A COMPARISON OF THE TRIANGLE ALGORITHM AND SEQUENTIAL MINIMAL OPTIMIZATION ALGORITHM FOR SOLVING THE HARD MARGIN PROBLEM.

BY MAYANK GUPTA

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Computer Science

Written under the direction of

Bahman Kalantari

and approved by

New Brunswick, New Jersey

May, 2016

ABSTRACT OF THE THESIS

A COMPARISON OF THE TRIANGLE ALGORITHM AND SEQUENTIAL MINIMAL OPTIMIZATION ALGORITHM FOR SOLVING THE HARD MARGIN PROBLEM.

by MAYANK GUPTA

Thesis Director: Bahman Kalantari

In this article we consider the problem of testing, for two finite sets of points in the Euclidean space, if their convex hulls are disjoint and computing an optimal supporting hyperplane if so. This is a fundamental problem of classification in machine learning known as the *hard-margin SVM*. The problem can be formulated as a quadratic programming problem. The *SMO algorithm* [1] is the current state of art algorithm for solving it, but it does not answer the question of separability. An alternative to solving both problems is the *triangle algorithm* [2], a geometrically inspired algorithm, initially described for the *convex hull membership* problem [3], a fundamental problem in linear programming. First, we describe the experimental performance of the triangle algorithm for testing the intersection of two convex hulls. Next, we compare the performance of triangle algorithm with SMO for finding the optimal supporting hyperplane. Based on experimental results ranging up to 5000 points in each set in dimensions up to 10000, the triangle algorithm outperforms SMO.

Acknowledgements

I would like to express my gratitude towards my adviser, Prof. Bahman Kalantari, who patiently supported me throughout my thesis. I appreciate his vast knowledge and skills in my research area. A special thanks to my defense committee, Prof. William Steiger, and Prof. Pranjal Awasthi for their interest in my work.
I would also like to acknowledge the Department of Computer Science at Rutgers. I really look up to all the faculty here. Finally, I would also like to thank the staff at Rutgers, specially the health services, for taking care of students throughout the year.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Given a pair of finite sets, determining if they are linearly separable and if so, finding a separating hyperplane is a problem of classification dealt with in statistics and machine learning. In two-class classification, we wish to estimate a function $f : \mathbb{R}^m \to \{\pm 1\}$ using the input-output data

$$(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^m \times \{\pm 1\} \tag{1.1}$$

Given some new data point x, we use $f(x)$ to classify its label. According VC-theory minimizing the error on the test set depends not only on minimizing the empirical risk but also on the capacity of the function class. This has led to the development of the class of functions whose capacity can be computed.

Vapnik and Chervonenkis (1964) and Vapnik and Lerner (1969) considered the class of hyperplanes

$$w^T x - b = 0, \quad w \in \mathbb{R}^m, b \in \mathbb{R} \tag{1.2}$$

corresponding to the decision function

$$f(x) = sign(w^T x - b), \tag{1.3}$$

and proposed a learning algorithm for separable problems, to construct the function f from empirical data. This optimal margin classifier [4] for the linearly separable case is based upon the minimization of the number of classification errors by placing optimal boundaries between classes. This has also led to the tremendous success of *support vector machines* (SVM) in classification tasks.

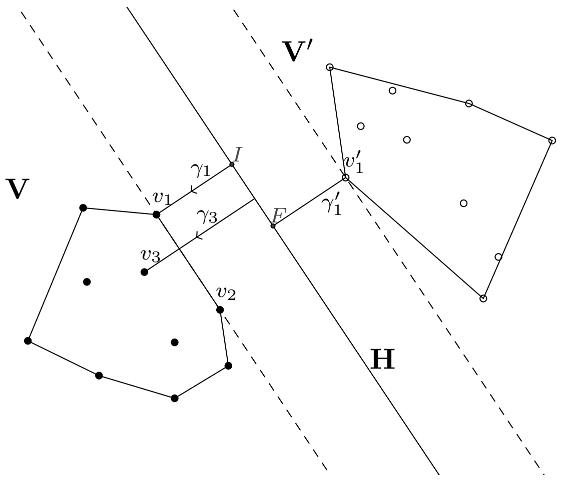


Figure 1.1: Binary classification

Figure 1.1 is an example of a binary or two-class classification problem. Here $V = \{v_1, \ldots, v_n\}$ and $V' = \{v'_1, \ldots, v'_{n'}\}$ are samples from the two classes. The convex hulls, $K = conv(V)$ and $K' = conv(V')$, for both classes are drawn. The class membership y_i is 1 if $x \in V'$ and -1 if $x \in V$.

According to the classical separating hyperplane theorem, K and K' are disjoint if and only if they can be separated by a hyperplane, i.e. if there exists $w \in \mathbb{R}^m$ and $b \in \mathbb{R}$ such that

$$\begin{aligned} w^T x < b, &\quad \forall x \in K \\ w^T x > b, &\quad \forall x \in K' \end{aligned} \tag{1.4}$$

The hyperplane

$$W = \{x \in \mathbb{R}^m : \quad w^T x = b\} \tag{1.5}$$

separates K and K'.

The optimal hyperplane H is orthogonal to the shortest line connecting the convex hulls, K and K', of the two classes, and intersects this line segment half-way. Among all the hyperplanes separating the data, H is the unique one yielding the maximum margin of separation between K and K'. The distance between a test point and the optimal hyperplane also provides an estimate on the confidence in the classification.

In Figure 1.1, the minimum distance γ_1 of K to the hyperplane is given by the line segment Iv_1.

The point v_1 represents $x^{(i)}$, P can be represented as

$$I = v_1 - \gamma_1 . \frac{w}{||w||} \tag{1.6}$$

Since I lies on the hyperplane w

$$\begin{gathered} w^T I - b = 0 \\ w^T (x^{(i)} - \gamma_i . \frac{w}{||w||}) - b = 0 \end{gathered} \tag{1.7}$$

Solving for γ_i lends

$$\gamma_i = y^{(i)} (\frac{w}{||w||})^T x^{(i)} - \frac{b}{||w||}) \tag{1.8}$$

To obtain a unique solution, H, to the (1.8), we set $||w|| = 1$. By definition the maximally separating hyperplane will be such that

$$\gamma = min\{\gamma_i : i = 1, ..m\} \tag{1.9}$$

The solution to this relies on the fact that the minimum of the Euclidean distance function $d(x, x')$ is attained and is positive

$$min\{d(x, x') : x \in K, x' \in K'\} > 0 \tag{1.10}$$

To find the maximum of the minimum margins, (1.9) can be written as

$$\begin{aligned} & \underset{\gamma, w, b}{max} \quad \gamma \\ s.t. \quad & y_i (w^T x^{(i)} + b) \geq \gamma, \quad i = 1, \ldots, m \\ & ||w|| = 1 \end{aligned} \tag{1.11}$$

Our goal in this article is to consider the hard margin problem and study the performance of two distinct algorithms; the *Triangle Algorithm* and the *Sequential Minimal Optimization(SMO)* algorithm.

The triangle algorithm is a geometrically inspired algorithm that works in two phases. In the first phase, given (K, K') described above, it determines whether they are separable or not. The first phase begins with a pair of iterates $(p, p') \in K \times K'$ .In each iteration it either moves p close to p' using a *p-pivot* to a new point $p \in K$, or it moves p' closer to p using a *p'-pivot*. A p-pivot is a point $v \in V$ that lies in the *Voronoi region*(1.12 of p', $Vor(p')$. Similarly for p'-pivot. If the triangle algorithm fails to find any such pivot, it implies that $K \in Vor(p)$ and $K \in Vor(p')$, returning a *witness pair* $(p, p') \in K \times K'$, the perpendicular bisector of which is a hyperplane that separates (K, K'). In the second phase it gradually reduces the gap between p and p' to compute the optimal hyperplane.

$$Vor(p') = \{x \in \mathbb{R}^m :, d(x, p') < d(x, p)\} \tag{1.12}$$

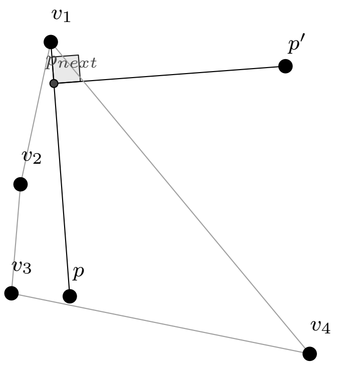


Figure 1.2: Triangle Algorithm: Moving p closer to p'

Figure 1.2 shows p moving to the point closest to p' along the segment pv_1, such that $d(p_{next}, p') < d(p, p')$. The convex hull $K = conv(\{v_1, v_2, v_3, v_4\})$ is also drawn.

In the next phase the triangle algorithm starts with a witness pair (p, p') and its orthogonal separating hyperplane $w^T x = b$ to calculate the optimal hyperplane H. The distance, $d(p, p')$, is an upper bound to the optimal distance. It computes the lower bound to the distance and reduces $d(p, p')$ by moving p to p_{final}. It does so by finding a point nearest to p' on the segment pv_{ext}, where v_{ext} is an *extreme vertex* of K defined as

$$v_{ext} = max\{w^T v_i : \quad v_i \in V\} \tag{1.13}$$

It terminates when the iterates are close to each other within a prescribed tolerance. Figure 1.3 shows a special case of a set and a point for triangle algorithm II. The input parameters are $(p, p'), V$ and the output is (p_{final}, p'). Here v_1 and v_4 act as the extreme points.

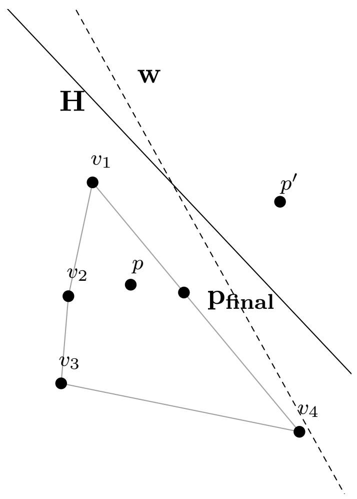


Figure 1.3: Triangle Algorithm II

The SMO algorithm however, is a quadratic programming method for solving the problem posed by (1.9). Setting the derivative of its Lagrangian to zero (see section 3 for details), we get

$$w = \sum_{i=1}^{n} \alpha_i y^{(i)} x^{(i)} \tag{1.14}$$

$$\sum_{i=1}^{n} \alpha_i y^{(i)} = 0 \tag{1.15}$$

as a partial set of equations. The key idea of SMO is to fix all α_i's except for a pair(α_i, α_j) of them. Having selected (α_i, α_j), SMO then reoptimizes w with respect to them.

The SMO algorithm, as we shall see in section 3, does not answer the question of separability of the two sets.

For the intended comparison we implemented both the algorithms, triangle and SMO, in matlab. For our experiments we generated two unit balls with random mean and placed them at some distance apart, to analyze the results of both algorithms. We tested and analyzed the results for both algorithms for up to 1000 dimensions with up to 2000 points in each set.

The remainder of this article is organized as follows. In section 2 we describe the *distance duality*, triangle algorithm and its complexity. In section 3 we describe the *langrange duality* and SMO algorithm. In section 4, we describe the performance of the triangle algorithm for testing the intersection or separation of two convex hulls. In section 5, we compare the performance of the Triangle algorithm with SMO for finding the optimal hyperplane. In section 6, we discuss some ideas for an efficient implementation of the triangle algorithm and future work.

Chapter 2

Triangle Algorithm

Given a finite set $V = \{v_1, \ldots, v_n\} \subset \mathbb{R}^m$, and a distinguished point $p \in \mathbb{R}^m$, the *convex hull membership problem*(or *convex hull decision problem*), is to test if $p \in conv(V)$, or convex hull of V. Given a desired tolerance $\epsilon \in (0, 1)$, we call a point $p_\epsilon \in conv(V)$ an ϵ-approximate solution if $d(p_\epsilon, p) \leq \epsilon R$, where $R = max\{d(p, v_i) : \quad i = 1..n\}$.

A recent algorithm for the convex hull membership problem is the triangle algorithm [3]. It can either compute an ϵ-approximate solution, or when $p \notin conv(V)$ a separating hyperplane and a point that approximates the distance from p to $conv(V)$ to within a factor of 2. Based on preliminary experiments, the triangle algorithm performs quite well on reasonably large size problem, see [5].

Here we review the terminology and some results from [3] [2] for the case of two finite convex hulls.

2.1 Definitions

The Euclidean distance is denoted by $d(\cdot, \cdot)$. Let $V = \{v_1, \ldots, v_n\}$ and $V' = \{v'_1, \ldots, v'_{n'}\}$, $K = conv(V)$ and $K' = conv(V')$. Assume we are given $p_0 \in K$, $p'_0 \in K'$. Let

$$\delta_* = d(K, K') = \min\{d(p, p') : p \in K, p' \in K'\}. \tag{2.1}$$

It is easy to prove that $\delta_* = 0$ if and only if $K \cap K' \neq \emptyset$. Suppose $\delta_* = 0$. We say a pair $(p, p') \in K \times K'$ is an *ϵ-approximation solution to the intersection problem* if

$$d(p, p') \leq \epsilon d(p, v), \quad \text{for some} \quad v \in K, \quad \text{or} \quad d(p, p') \leq \epsilon d(p', v'), \quad \text{for some} \quad v' \in K'. \tag{2.2}$$

Triangle Algorithm I computes an ϵ-approximate solution to the intersection problem when $\delta_* = 0$, or a pair of separating hyperplane when $\delta_* > 0$. In each iteration it begins with a pair of iterates $p_i \in K$, $p'_i \in K'$ and searches for a p'-pivot $v \in V$ and p-pivot $v' \in V'$ along with the optimal pair of supporting hyperplanes defined below.

Given a pair $(p, p') \in K \times K'$ (see Figure 2.1), we say $v \in K$ is a *p'-pivot* for p if

$$d(p, v) \geq d(p', v). \tag{2.3}$$

We say $v' \in K'$ is a *p-pivot* for p' if

$$d(p', v') \geq d(p, v'). \tag{2.4}$$

Consider the Voronoi diagram of the set $\{p, p'\}$ and the corresponding Voronoi cells

$$Vor(p) = \{x : d(x, p) < d(x, p')\}, \quad Vor(p') = \{x : d(x, p') < d(x, p)\}. \tag{2.5}$$

If $H = \{x : h^T x = a\}$ is the orthogonal bisecting hyperplane of the line pp', it intersects K if and only if there exists $v \in K$ that is a p'-pivot for p, and H intersects K' if and only if there exists $v' \in K'$ that is a p-pivot for p'. Given $(p, p') \in K \times K'$, we say it is a *witness pair* if the orthogonal bisecting hyperplane of the line segment pp' separates K and K'. Figure 1.3 shows an example of a witness-pair (p, p') with w as their orthogonal bisecting hyperplane. In Figure 2.1, the point v and v' are pivots for p' and p, respectively. The four points p, p', v, v' need not be coplanar.

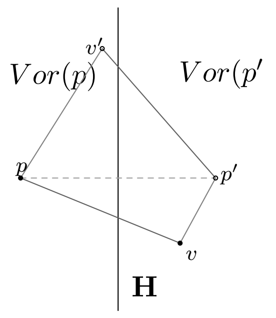


Figure 2.1: v is p'-pivot for p (left); v' is p-pivot for p'

Each iteration of Triangle Algorithm I requires computing for a given pair $(p, p') \in K \times K'$ a p'-pivot v for p, or a p-pivot v' for p'. By squaring (2.3) and (2.4), these are respectively equivalent to checking if

$$2v^T(p' - p) \geq \|p'\|^2 - \|p\|^2, \quad 2v'^T(p - p') \geq \|p\|^2 - \|p'\|^2. \tag{2.6}$$

From the above it follows that the existence and computation of a pivot can be carried out by solving the following

$$\max\{(p'-p)^T v: \quad v \in V\}, \quad \max\{(p-p')^T v': \quad v' \in V'\}. \tag{2.7}$$

Triangle Algorithm II begins with a witness pair $(p,p') \in K \times K'$, then it computes an ϵ-approximate solution to the distance problem. Since (p,p') is a witness pair, there exists no p'-pivot for p, or a p-pivot for p'.

We say a witness pair $(p,p') \in K \times K'$ is an *ϵ-approximation solution to the distance problem* (or *ϵ-approximation solution to δ_**) if

$$d(p,p') - \delta_* \leq \epsilon d(p,p'). \tag{2.8}$$

A pair of parallel hyperplanes (H,H') *supports* (K,K'), if H contains a boundary point of K, H' contains a boundary point of K', $K \subset H_+$, $K' \subset H'_+$, where H_+, H'_+ are disjoint halfspaces corresponding to H, H'. Figure 1.1 shows an example of hyperplanes supported by boundary points. A witness pair $(p,p') \in K \times K'$ is an *ϵ-approximate solution to the supporting hyperplanes problem* if

$$d(p,p') - \delta_* \leq \epsilon d(p,p') \tag{2.9}$$

and there exists a pair of parallel supporting hyperplanes (H,H') orthogonal to the line segment pp' such that the distance between them satisfies

$$\delta_* - d(H,H') \leq \epsilon d(p,p') \tag{2.10}$$

If (p,p') is not already an ϵ-approximate solution to δ_*, the algorithm makes use of a *weak-pivot*. Given a witness pair $(p,p') \in K \times K'$, suppose that H is the orthogonal bisecting hyperplane of the line segment pp'. We shall say $v \in V$ is a *weak p'-pivot* for p if

$$d(p,H) > d(v,H) \tag{2.11}$$

(i.e. if H_v is the hyperplane parallel to H passing through v, it separates p from p', see Figure 2.2a). Similarly, we shall say $v' \in V'$ is a *weak p-pivot* for p' if

$$d(p',H) > d(v',H). \tag{2.12}$$

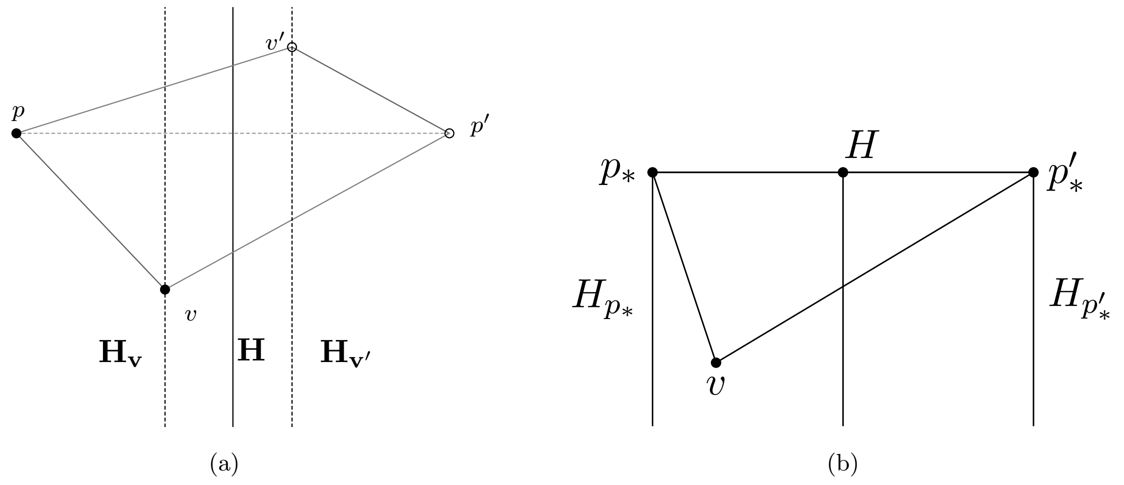


Figure 2.2: Hyperplanes depicting the lower and upper bounds on the optimal distance

In an iteration of Triangle Algorithm II a given pair $(p_k, p'_k) \in K \times K'$ may or many not be a witness pair. The algorithm searches for a weak-pivot or a pivot in order to reduce the current gap $\delta_k = d(p_k, p'_k)$ until ϵ-approximate solutions to both the distance and supporting hyperplanes problems are reached.

The correctness of the triangle algorithm relies on the following distance duality.

Theorem 1. (Distance Duality [2]) *$K \cap K' \neq \emptyset$ if and only if for each $(p, p') \in K \times K'$, either there exists $v \in S$ such that $d(p, v) \geq d(p', v)$, or there exists $v' \in S'$ such that $d(p', v') \geq d(p, v')$.*
$\square$

Suppose $d(K, K') = d(p_*, p'_*)$, where $(p_*, p'_*) \in K \times K$. Then if H_{p_*} and $H_{p'_*}$ are orthogonal hyperplanes to the line segment $p_*p'_*$ at p_* and p'_* respectively, they are optimal supporting hyperplanes to K and K', respectively. In other words, $d(K, K') = d(p_*, p'_*) = d(H_{p_*}, H_{p'_*})$.

Complexity of computing ϵ-approximation solution	Intersection $K \cap K' \neq \emptyset$	Separation $K \cap K' = \emptyset$	Distance and Support $\delta_* = d(K, K')$
$K = conv(\{v_1, \ldots, v_n\})$ complexity w. preprocessing $K' = \{p'\}$	$O\big(mn\frac{1}{\epsilon^2}\big)$ $O\big((m+n)\frac{1}{\epsilon^2}\big)$	$O\big(mn\big(\frac{\rho_*}{\delta_*}\big)^2\big)$ $O\big((m+n)\big(\frac{\rho_*}{\delta_*}\big)^2\big)$	$O\big(mn\big(\frac{\rho_*}{\delta_*\epsilon}\big)^2\big)$ $O\big((m+n)\big(\frac{\rho_*}{\delta_*\epsilon}\big)^2\big)$
$K = conv(\{v_1, \ldots, v_n\})$ $N = \max\{n, n'\}$ complexity w. preprocessing $K' = conv(\{v'_1, \ldots, v'_{n'}\})$	$O\big(mN\frac{1}{\epsilon^2}\big)$ $O\big((m+N)\frac{1}{\epsilon^2}\big)$	$O\big(mN\big(\frac{\rho_*}{\delta_*}\big)^2\big)$ $O\big((m+N)\big(\frac{\rho_*}{\delta_*}\big)^2\big)$	$O\big(mN\big(\frac{\rho_*}{\delta_*\epsilon}\big)^2 \ln\frac{\rho_*}{\delta_*}\big)$ $O\big((m+N)\big(\frac{\rho_*}{\delta_*\epsilon}\big)^2 \ln\frac{\rho_*}{\delta_*}\big)$

Table 2.1: The complexities of Triangle Algorithms I and II. ρ_* is maximum of diameters of K and K'.

Table 2.1 summarizes the complexity of Triangle Algorithms I and II in solving the optimal hyperplane problem.(See [2])

2.2 Triangle Algorithm I

In this section we describe the details of *Triangle Algorithm I.* This is a generalization of the original Triangle Algorithm in the case when $V = \{v_1, \ldots, v_n\}$ and $V' = \{v'_1, \ldots, v'_{n'}\}$.

The algorithm searches for a triangle $\triangle pp'v'$ where $v' \in V'$, such that $d(p', v') \geq d(p, v')$; or a triangle $\triangle pp'v$ where $v \in V$, such that $d(p, v) \geq d(p', v)$. Given that such triangle exists, it uses v or v' as a pivot to bring p, p' in current iterate $(p, p') = (p_k, p'_k) \in K \times K'$ closer to each other by generating either a new iterate $p_{k+1} \in K$, or new iterate $p'_{k+1} \in K'$ such that if we denote the new iterate by (p_{k+1}, p'_{k+1}), $d(p_{k+1}, p'_{k+1}) < d(p_k, p'_k)$.

Given three points $x, y, z \in \mathbb{R}^m$ such that $d(y, z) \geq d(x, z)$. Let $nearest(x; yz)$ be the nearest point to x on the line segment joining y to z.

We have Given three points $x, y, z \in \mathbb{R}^m$, let the *step-size* be

$$\alpha = \frac{(x-y)^T(z-y)}{d^2(y, z)}. \tag{2.13}$$

Then

$$nearest(x;yz) = \begin{cases} (1-\alpha)y + \alpha z, & \text{if } \alpha \in [0,1]; \\ z, & \text{otherwise.} \end{cases} \tag{2.14}$$

Here we describe Triangle Algorithm I for testing if two finite convex hulls K, K' intersect. It computes a pair $(p,p') \in K \times K'$ such that either $d(p,p')$ is to within a prescribed tolerance, or it is a witness pair. It assumes we are given points $(p_0, p'_0) \in K \times K'$ and $\epsilon \in (0,1)$.

Triangle Algorithm I ($(p_0, p'_0) \in K \times K'$, $\epsilon \in (0,1)$)

- **Step 0.** Set $p = v = p_0$, $p' = v' = p'_0$.
- **Step 1.** If $d(p,p') \leq \epsilon d(p,v)$, or $d(p,p') \leq \epsilon d(p',v')$, stop.
- **Step 2.** Test if there exists $v \in V$ that is a p-pivot for p', i.e.

$$2v^T(p'-p) \geq \|p'\|^2 - \|p\|^2 \tag{2.15}$$

  If such pivot exists, set $p' \leftarrow nearest(p';pv)$, and go to Step 1.
- **Step 3.** Test if there exists $v' \in V'$ that is a p'-pivot for p, i.e.

$$2v'^T(p-p') \geq \|p\|^2 - \|p'\|^2 \tag{2.16}$$

  If such pivot exists, set $p \leftarrow nearest(p;p'v')$, and go to Step 1.
- **Step 4.** Output (p,p') as a witness pair, stop ($K \cap K' = \emptyset$).

2.3 Triangle Algorithm II

Triangle Algorithm II begins with a witness pair $(p,p') \in K \times K'$, then it computes an ϵ-approximate solution to the distance problem. Since (p,p') is a witness pair there exists no p'-pivot for p, or a p-pivot for p'. However, if (p,p') is not already an ϵ-approximate solution to δ_*, the algorithm makes use of a *weak-pivot*.

Given a pair $(p,p') \in K \times K'$, the orthogonal bisector hyperplane of the line segment pp' is

$$H = \{x \in \mathbb{R}^m : \quad h^T x = a\}, \quad h = p - p', \quad a = \frac{1}{2}(p^Tp - p'^Tp'). \tag{2.17}$$

If (p,p') is a witness pair then

$$K \subset H_+ = \{x \in \mathbb{R}^m : h^T x > a\}, \quad K' \subset H_- = \{x \in \mathbb{R}^m : h^T x < a\}. \quad \Box \tag{2.18}$$

Let

$$v = \text{argmin}\{h^T x : x \in V\}, \quad v' = \text{argmax}\{h^T x : x \in V'\}. \tag{2.19}$$

$$H_v = \{x : h^T x = h^T v\}, \quad H_{v'} = \{x : h^T x = h^T v'\}. \tag{2.20}$$

Then the hyperplanes H_v and $H_{v'}$ give supporting hyperplanes to K and K', respectively, and the distance between them, $d(H_v, H_{v'})$ is a lower bound to δ_*. Specifically, let

$$\delta_v = d(v, H), \quad \delta_{v'} = d(v', H). \tag{2.21}$$

Then if

$$\underline{\delta} = \delta_v + \delta_{v'}, \tag{2.22}$$

we have

$$d(H_v, H_{v'}) = \underline{\delta} = \frac{h^T v - h^T v'}{\|h\|}, \tag{2.23}$$

and

$$\underline{\delta} \le \delta_* \le \delta = d(p,p'). \tag{2.24}$$

Figure 2.3, gives a geometric interpretation of the distances. Let (p,p') be a witness pair and consider v, v' as defined in (2.19). Also let H_v, $H_{v'}$ and H be the hyperplanes defined earlier. Let $\delta = d(p,p')$, $\rho = d(p,v)$, and $\rho' = d(p',v')$. Let $\underline{\delta} = d(H_v, H_{v'})$, and $E = \delta - \underline{\delta}$. We shall say (p,p') gives a *strong ϵ-approximate solution* to δ_* if either

$$E \le \epsilon\rho, \quad \text{or} \quad E \le \epsilon\rho'. \tag{2.25}$$

Given a witness pair (p,p'), let $\delta = d(p,p')$ and δ_v, $\delta_{v'}$ be as defined in (2.21). Define

$$E_v = (\frac{1}{2}\delta - \delta_v), \quad E_{v'} = (\frac{1}{2}\delta - \delta_{v'}). \tag{2.26}$$

Clearly,

$$E = \delta - \underline{\delta} = E_v + E_{v'}. \tag{2.27}$$

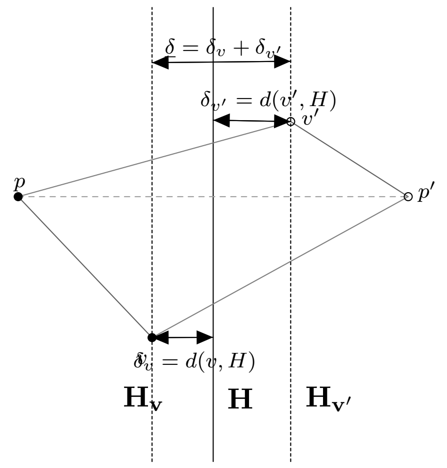


Figure 2.3: Distances from the separating hyperplane

The input to *Triangle Algorithm II* is a witness pair (p, p'). It computes a new witness pair (p, p') that gives an ϵ-approximate solution to δ_*, as well as a pair $(v, v') \in K \times K'$, where the hyperplanes parallel to the orthogonal bisecting hyperplane of pp' passing through v, v' form a pair of supporting hyperplanes, giving an ϵ-approximate solution to the supporting hyperplanes problem.(See box)

Triangle Algorithm II ($(p, p') \in K \times K'$, a witness pair, $\epsilon \in (0, 1)$)

- **Step 1.** Set $h = p - p'$. Compute

$$v = \text{argmin}\{h^T x : x \in V\}, \quad v' = \text{argmax}\{h^T x : x \in V'\}.$$

$$E = \delta - \underline{\delta}, \quad \delta = d(p, p'), \quad \underline{\delta} = (h^T v - h^T v')/\|h\|.$$

- **Step 2.** If $E \leq \epsilon\rho$, or $E \leq \epsilon\rho'$, with $\rho = d(p, v)$, $\rho' = d(p', v')$, output (p, p'), $(H_v, H_{v'})$, stop.

- **Step 3.** If $E_v > \frac{1}{2}\epsilon\rho$, compute $p' \leftarrow nearest(p'; pv)$, go to Step 5.

- **Step 4.** If $E_{v'} > \frac{1}{2}\epsilon\rho'$, compute $p \leftarrow nearest(p; p'v')$, go to Step 5.

- **Step 5.** Call Triangle Algorithm I with (p, p') as input. Go to Step 1.

In section 6 we discuss certain notes which are useful in an efficient implementation of the

Chapter 3

Sequential Minimal Optimization

The SMO algorithm relies on the fact that the optimal hyperplane w can be expressed as a convex combination of the input vectors i.e. $w = \sum_{i=1}^{m} \alpha_i y^{(i) x^{(i)}}$. It then performs a coordinate descent over the parameters $\alpha_1 \ldots \alpha_m$ to find the optimal hyperplane. The remainder of this section describes the Lagrange duality for the optimization problem and SMO algorithm.

3.1 Lagrange Duality

Given the training set $x^{(i)}$ and their convex hull membership denoted by y_i we define the *functional margin* γ to be

$$\gamma = y_i(w^T x^{(i)} + b) \quad y_i \in \{-1, 1\} \tag{3.1}$$

To keep the functional margin to be large we need $w^T x^{(i)} + b$ to be large positive number in case of $y^{(i)} = 1$ and a large negative number in case of $y^{(i)} = -1$. However, choosing $(2w, 2b)$ instead of (w, b) would also increase the functional margin without doing anything meaningful.

Enforcing $||w|| = 1$ ensures that the functional margin equals the geometric margin, and guarantees that all geometric margins are at least equal to γ.

To find the maximum of the minimum margins, equation **??** can be written as

$$\begin{aligned} & \underset{\gamma,w,b}{max}\ \gamma \\ s.t. \quad & y_i(w^T x^{(i)} + b) \geq \gamma, \quad i = 1, ...m \\ & \|w\| = 1 \end{aligned} \tag{3.2}$$

As $\|w\| = 1$ leads to a non-convex constraint, we formulate the following problem,

$$\begin{aligned} & max_{\gamma,w,b}\frac{\gamma}{||w||} \\ s.t. \quad & y^{(i)}(w^T x^{(i)} + b) \geq \gamma, \quad i = 1, ...m \end{aligned} \tag{3.3}$$

Maximizing $\frac{\gamma}{||w||} = \frac{1}{||w||}$ is equivalent to minimizing $||w||^2$. We now have an optimization problem that can be efficiently solved and the solution to which gives an *optimal-margin classifier.*

$$\begin{aligned} & \underset{w,b}{min}\ \frac{1}{2}\|w\|^2 \\ & s.t.\ y_i(w^T x^{(i)} + b) \geq 1, \quad i = 1, ...m \end{aligned} \tag{3.4}$$

In the machine learning literature, when their convex hulls are disjoint, the hard margin SVM problem is to compute a pair of supporting hyperplanes with maximum margin. When the convex hulls intersect, a soft margin SVM formulates the problem as a convex programming whose optimal solution provides a pair of supporting hyperplanes, necessarily allowing some mis-classifications.

In that case we define a relaxation coefficient for the separation and penalize the objective function with respect to the relaxation coefficient resulting in the soft margin formulation

$$\begin{aligned} & \underset{w,b}{min}\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i \\ & s.t.\ y_i(w^T x^{(i)} + b) \geq 1 - \xi_i, i = 1, ...m \\ & \xi_i \geq 0, i = 1, ...m \end{aligned} \tag{3.5}$$

The parameter C controls the weighting between the twin goals of minimizing $||w||^2$ and ensuring the most examples have a functional margin at least 1.

The Lagrangian for the optimization problem, as described in [6] turns out to be

$$L(w, b, \xi, \alpha, r) = \frac{1}{2}w^T w + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i[y_i(x^T w + b) - 1 + \xi_i] - \sum_{i=1}^{m} r_i\xi_i \tag{3.6}$$

Taking the derivative of the Lagrangian and setting it yo zero we get,

$$\begin{aligned} w &= \sum_{i=1}^{m}\alpha_i y_i x^{(i)} \\ \sum_{i=1}^{m}\alpha_i y_i &= 0 \end{aligned} \tag{3.7}$$

Plugging (3.7) back to (3.6) gives way to the Wolfe dual of the problem

$$\begin{aligned} &\underset{\alpha}{max} W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j x^{(i)} x^{(j)} \\ &s.t.\ 0 \leq \alpha_i \leq C, i = 1, ...m \\ &\sum_{i=1}^{m} \alpha_i y_i = 0 \end{aligned} \tag{3.8}$$

The KKT conditions for convergence are

$$\begin{aligned} \alpha_i = 0 &\implies y_i(wx^{(i)} + b) > 1 \\ \alpha_i = C &\implies y_i(wx^{(i)} + b) < 1 \\ 0 < \alpha_i < C &\implies y_i(wx^{(i)} + b) = 1 \end{aligned} \tag{3.9}$$

Notice that (3.8) does not solve for the constant b explicitly. Once we have the optimal w^*, b^* can be calculated as

$$b^* = -\frac{1}{2}(\underset{i:y_i=-1}{max} w^{*T} x^{(i)} + \underset{i:y_i=1}{min} w^{*T} x^{(i)}) \tag{3.10}$$

3.2 SMO algorithm

The SMO algorithm performs a coordinate descent over the set of all $\alpha_i s$. The constraint in (3.7) forces the sum of the products $\alpha_i y_i$ to be zero. For

$$\alpha_1 = \frac{1}{y_1}(-\sum_{i=2}^{m} \alpha_i y_i) \tag{3.11}$$

coordinate descent cannot be performed by adjusting a single value of α_i. We can perform coordinate descent between a pair (α_i, α_j) such that their sum is constant.

The SMO algorithm can be described as following:

- Select a pair (α_i, α_j) to update.
- Reoptimize $W(\alpha)$ with respect to (α_i, α_j), while holding other α_k $(k \neq i, j)$ fixed.

Specifically we choose α_i and α_j such that

$$\alpha_i y_i + \alpha_j y_j = -\sum_{k=1}^{i-1} \alpha_k y_k - \sum_{k=i+1}^{j-1} \alpha_k y_k - \sum_{k=j+1}^{m} \alpha_k y_k \tag{3.12}$$
$$\alpha_i y_i + \alpha_j y_j = \varsigma$$

where ς is some constant. This forces a line constraint. Moreover the KKT conditions(3.9) enforces the following box constraint.

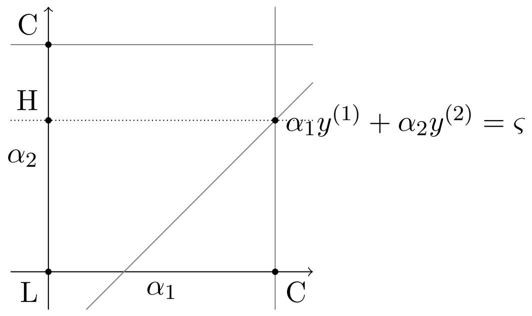


Figure 3.1: Constraints for (α_i, α_j) and $y^{(i)} = y^{(j)}$

Given the choice of two α_is to optimize we first compute the lower and upper bounds as following:

- $y_i \neq y_j, \qquad L = max(0, \alpha_j - \alpha_i), H = min(C, C + \alpha_j - \alpha_i)$ (3.13)
- $y_i = y_j, \qquad L = max(0, \alpha_j + \alpha_i - C), H = min(C, \alpha_j + \alpha_i)$ (3.14)

Now we want the optimal value for α_j to maximize the objective function. If this value ends up lying outside the bounds L and H, we simply clip the value of α_j to lie within this range. The update for α_j as discussed in [1] is

$$\alpha_j = \alpha_j + \frac{y_j(E_i - E_j)}{\eta} \tag{3.15}$$

where

$$f(x) = \sum_{i=1}^{m} \alpha_i y_i (x^{(i)}.x) + b \tag{3.16}$$

$$E_k = f(x^{(k)}) - y_k \tag{3.17}$$

$$\eta = 2x^{(i)}x^{(j)} - x^{(i)}x^{(i)} - x^{(j)}x^{(j)} \tag{3.18}$$

If the optimal value of α_j is not within the bounds we clip it as following

$$\alpha_j = \begin{cases} H, & \text{if } \alpha_j > H \\ \alpha_j, & \text{if } L \leq \alpha_j \leq H \\ L, & \text{if } \alpha_j < L \end{cases}$$

After solving for α_j we solve for α_i defined as

$$\alpha_i = \alpha_i + y_i y_j (\alpha_j^{(old)} - \alpha_j) \tag{3.19}$$

Where $\alpha_j^{(old)}$ is the value of α_j from the previous iteration.

Next we select the threshold b to satisfy the KKT conditions. If α_i is not at bounds (i.e. $0 < \alpha_i < C$) then the following threshold b_1 is valid

$$b_1 = b - E_i - y_i(\alpha_i - \alpha_i^{(old)})(x^{(i)} x^{(i)}) - y_j(\alpha_j - \alpha_j^{(old)})(x^{(i)} x^{(j)}) \tag{3.20}$$

Similarly if α_j is not at bounds the following threshold b_2 is valid

$$b_1 = b - E_j - y_i(\alpha_i - \alpha_i^{(old)})(x^{(i)} x^{(j)}) - y_j(\alpha_j - \alpha_j^{(old)})(x^{(j)} x^{(j)}) \tag{3.21}$$

If both the thresholds are not at bounds, then all thresholds between b_1 and b_2 satisfy the KKT conditions. This gives the following update equation

$$b = \begin{cases} b_1 & \text{if } 0 < \alpha_i < C \\ b_2 & \text{if } 0 < \alpha_j < C \\ (b_1 + b_2)/2 & \text{otherwise} \end{cases}$$

Each iteration of the SMO algorithm has two loops. The outer loops picks α_1 that corresponds to a sample violating the KKT conditions. The inner loop then picks another α_2 such that the error $E_2 - E_1$ is maximum.

SMO algorithm (Main loop)

- **Step 1.** Initialize $\alpha_i = 0$, $\forall$ i, b=0, passes=0, numChanged=0, examineAll=0
- **Step 2.** while ($numChanged > 0 \parallel examineAll$)

```
    ◦ numChanged=0
    ◦ if (examineAll)
        ◦ loop I over all training examples
            ◦ numChanged=numChanged + examineExample(I)
    ◦ else
        ◦ loop I over all training examples where alpha is not 0 or not C
            ◦ numChanged=numChanged + examineExample(I)
    ◦ if (examineAll == 1)
        ◦ examineAll = 0
    ◦ if (examineAll == 0)
        ◦ examineAll = 1
```

```
examineExample(i2)
  • Step 1. y2= target(i2), alph2=Lagrange multiplier for i2, E2 = error(i2) - y2, r2=E2*y2
  • Step 2. if $(r2 < -tol \,\&\, alph2 < C) \parallel (r2 > tol \,\&\, alph2 > 0)$
      ◦ if (number of non-zero and non-C alpha > 1)
          ◦ i1 = result of second choice heuristics
          ◦ if takestep(i1,i2) return 1
      ◦ loop over all non-zero and non-C alpha
          ◦ i1 = identity of current alpha
          ◦ if takestep(i1,i2) return 1
      ◦ loop over all possible i1
          ◦ i1 = loop variable
          ◦ if takestep(i1,i2) return 1
      ◦ return 0
```

```
takeStep(i1,i2)
  • Step 1. Clip and update $alph1$ and $alph2$
  • Step 2. Update threshold to reflect change in Lagrange multiplier
  • Step 3. Update weight vector to reflect change in alph1 and alph2
  • Step 4. Update error cache in using Lagrange multiplier
```

The SMO algorithm however, does not answer the question of separability of two convex sets. A hard margin classification can be enforced by setting the value of C to infinity, but in case of overlapping convex hulls it doesn't converge.

Chapter 4

Testing intersection or separation of Convex Hulls

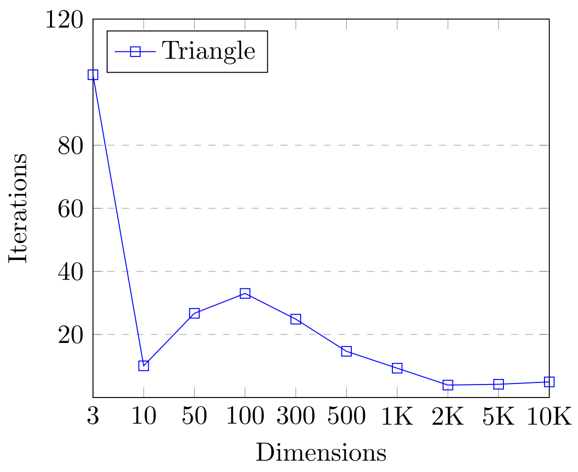
Testing for intersection of two finite Convex Hulls is a general case for testing if a point lies inside the Convex Hull. The triangle algorithm is efficient in answering the convex hull problem. [5]

In case of linearly separable sets we report the number of iterations it takes for the triangle algorithm to converge. The value of $\epsilon = 0.001$ was used and the maximum number of iterations was set to be 10^4.

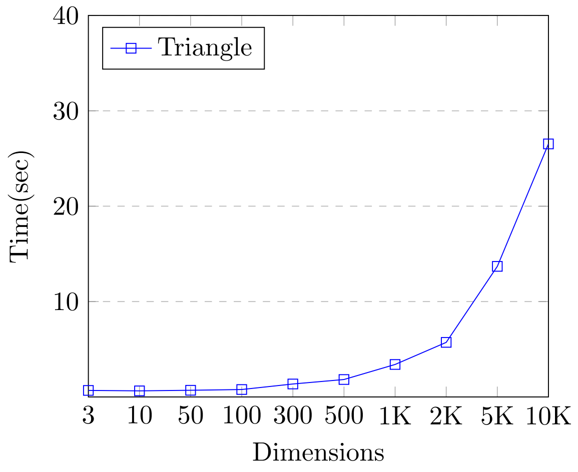
In the experimental setup we generated points in $V = \{v_1 \dots v_n\}$ and $V' = \{v'_1 \dots v'_n\}$ from two unit balls. Let $K = conv(V)$ and $K' = conv(V')$. We translated the smaller ball along a random direction by $\frac{9}{10} max\{diam(K), diam(K')\}$ units. Where

$$diam(K) = max\{d(v_i, v_j) : \quad (v_i, v_j) \in K\} \tag{4.1}$$

and $d(v_i, v_j)$ is the euclidean distance between v_I andv_j. The number of points in each of the set K and K' were 5000. Figure 4.1 shows the performance in the number of iterations and time.



(a)



(b)

Figure 4.1: Performance of triangle algorithm for testing intersection

	Triangle	
Dimensions	Iterations	Time(sec)
3	102.35	0.682
10	10.05	0.638
50	26.7	0.698
100	33	0.778
300	24.85	1.358
500	14.65	1.827
1,000	9.3	3.404
2,000	3.95	5.72
5,000	4.2	13.69
10,000	4.95	26.53

Table 4.1: Triangle algorithm performance for set intersection

Chapter 5

Testing approximation of Distance and Optimal support

We compared the performance of triangle algorithm with SMO for calculating the optimal support, given that the convex hulls are separate. To enforce a hard-margin classification we set the constant $C = Infinity$ for SMO. The value of $\epsilon = 0.001$ was used for both algorithms. Additionally the maximum number of iterations were set to be 10^4 for both algorithms.

5.1 Comparison Based on Dimension

In the experimental setup we generated points in $V = \{v_1 \ldots v_n\}$ and $V' = \{v'_1 \ldots v'_{n'}\}$ from two unit balls. Let $K = conv(V)$ and $K' = conv(V')$. We translated the smaller ball along a random direction by $max\{\frac{11}{10}diam(K), \frac{11}{10}diam(K')\}$ units. Where diam is defined in (4.1). The lower bound for the distance between the two convex hulls, $K = conv(V)$ and $K' = conv(V')$, becomes $max\{\frac{1}{10}diam(K), \frac{1}{10}diam(K')\}$. The number of points in each of the set V and V' were 5000. In our results we also report the sparsity and mean estimated distance calculated by both SMO and Triangle algorithm. From (3.7) we know that the hyperplane w is a sparse combination of the points in V and V'. The table shows the number of points it took to represent the optimal w. The distance is the reported distance between the sets V and V', which is calculated as

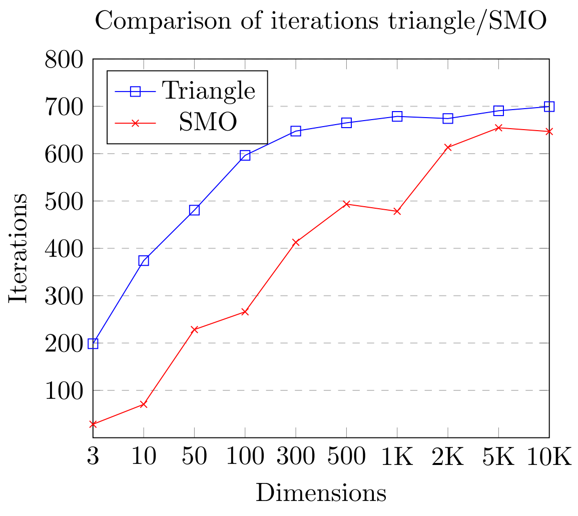
$$d = \frac{1}{\|w\|}(-max\{w^T v + b : \quad v \in V\} + min\{w^T v' + b : \quad v' \in V'\}) \tag{5.1}$$

Figure 5.1 shows the performance of both algorithms in terms of iterations and time.

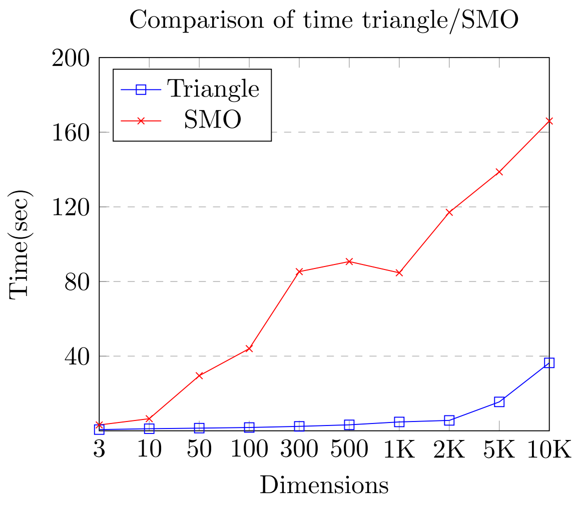
Based on our results we found that the triangle algorithm was faster in case of runtime against the SMO algorithm. This speed may be attributed to the linear work done in each iteration of the triangle algorithm, whereas each iteration of SMO is between linear and quadratic in, n, i.e.

	Triangle				SMO			
Dimensions	Iterations	Time(sec)	Sparsity	Distance	Iterations	Time(sec)	Sparsity	Distance
3	198.53	0.64	4.57	2.25	28.47	3.13	4.73	2.25
10	374.07	1.11	10.47	3.47	70.53	6.48	10	3.47
50	480.67	1.42	23.27	6.01	228.43	29.52	27.4	6.01
100	596.37	1.77	33.43	8.48	265.97	44.03	38.43	8.47
300	647.63	2.37	59.13	12.3	412.8	85.29	65.03	12.3
500	665.1	3.2	75.3	16.16	493.37	90.69	82.3	16.16
1,000	678.67	4.74	107.1	21.02	478.23	84.67	119.43	21.01
2,000	674.17	5.57	143.5	27.23	613.3	117.08	160.37	27.22
5,000	690.57	15.49	218.33	36.61	654.63	138.74	245.2	36.6
10,000	699.37	36.39	290.07	157.01	646.77	165.99	333.07	156.94

Table 5.1: Performance based on dimension



(a)



(b)

Figure 5.1: Performance of triangle and SMO for calculation of optimal support

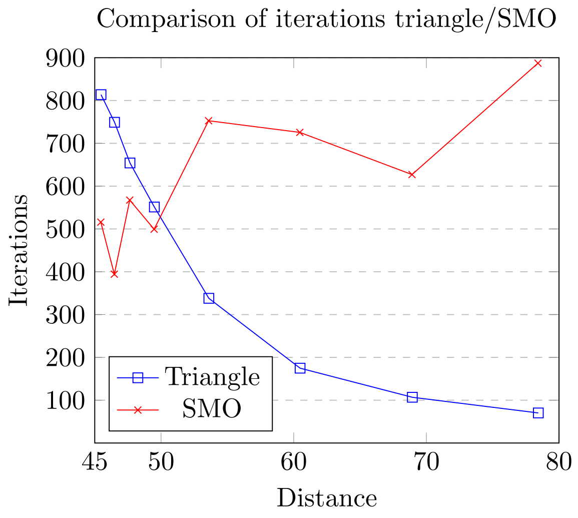
the number of points. We noticed that while being faster, the triangle algorithm calculates results similar to SMO in terms of sparsity and distance.

5.2 Comparison Based on Distance

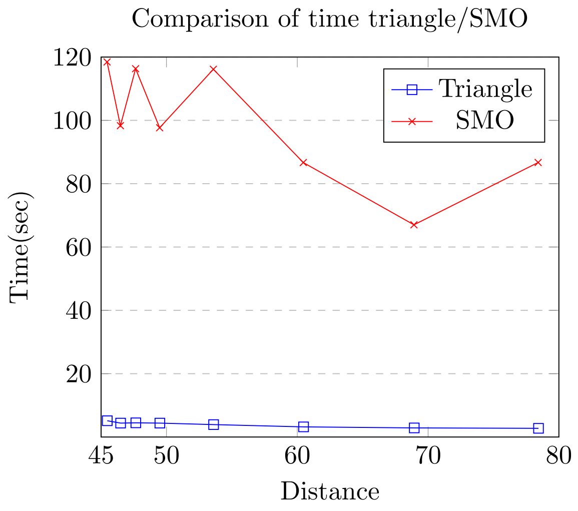
In this experimental setup we generated points in $V = \{v_1 \dots v_n\}$ and $V' = \{v'_1 \dots v'_{n'}\}$ from two unit balls. Let $K = conv(V)$ and $K' = conv(V')$. We translated the smaller ball along a random direction by $(1-k)max\{diam(K), diam(K')\}, \quad k \in [0,1]$ units. Where diam is defined in (4.1). By adjusting the value of k we were able to vary the distance between the convex hulls. The number of points in V and V' were 5000. The dimensionality was 1000.

	Triangle			SMO	
Iterations	Time(sec)	Distance	Iterations	Time(sec)	Distance
813.7	5.14	45.473	515.867	118.436	45.457
749.1	4.393	46.494	394.367	98.319	46.478
654.1	4.481	47.656	567.3	116.306	47.639
551.267	4.392	49.494	499.4	97.664	49.477
337.967	3.921	53.602	752.733	116.129	53.584
174.8	3.203	60.481	725.533	86.672	60.459
106.867	2.874	68.942	627.067	67.014	68.919
70.233	2.745	78.441	887.167	86.691	78.414

Table 5.2: Performance based on distance. Dimensions = 1000


Comparison of iterations triangle/SMO
Iterations
Distance
Triangle
SMO
900
800
700
600
500
400
300
200
100
45
50
60
70
80

(a)


Comparison of time triangle/SMO
Time(sec)
Distance
Triangle
SMO
120
100
80
60
40
20
45
50
60
70
80

(b)

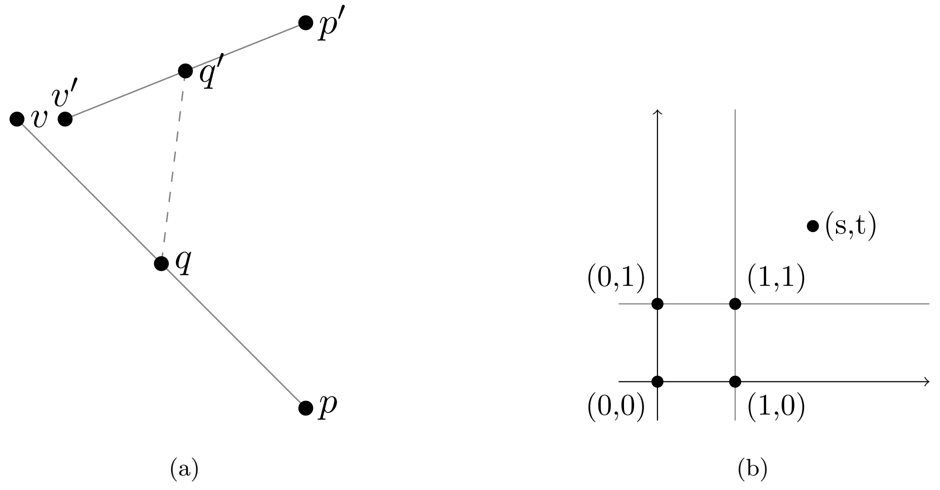
Chapter 6

Implementation notes

6.1 Closest Points Between Two Line Segments

In *Triangle Algorithm I and II*, in Step 2 and 3 we either move p close to p' along the segment pv or moves p' close to p along the segment $p'v'$. We can also jointly move p, p' closer to each other by finding the points closest to each other along the line segments pv and $p'v'$. Given two lines $L_1 = p - v$ and $L_2 = p' - v'$, we wish to determine points q, q' on L_1 and L_2 respectively such that

$$d(q, q') = min\{d(x, y) : \quad x \in L_1, y \in L_2\} \tag{6.1}$$



If L_1 and L_2 are not parallel and do not intersect, then the line segment $L_3 = qq'$ is perpendicular to both L_1 and L_2.

Let $\vec{a} = \overrightarrow{(v - p)}$ and $\vec{b} = \overrightarrow{(v' - p')}$. such that,

$$q = p + s\vec{a}, \quad s \in [0, 1] \qquad q' = p' + t\vec{b}, \quad t \in [0, 1] \tag{6.2}$$

then,

$$L_3 = q - q' \qquad L_3 = p + s\vec{a} - p' - t\vec{b} \tag{6.3}$$

Substituting (6.3) in (6.2), we get

$$(\vec{a}.\vec{a})s - (\vec{a}.\vec{b})t = -\vec{a}.(p - p') \qquad (\vec{b}.\vec{a})s - (\vec{b}.\vec{b})t = -\vec{b}.(p - p') \tag{6.4}$$

Solving for s and t, we get

$$s = \frac{(\vec{a}.\vec{b})(\vec{b}.(p - p')) - (\vec{b}.\vec{b})(\vec{a}.(p - p'))}{(\vec{a}.\vec{a})(\vec{b}.\vec{b}) - (\vec{a}.\vec{b})^2} \tag{6.5}$$

$$t = \frac{(\vec{a}.\vec{a})(\vec{b}.(p - p')) - (\vec{a}.\vec{b})(\vec{a}.(p - p'))}{(\vec{a}.\vec{a})(\vec{b}.\vec{b}) - (\vec{a}.\vec{b})^2} \tag{6.6}$$

If the lines are parallel the denominator in (6.5) and (6.6) goes to zero. In this case we can keep one of the parameters constant, say s=0, and solve for t.

After getting the initial estimates for (s, t) we wish to clip them such that the point lies between the line segments $[v, p]$ and $[v', p']$. (6.2)

$$s = \begin{cases} 0, & \text{if } s < 0 \\ s, & \text{if } 0 \leq s \leq 1 \\ 1, & \text{if } s > 1 \end{cases} \qquad t = \begin{cases} 0, & \text{if } t < 0 \\ t, & \text{if } 0 \leq t \leq 1 \\ 1, & \text{if } t > 1 \end{cases} \tag{6.7}$$

This method using the dot product works for points in any number of dimensions. [7]

6.1.1 Caching results

Step 2 and 3 of the *Triangle Algorithm I* , solves a maximization problem for finding the pivots i.e. if the value of $max\{(p - p')^T v : \quad v \in K\}) < \frac{1}{2}(||p'||^2 - ||p||^2)$ then v acts as a pivot.

Notice that we solve the same problem in Step 3 and 4 of *Triangle Algorithm II.* We can cache these results from Triangle Algorithm I and use it in Triangle Algorithm II. Specifically

Algorithm 1: Pivot selection

```
1 if $max\{(p-p')^T v : \quad v \in K\}) < \frac{1}{2}(||p'||^2 - ||p||^2)$ then
2     $pivot \leftarrow v$;
3 else
4     $extremepoint \leftarrow v$;
5 end
```

During each iteration we update the iterates as

$$\begin{aligned} p^{new} &= (1-\alpha)p^{old} + \alpha v \\ p'^{new} &= (1-\alpha')p'^{old} + \alpha' v' \end{aligned} \tag{6.8}$$

For a vector x let,

$$E_p = p^T x \quad E_{p'} = p'^T x \tag{6.9}$$

therefore,

$$E = (p-p')^T x = E_p - E_{p'} \tag{6.10}$$

We can update the errors as

$$\begin{aligned} {E_p}^{new} &= (1-\alpha){E_p}^{old} + \alpha v^T x \\ {E_{p'}}^{new} &= (1-\alpha'){E_{p'}}^{old} + \alpha' v'^T x \end{aligned} \tag{6.11}$$

The update for the error E is

$$E^{new} = {E_p}^{new} - {E_{p'}}^{new} \tag{6.12}$$

If we precompute the dot-products, similar to SMO, each update is $O(n)$.

6.2 Avoiding zig-zag

The Triangle Algorithm has been observed to suffer from zig-zagging. In this section we discuss the case for the same and a simple idea to overcome it. Let $V = \{v_1, \ldots, v_n\}$, $K = conv(V)$, $p \in K$ and we wish to move p closer to p' In this case both v_1 and v_2 are suitable for pivot selection. But p can either move along the line segment pv_1 or pv_2, and this causes it to zig-zag between v_1 and v_2 by choosing them as a pivot in an alternating fashion. We wish to move it along the segment pp'

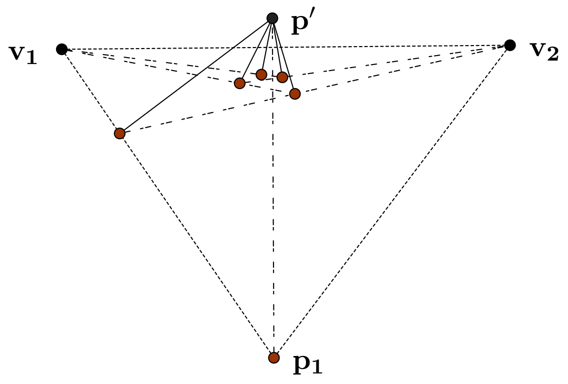


Figure 6.1: Case for Zig Zag.

instead. Figure 6.1 shows the trajectory of the movement of the iterate that is zig-zagging between v_1 and v_2. A simple solution would be to add another point $v_{ext} = (v_1 + v_2)/2$ which will be a better pivot choice and move p along the desired trajectory. To identify zig-zagging we can check if $\triangle d(p, p') < \epsilon d(p, p')$ where $\epsilon \in [0, 1]$. If this condition is met we can add the midpoint of two most frequently used pivots. This approach has been found to be working well in our experiments.

6.3 Reducing floating point operations

A remarkable property of Triangle Algorithm is that it continuously reduces the gap between the iterates $p \in conv(V)$ and $p' \in conv(V')$ in each iteration. This gap $d(p, p')$ also gives an estimate of the distance between the convex hulls. Between each iterations we have some points that are non-bounding i.e. $w^T v_i > w^T p, \quad \forall v_i \in V$ and $w^T v'_i < w^T p', \quad \forall v'_i \in V'$. Therefore, it makes sense to search for the next-pivot or extreme points within these non-bounding points first. Figure 6.2 shows the non-bounding points where p and p' act as iterates.

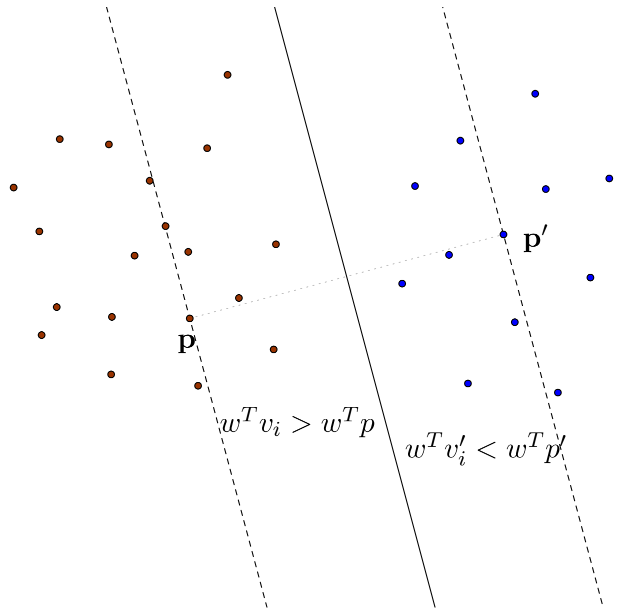


Figure 6.2: Non bounding points

If none of them satisfy the property then we can scan all the points. Notice that when we have the optimal hyperplane then we will find no such points in both the non-bounding set and the complete set. This is a heuristic idea to reduce the number of scans over the whole input.

6.4 Future work

Our experiments show that the triangle algorithm does well on the hard-margin formulation of the optimization problem. In this section we present the idea for solving the soft-margin problem using the triangle algorithm.

We use the following formulation for the soft-margin problem

$$\begin{aligned} &min \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i^2 \\ &s.t. \quad y_i(w^Tx^{(i)} + b) \geq 1 - \xi_i, \quad y_i \in \{-1, 1\} \end{aligned} \tag{6.13}$$

If we let

$$\eta_i = \sqrt{\frac{C}{2}}\xi_i, \quad i = 1, \dots, n \tag{6.14}$$

and use the following problem,

$$\begin{aligned} &min \quad \frac{1}{2}\|w\|^2 + \frac{1}{2}\|\eta\|^2 \\ &s.t. \quad y_i(w^T x^{(i)} + \sqrt{\frac{2}{c}}\eta_i + b) \geq 1, \quad y_i \in \{-1, 1\} \end{aligned} \tag{6.15}$$

it is solvable by the triangle algorithm.

References

[1] John C. Platt. Sequential minimal optimization:a fast algorithm for training support vector machines. *Microsoft Research, Technical Report MSR-TR-98-14*, (1), 1998.

[2] Bahman Kalantari. An algorithmic separating hyperplane theorem and its application. (2), 2014. arXiv:1412.0356v1.

[3] Bahman Kalantari. A characterization algorithm and an algorithm for a convex hull problem. *Annals of Operations Research*, 2014.

[4] Vladimir Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag New York, 2006.

[5] Bahman Kalantari and Meng Li. Experimental study of the convex hull decision problem via a new geometric algorithm, 2-page extended abstract. *23nd Annual Fall Workshop on Computational Geometry*, 2013.

[6] Vladimir Vapnik. Three remarks on the support vector method of function estimation. *Advances in Kernel Methods – Support Vector Learning, B. Scholkopf, C. Burges, A. Smola, eds., MIT Press*, 1998.

[7] Distance between 3d lines and segments. `http://geomalgorithms.com/a07-_distance.html`. Accessed: 2016-10-03.