obtained.

$$\left\|u_{k_j+p} - u_{k_j}\right\| \le d\sum_{l=1}^{s}\alpha_{k_j+l-1} < \frac{\delta}{2}$$

$$\Longrightarrow \left\| u_{k_j+p} - u^{\#}\right\| \le \left\|u_{k_j+p} - u_{k_j}\right\| + \left\|u_{k_j} - u^{\#}\right\| < \delta$$

By the claim, $H_1(u_{k_j}) \subset H_1^*$

Now, for all $u_{av} \in H_1^*$,

$$\begin{aligned}\left(u_{av}, u_{av,k_j+p}\right) &= (u_{av}, u^{\#}) + (u_{av}, u_{av,k_j+p} - u^{\#}) \\ &= (u_{av}, u^{\#}) + (u_{av}, u_{av,k_j+p}) - (u_{av}, u^{\#}) \\ &\le \rho^* + \tfrac{\tau}{4} \\ &\le \Delta^{\#}(u^{\#}) - \tau + \rho^* + \tfrac{\tau}{4} \\ &= \Delta^{\#}(u^{\#}) + \rho^* - \tfrac{3\tau}{4}\end{aligned}$$

For large k, $k_j > k$,

$u_{k_j+1} = u_{k_j} + \alpha_{k_j}(\overline{u}_{av,k_j} - u'_{av,k_j})$.

Then for any $p \ge 1$, u_{k_j+p+1} involve at least one of the vector from $u'_{av,k_j+p} = \frac{u'_{av,k_j+p} + u'_{av2,k_j+p}}{2}$ with zero coefficent. Then for any $q \ge 1$, u_{k_j+p+q} does not involve the vectors u'_{av,k_j+p+l}, for $l < q$ because $\overline{u}_{k_j+p+l}$ and $\overline{u}_{2,k_j+p+l}$ are in H_ρ. Since X has a finite number of vectors, H^*, H_1^* and H_2^* have finite number of vectors. This means for large r, $u_{k_j+p+r+1} = u_{k_j+p+r} + \alpha_{k_j}(\overline{u}_{av,k_j+p+r} - u'_{av,k_j+p+r})$ where $\overline{u}_{av,k_j+p+r}, u'_{av,k_j+p+r} \in H_1^*$.

Now $(u_{k_j+p+r}, u'_{av,k_j+p+r}) \le \Delta^{\#}(u^{\#}) + \rho^* - \frac{3\tau}{4}$ is obtained and $(u_{k_j+p+r}, \overline{u}_{av,k_j+p+r}) \ge \rho^* - \frac{\tau}{4}$

This implies $\Delta_{k_j+p+r}(u_{k_j+p+r}) = (u_{k_j+p+r}, u'_{av,k_j+p+r}) - (u_{k_j+p+r}, \overline{u}_{av,k_j+p+r}) \le \Delta^{\#}(u^{\#}) - \frac{\tau}{2}$

This contradicting (a). This proves the lemma. ■

Theorem A.1 Let $\{u_k\}$ be a sequence obtained from Algorithm 2. Then $u_k \to u^*$.

(proof) By the lemma A.2, $\underline{\lim}_{k\to\infty}\Delta_k(u_k) = 0$ is obtained. By the condition in improved MDM, $\delta(u_k) \le \Delta(u_k)$. Then by Corollay 2 to Lemma 2 in [1], $u_k \to u^*$.■

Appendix B

Improved version of algorithm 2.1

Algorithm 2.1 can be combined with appendix A. so each iteration, 4 vectors are updated meanwhile algorithm 2.1 update only update 2 vector at each iteration.

The following algorithm is improved version of algorithm 2.1.

Improved 2.1

1. Choose $u_k \in U$ and $v_k \in V$

2. Find $x_{i''k}$, $x_{2,i''k}$, $x_{i'k}$ and $x_{2,i'k}$

where $(x_{i''k}, u_k - v_k) = \min(x_{ik}, u_k - v_k)$,

$$(x_{2,i''k}, u_k - v_k) = \min_{x_{2,i''k} \neq x_{ik}} (x_{ik}, u_k - v_k),$$

$$(x_{i'k}\ , u_k - v_k) = \max_{a_{ik}>0} (x_{i'k}\ , u_k - v_k)$$

and $(x_{2,i'k}, u_k - v_k) = \max_{a_{ik}>0 \text{ and } a_{ik} \neq \alpha_{i'k}} (x_{i'k}\ , u_k - v_k)$

If $x_{2,i''k}$ does not exist, then go to step 3 in the algorithm 2.1

if $x_{2,i'k}$ does not exist, then go to step 3 in the algorithm 2.1

3. $u_{k+1} = u_k + t_k \alpha_k (x_{\min,av} - x_{\max,av})$

where $0 \leq t_k \leq 1$, $\alpha_k = \min\{\alpha_{i'k}, \alpha_{2,i'k}\}$,

$x_{\min,av} = \frac{x_{i''k} + x_{2,i''k}}{2}$, $x_{\max,av} = \frac{x_{i'k} + x_{2,i'k}}{2}$

and t_k is defined by

$(u_k(t_k), u_k(t_k)) = \min_{t \in [0,1]} (u_k(t), u_k(t))$ where $u_k(t) = u_k + t\alpha_k (x_{\min,av} - x_{\max,av})$

4. Find $y_{j''k}$, $y_{2,j''k}$, $y_{j'k}$ and $y_{2,j'k}$

where $(y_{j''k}, v_k - u_k) = \min(y_{jk}, v_k - u_k)$,

$$(y_{2,j''k}, v_k - u_k) = \min_{y_{2,j''k} \neq y_{jk}} (y_{jk}, v_k - u_k),$$

$$(y_{j'k}\ , v_k - u_k) = \max_{\beta_{jk}>0} (y_{j'k}\ , v_k - u_k)$$

and $(y_{2,j'k}, v_k - u_k) = \max_{\beta_{jk}>0 \text{ and } \beta_{jk} \neq \beta_{j'k}} (y_{j'k}\ , v_k - u_k)$

If $y_{2,j''k}$ does not exist, then go to step 5 in the algorithm 2.1

if $y_{2,j'k}$ does not exist, then go to step 5 in the algorithm 2.1

5. $v_{k+1} = v_k + t'_k (y_{\min,av} - y_{\max,av})$

where $0 \leq t'_k \leq 1$, $\beta_k = \min\{\beta_{j'k}, \beta_{2,j'k}\}$,

$y_{\min,av} = \frac{y_{i''k} + y_{2,i''k}}{2}$, $y_{\max,av} = \frac{y_{i'k} + y_{2,i'k}}{2}$

and t'_k is defined by

$$(v_k(t'_k), v_k(t'_k)) = \min_{t' \in [0,1]} (v_k(t'), v_k(t')) \text{ where } v_k(t') = v_k + t'\beta_k(y_{\min,av} - y_{\max,av})$$

BIBLIOGRAPHY

[1] B. F. Mitchell, V. F. Dem'yanov, and V. N. Malozemov, *Finding the point of a polyhedron closest to the origin*, SIAM J. Contr., vol. 12, pp. 19-26, 1974.

[2] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, *A fast iterative nearest point algorithm for support vector machine classifier design*, IEEE Trans. Neural Networks, vol. 11, pp. 124-136, Jan. 2000.

[3] T. T. Frie, N. Cristianini, and C. Campbell, *The kernel adatron algorithm: A fast and simple learning procedure for support vector machines*, in Proc. 15th Int. Conf. Machine Learning, San Mateo, CA, 1998.

[4] C. J. C. Burges, *A tutorial on support vector machines for pattern recognition*, Data Mining and Knowledge Discovery, vol. 2, no. 2, 1998.

[5] E. G. Gilbert, *Minimizing the quadratic form on a convex set*, SIAM J. Contr., vol. 4, pp. 61-79, 1966.

[6] V. Vapnik, *The Nature of Statistical Learning Theory*, New York, Springer-Verlag, 1995.

[7] Vapnik, *Statistical Learning Theory*, Springer, N.Y., 1998.

[8] P.Wolfe, *Finding the nearest point in a polytope*, Math. Programming, vol. 11, pp. 128-149, 1976.

[9] Jorge Nocedal, Stephen J.Wright,*Numerical Optimization*, Springer Series in Operations Research 1999.

[10] N. Cristanini and John Shawe-Taylor ,An Introduction to Support Vecror Machines. Cambridge University Press, 2000.

[11] J.C. Platt, *Fast Training of Support Vector Machines Using Sequential Minimal Optimization*, Advances in Kernel Methods: Support Vector Machines, B. Scholkopf, C.J.C. Burges, and A. Smola, eds., pp. 185-208, Cambridge, Mass.: MIT Press, Dec. 1998.

[12] E. Osuna, R. Freund, and F. Girosi, *Training support vector machines: An application to face detection*, in IEEE Proc. CVPR'97, New York, 1997, pp. 130-136.

[13] V.F. Dem'yanov, and V.N. Malozemov. *Introduction to Minimax*, Dover Publications, New York, 1974.

[14] D Kaown and J Liu, *Geometric methods for support vector machines*, Proceedings of the 8th National Conference of Operations Research Society of China, China, 2006, pp. 723-728.

[15] Marti A. Hearst, *Support Vector Machine*, IEEE Intelligent Systems, v.13 n.4, p 18-28, July 1998.

[16] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. *A practical guide to support vector classification*, Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, 2003. http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[17] Andrew W. Moore, *Support Vector Machine*, Lecture note, School of Computer-Science Carnegie Mellon University.

[18] Kristin P Bennett, Erin J Bredensteiner, *Duality and Geometry in SVM Classifiers Proceedings of the Seventeenth International Conference on Machine Learning*, p 57-64, June 29-July 02, 2000.

[19] E. Osuna, R. Freund, and F. Girosi, *An Improved Training Algorithm for Support Vector Machines*, in Proc. IEEE Neural Netw. for Sign. Proc. NNSP97, 1997, pp. 276-285.

[20] J.P Lewis, *A Short SVM(Support Vector Machine) Tutorial*, CGIT Lab / IMSC, 2004.

[21] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, *A fast procedure for computing the distance between complex objects in three-dimensional space*, IEEE Trans. Robot. Automat., vol. 4, pp. 193-203, Apr. 1988.

[22] E. G. Gilbert and C-K Foo, *Computing the distance between general convex objects in three-dimensional space*, IEEE Trans. Robot. Automat., vol. 6, pp. 53-61, Feb. 1990.

[23] Tong Weng, http://seesar.lbl.gov/anag/staff/wen/link.html.

[24] Tong Wen, Alan Edelman, and David Gorsich, *A fast projected conjugate gradient algorithm for training support vector machines*, Contemporary mathematics, theory and applications., pp 245-263, 2001.

[25] Gene H. Golub, Charles F. Van Loan, *Matrix Computation*, The Johns Hopkins University Press, Baltimore and London, 1996.

[26] Gunnar Raetsch, http://theoval.sys.uea.ac.uk/matlab/default.html, Gunnar Raetsch's Benchmark Datasets.

[27] V.F. Demyanov, and V.N. Malozemov, *Introduction to Minimax*, Dover Publications, New York, 1974.