A Simplex-cut Method for Nearest Facets in Minkowski Polytopes

by

Zhan Gao

Bachelor of Software Engineering, Southeast University, 2010

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science

In the Graduate Academic Unit of Faculty of Computer Science

Supervisor(s):	David Bremner, Ph.D, Computer Science
Examining Board:	Patricia Evans, Ph.D, Computer Science, Chair
	Huajie Zhang, Ph.D, Computer Science
	Barry Monson, Ph.D, Mathematics & Statistics

This thesis is accepted by the

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

April, 2014

Abstract

The Support Vector Machine algorithms are well known machine learning algorithms focused on classification and regression. The main idea of an SVM problem is to find a function to separate two data sets with a maximum *margin*. In this thesis, we focus on solving the linear SVM problem where the training sets cannot be separated by a linear function. We follow Cui's thesis [6] which converts the problem into a Minkowski Norm Minimization problem. We first introduce the basic formulation of SVM problem and some theory. We also briefly introduce Cui's geometric framework and show how the SVM problem can be view geometrically. In order to solve the Minkowski Norm Minimization problem, we propose to algorithms two trim the polytope. The implementation uses lrs [1] for enumeration and cdd [8] for linear programming; gmplib [7] is used for multi-precision calculation. The results of the experiments show that our methods have better performance than brute-force enumeration and that the cutting polytope method has better performance than cutting planes.

Table of Contents

Abstract **ii**

Table of Contents **vi**

List of Tables **vii**

List of Figures **ix**

1 Introduction **1**

1.1 Classification Problem 1

1.2 Support Vector Machine 2

1.3 Related Work 7

1.3.1 Decomposition methods 7

1.3.2 Approximation methods 7

1.3.3 Computational geometry methods 8

1.4 Improvement 8

2 Background **10**

2.1 Support Vector Machine 10

2.2 Basic formulation 11
2.3 The KKT condition and Lagrangian formulation 13
2.4 Classification violation formulation 15

3 Geometric Background **18**

3.1 Convex polytopes 18
3.1.1 Mathematical representation 20
3.2 Linearly inseparable SVM 20
3.3 Minkowski polytope 22
3.4 Brute-force approach 24
3.5 Main approach 25

4 Cutting-plane Method **26**

4.1 Basic Idea 26
4.2 Detailed description 28
4.2.1 Choose a radius 28
4.2.2 Choose a direction 29
4.2.3 Applying the cutting plane 30
4.3 Algorithm and Implementation 32
4.3.1 Determine a reference facet 33
4.3.2 Determine a reference direction 34
4.3.3 Generate new polytopes 35
4.4 Discussion 36
4.4.1 Drawbacks 37

5 Cutting-polytope Method **38**

5.1 Basic Idea 38

5.1.1 Radius of the cutting sphere 39

5.1.2 Non-iterative approach 40

5.1.3 Aspects of the cutting-polytope 41

5.1.3.1 "Shape" 41

5.1.3.2 "Type" 43

5.2 Algorithm and Implementation 44

5.2.1 Determine a reference vertex 44

5.2.2 Create cutting sphere 45

5.2.3 Create cutting simplex 45

5.2.4 Intersection points 48

5.3 Discussion 50

6 Experimental Results **51**

6.1 Experimental dataset 51

6.2 Experimental method 52

6.2.1 Comparison across different dimensions 52

6.2.2 Comparison across different methods 53

6.2.3 Comparison across different vertex numbers 57

6.3 Results 59

7 Conclusion and Future Work **60**

7.1 Summary and Conclusion 60

7.2 Open Questions and Future Work 61

Bibliography **66**

Vita

List of Tables

6.1 Size of datasets 52

List of Figures

1.1 Standard support vector machine 3

1.2 Classification violation 6

2.1 Margin of SVM 12

3.1 Convex polytope 19

3.2 Datasets represented as polytopes 21

3.3 Minkowski difference 23

4.1 Applying a cutting plane 27

4.2 A “best” direction 29

4.3 After applying the cutting plane 30

5.1 Cutting-polytope method 39

5.2 Radius of cutting sphere 40

5.3 Orthotopes 42

5.4 Orthoplices 42

5.5 Simplices 43

6.1 Average running time for each dimension 53

6.2 Running time for 5 dimensions 54
6.3 Running time for 6 dimensions 54
6.4 Running time for 7 dimensions 55
6.5 Running time for 8 dimensions 55
6.6 Running time for 9 dimensions 56
6.7 Running time for 10 dimensions 56
6.8 Running time for cutting plane method 58
6.9 Running time for cutting polytope method 58

Chapter 1

Introduction

1.1 Classification Problem

The classification problems are a group of supervised learning tasks. Supervised learning is to infer a function based on a given set of supervised training data, where each training example consists a set of inputs and a desired output. The inputs are represented by a set of attributes and the output value is known as the class label, especially when the output is discrete.

In machine learning, classification refers to a set of problems that assign new observations to one of the given set of classes by analyzing a labelled training dataset. Let x denote a vector of observed attributes for the inputs and y denote the class label. Then for a given dataset $\{(x_i, y_i)\}$, the classifiers will create a classification function h to evaluate any new example x^* and produce its label $y^* = h(x^*)$. Some of the widely applied classifiers include Decision

Trees, Neural Networks, K-nearest Neighbour and Support Vector Machine [16].

1.2 Support Vector Machine

The support vector machine algorithms (see Fig. 1.1) are one of the main pattern classification algorithms. In the case of SVM algorithms, the inputs are represented as data points, usually in a finite dimensional space. Many algorithms will also map the original space into a higher dimensional space, the space is called *feature space* and the mapping function is called a *kernel function*.

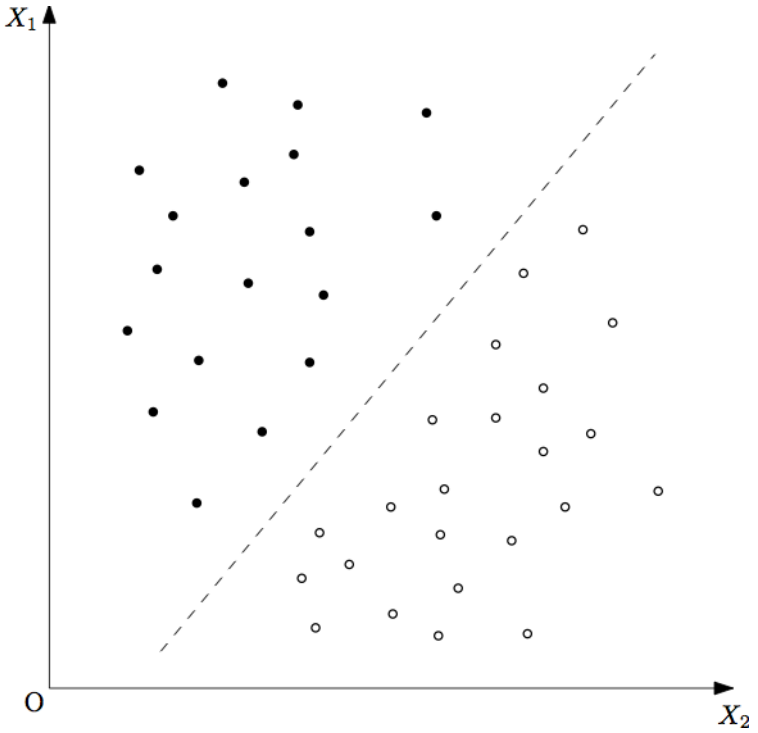


Figure 1.1: Standard support vector machine

Definition 1.2.1 (Inner products)**.** *For $z, x \in \mathbb{R}^n$, the inner product $\langle z, x \rangle$ is the value of the functional z at the point x, and we define it as follows:*

$$\langle z, x \rangle = \sum_{i=1}^{n} z_i x_i$$

An inner products gives a simple type of similarity measure. Since we need to measure the similarity of the new data points with the given ones, i.e. the training data, we will define a function as follows:

$$K(x, \tilde{x}) = \langle \phi(x), \phi(\tilde{x}) \rangle.$$

This function is called kernel function, where x and $\tilde{x}$ are two data points and the function ϕ is the map function from the original space $\mathbb{X}$ to the feature space $\mathbb{H}$.

There are two major benefits of this: one is that in the original space the data set may not be linearly separable, by mapping it to a higher dimensional space, it might be easier to make the separation; the other benefit is that the *kernel trick* [19] can be applied for nonlinear classifiers. The kernel trick is that for a given algorithm that is constructed with a positive definite kernel k, by replacing k with another positive definite kernel function $\hat{k}$ the original function ϕ can be mapped into a new function ϕ'. Normally the kernel function can be lifted into another space so that the problem can be linearly separable. We will discuss the details of the SVM in the next chapter.

However, some complicated kernel functions map the training data into a much higher dimensional space to obtain separability, which makes solving the problem much harder. For example, image recognition of a 16×16 pixel input image with a degree 5 monomial kernel function will yield a dimension of almost 10^{10}. Many algorithms are proposed to address this problem to improve the *generalization performance*. In machine learning, the generalization performance of an algorithm is its ability to extract valuable information from former examples. These algorithms can be categorized into three groups: decomposition methods [17], approximation methods [10, 14, 20], and computational geometry methods [4, 13].

Decomposition methods will break down the optimization problem into smaller

ones. In each iteration, they only consider a small subset of variables. These methods are mainly used to solve the memory sensitive problems but will be very time consuming for large scale work. In order to speed up the training process, a variety of approximation variants of standard SVM have been proposed. Therefore the basic problems are able to be transformed into some linear system of equations with a comparatively moderate number of variables. An alternative way of solving the SVM problem uses computational geometry based methods. Instead of treating the SVM as a convex programming problem, it is transformed into a problem to compute the nearest points between two convex polytopes. This method is comparatively faster than traditional algorithms. However, one drawback of this method is that when classification violations (see Fig. 1.2) are allowed, the existing algorithms will fail.

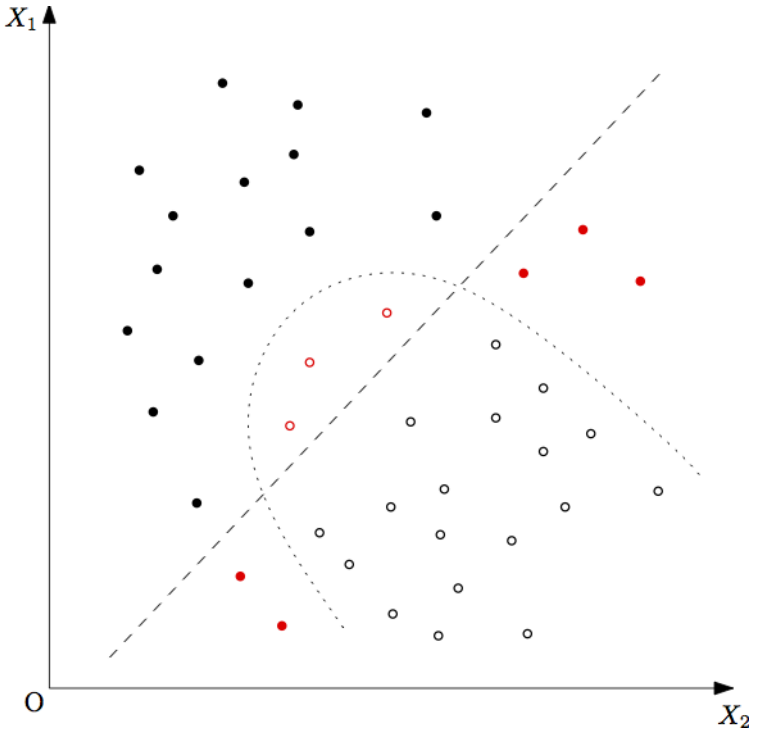


Figure 1.2: Classification violation

When classification violations are allowed, one approximation method is to introduce a trade-off parameter C to form a soft margin. Even though the outliers can be taken care of nicely by these methods, the problem becomes non-convex, which in turn makes it considerably more difficult to solve. Cui's thesis [6] presents a framework to compute the "exact" margin in the non-separable case, without incorporating any trade off parameters, by applying a non-Euclidean norm. In this thesis, we will use a similar approach and present another precise method using a Euclidean norm.

1.3 Related Work

As mentioned in section 1.2, the methods of solving SVM problems can be categorized into three groups:

1.3.1 Decomposition methods

The idea of decomposition methods is to divide the variables α_i (which will be defined in Chapter 2) into two sets: one set of free variables, which will be updated in the current iteration and a set of fixed variables, which are temporarily fixed and will be updated in later iterations. Depending on different algorithms, a few extra parameters will be added to the basic formulation to allow classification violations. One of the most popular decomposition methods is the Sequential Minimal Optimization (SMO) [17]. In each iteration, only two variable will be updated.

1.3.2 Approximation methods

The approximation methods are the most studied methods of all three categories, due to the motivation of speeding up the training process. Slack variables are added to the basic SVM formulation (will be introduced in chapter 2 as equation (2.2) (see page 12) in order to apply the approximation. Some algorithms are strongly convex when forming the dual problem under KKT conditions [18], such as LSVM [14]. We will discuss the details of approximation methods in chapter 2.

1.3.3 Computational geometry methods

Computational geometry methods will convert the convex programming problems into the problem of finding the nearest points of two convex polytopes. Because of the geometric nature of the algorithms, some data points can be purged to increase the speed of the training. The algorithm proposed by Keerthi [13] revised and combined several nearest point problem and minimal norm problem algorithms due to Gilbert [11, 12] and Wolfe [22]. These methods, despite their improved performance, do not generally deal with classification violations.

1.4 Improvement

In this thesis, we will describe two algorithms: cutting-plane method and cutting-polytope method. Both methods are computational geometry methods. Compared to typical geometric methods, the cutting-plane method provides an intuitive way to deal with classification violations; the cutting-polytope method makes a further improvement and solves some of the drawbacks found with the cutting-plane method.

Both the cutting-plane method and the cutting-polytope method will not have as good performance as approximation methods. But we do not introduce slack variables into these methods. The result is more accurate than approximation methods, since the only uncertainty is caused by unavoidable rounding of numbers instead of by newly introduced slack variables.

There is little basis for comparison with the decompositional methods, since their benefits are with problems involving limited memories. In general, both the cutting-plane method and the cutting polytope method will consume more memory but less time, as they will not split the variables into two sets. This reduces a large number of iterations. The cutting-plane method does have many iterations due to the fact that each cutting-plane requires a separate iteration to calculation all the intersection points; the cutting-polytope method, however, solves this drawback and eliminates the requirement of multiple iterations.

Chapter 2

Background

2.1 Support Vector Machine

The support vector machine algorithms are important pattern classification algorithms. The inputs of SVM problems are represented as data points, usually in a finite dimensional space. The space will usually be mapped into a much higher dimensional space using a *kernel function*; this space is called a feature space.

Definition 2.1.1 (Hyperplane)**.** *A* **hyperplane** *of an n-dimensional vector space V is a "flat" subset of dimension $n-1$, sometimes called an $(n-1)$-flat of V. If not otherwise noted, any hyperplane in this thesis is an* **affine hyperplane***, namely a subset of V, which can be described with a single linear equation:*

$$w_1 z_1 + w_2 z_2 + \cdots + w_n z_n = b. \tag{2.1}$$

(where at least one of the $w_i \neq 0$).

A hyperplane is thus a generalization of a two-dimensional plane in ordinary space.

Definition 2.1.2 (Linearly separability)**.** *Given a hyperplane $H \in \mathbb{R}^n$ defined by* (2.1)*, we say that two sets of points P and Q are linearly separable, if $\forall p \in P$ and $\forall q \in Q$, we have $a_1p_1 + a_2p_2 + \cdots + a_np_n < b$ and $a_1q_1 + a_2q_2 + \cdots + a_nq_n > b$. If this fails for all hyperplanes H, then P and Q are* **linearly inseparable***.*

2.2 Basic formulation

Since multi-class problems can be reformulated into multiple binary classifications, and since nonlinear SVMs can be transformed into linear SVMs via the kernel trick, in this chapter we can focus on the binary linear support vector machine, i.e. the basic SVM.

The basic SVM formulation does not allow classification violations, i.e. the training set is linearly separable. In the simplest situation, the SVM algorithm tries to find a hyperplane $H = \{z \mid w \cdot z + b = 0\}$ which separates two sets.

Let d^+ and d^- be the shortest distance from H to the closest positive and negative examples. We define the *margin* (see Fig. 2.1) of H be $d^+ + d^-$. The goal for the basic SVM is to find an H with the largest margin.

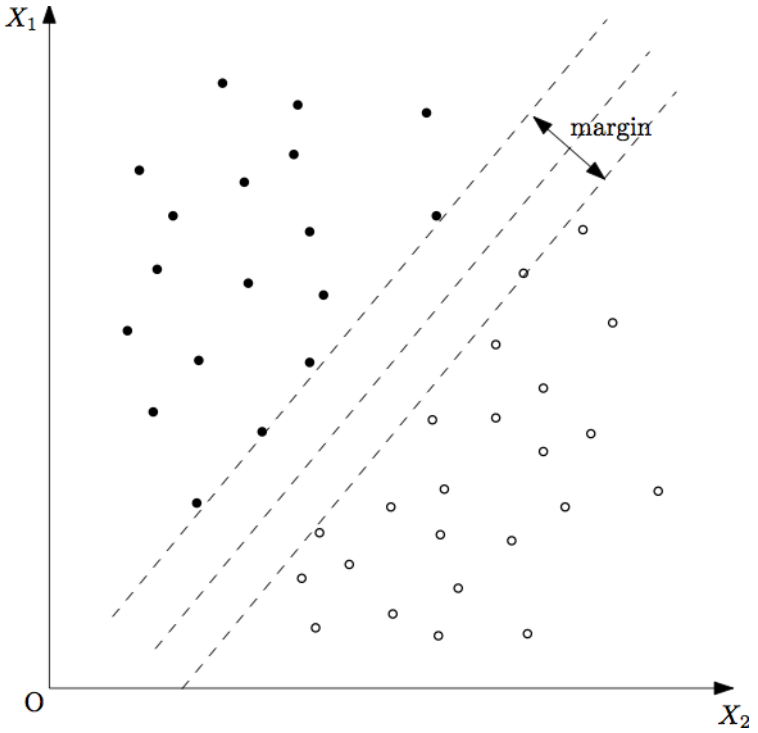


Figure 2.1: Margin of SVM

In general, the SVM training problem can be formulated as follows:

$$\begin{aligned} \min \quad & \frac{1}{2}\|w\|^2 & & \\ \text{such that} \quad & w \cdot z_i + b \geq 1 & \forall i \in I & \\ & w \cdot z_j + b \leq -1 & \forall j \in J & \end{aligned} \tag{2.2}$$

Here the vector w is the normal vector of the separating hyperplane H, and the z_i are the data points from one class and z_j are those from the other class. The value of margin equals to $\frac{2}{\|w\|}$. We also define $H^+ = \{z \mid w \cdot z + b = 1\}$

and $H^- = \{z \mid w \cdot z + b = -1\}$ as the bounding hyperplanes that separate two classes.

2.3 The KKT condition and Lagrangian formulation

Even though we don't directly use the KKT condition and Lagrangian formulation in our thesis, the concepts are briefly introduced in this section to derive several functions which will be applied in the next section.
Let us define a convex program [18] as follows:

$$\begin{aligned} \min \quad & f_0(x) \\ \text{such that} \quad & f_i(x) \leq 0 \qquad \forall i \in \{1, 2, \cdots, r\} \\ & f_j(x) = 0 \qquad \forall j \in r+1, r+2, \cdots, m \end{aligned} \tag{2.3}$$

where each f_i is a convex function from $\mathbb{R}^n$ to $\mathbb{R}$. A Kuhn-Tucker vector is a vector of Kuhn-Tucker coefficients $(\lambda_1, \lambda_2, \cdots, \lambda_m) \in \mathbb{R}^m$ for program (2.3), if the following conditions hold:

- $\lambda_i \geq 0, \forall i \in 1, 2, \cdots, m$,
- for $f = f_0 + \lambda_1 f_1 + \lambda_2 f_2 + \cdots + \lambda_m f_m$, $\inf(f)$ is finite,
- $\inf(f)$ is equal to the optimal value of program (2.3).

We will give the general definition of the *Lagrangian* that will be used in the *Lagrangian formulation*. Let C be the feasible region of a convex program (2.3) and define a vector E_r:

$$E_r = \{u^* = (\lambda_1, \lambda_2, \cdots, \lambda_m) \in \mathbb{R}^m \mid \lambda_i \geq 0, i = 1, 2, \cdots, m\}$$

where λ_i are called *Lagrange multipliers*, then the Lagrangian of (2.3) is a function L on $\mathbb{R}^m \times \mathbb{R}^n$, where m is the number of constraints and n is the dimension of C, we define:

$$L(u^*, x) = \begin{cases} f_0 + \lambda_1 f_1 + \lambda_2 f_2 + \cdots + \lambda_m f_m & if\ u^* \in E_r, x \in C, \\ -\infty & if\ u_* \notin E_r, x \in C \\ +\infty & if\ x \notin C \end{cases}$$

In order to convert the basic SVM to the Lagrangian formulation, we will need to add Lagrange Multipliers for each constraint in (2.2).

First combine two sets of constraints into one set of inequalities as follows:

$$y_i(w \cdot z_i) \geq 1 \quad \forall i \in \{1, 2, \cdots, m\} \tag{2.4}$$

Here y_i is the result label of the training example, where $y_i = 1$ when the example is positive and $y_i = -1$ when the example is negative. Notice that the ordinary linear constraints are now replaced by non-linear constraints.

In the Lagrangian reformulation, one non-negative Lagrange multiplier, α_i will be associated with each inequality constraint. Thus the Lagrangian is:

$$O_L = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha_i y_i (w \cdot z_i + b) + \sum_{i=1}^{m} \alpha_i \tag{2.5}$$

The primal problem is to minimize the objective function O_L with respect to the variables w and b subject to the constraints $\alpha_i \geq 0$, and meanwhile require that the derivative of O_L with respect to all the Lagrange multipliers α_i vanish. Apply this to the KKT conditions [18], we find the objective function of the dual problem by using Wolfe duality theory [23]:

$$O_{L_D} = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j \langle z_i, z_j \rangle \quad \forall i, j \in \{1, 2, \cdots, m\} \tag{2.6}$$

Thus the Lagrangian formulation of formula (2.2) is:

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle z_i \cdot z_j \rangle \\ \text{such that} \quad & \sum_{i=1}^{m} \alpha_i y_i = 0 \\ & \alpha_i \geq 0 \qquad\qquad \forall i, j \in \{1, 2, \cdots, m\} \end{aligned} \tag{2.7}$$

2.4 Classification violation formulation

The function (2.7) only applies to linearly separable cases. In this thesis we mainly deal with the situation when the training data is linearly inseparable in the feature space; hence classification violations will be allowed. To include

a penalty for the violations in the objective function, we reformulate the problem as follows:

$$\begin{aligned} \min \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i \\ \text{such that} \quad & w \cdot z_i + b \geq 1 - \xi_i \qquad \forall i \in I \qquad (2.8) \\ & w \cdot z_j + b \leq -1 + \xi_i \qquad \forall j \in J \\ & \xi_i \geq 0 \qquad \forall i = 1, 2, \cdots, m \qquad (2.9) \end{aligned}$$

Here a non-negative slack variable ξ_i is introduced for each constraint, and a trade-off parameter C is chosen by the user. The larger C is, the higher penalty errors, and when an error occurs, a corresponding positive value must be assigned to ξ_i in order to exceed 1.

In this case, the objective function of (2.7) is reformulated as follows:

$$O_{L_{vl}} = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i(y_i(w \cdot z_i + b) - 1 + \xi_1) - \sum_{i=1}^{m}\gamma_i\xi_i \quad (2.10)$$

Here the γ_i are the Lagrange multipliers introduced for non-negativity constraint. By applying this to the KKT conditions we get:

$$O_{L_{VD}} = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\alpha_i\alpha_j y_i y_j \langle z_i \cdot z_j \rangle \quad \forall i, j \in \{1, 2, \cdots, m\} \qquad (2.11)$$

So the dual problem of (2.8) is:

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle z_i \cdot z_j \rangle \\ \text{such that} \quad & \sum_{i=1}^{m} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \qquad\qquad \forall i, j \in \{1, 2, \cdots, m\} \end{aligned} \tag{2.12}$$

However, after this transformation, the problem is no longer convex, thus making it much more difficult to find a global solution.

Chapter 3

Geometric Background

3.1 Convex polytopes

Definition 3.1.1 (Convex set)**.** *A set $C \subseteq \mathbb{R}^d$ said to be* **convex** *if, $\forall x$ and $y \in C$, and $\forall t \in [0,1]$, the point $(1-t)x + ty \in C$.*

Definition 3.1.2 (Convex Hull)**.** *Convex hull of a set $X \subseteq \mathbb{R}^d$ is the smallest convex set that contains X. It is easy to see that this is the intersection of all convex subsets of $\mathbb{R}^d$ which do contain X.*

Definition 3.1.3 (Convex Combination)**.** *Given a finite set of points x_1, x_2, ..., $x_n \in \mathbb{R}^d$, a convex combination of these points is any point of the form $\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n$, where $\alpha_i \geq 0$ and $\alpha_1 + \alpha_2 + \cdots + \alpha_n = 1$.*

Definition 3.1.4 (Bounded Set)**.** *A set $S \subseteq \mathbb{R}^d$ is called* bounded *if it is contained in a ball of finite radius.*

A set S of real numbers is called bounded from above *if there is a number k such that $k \geq s$ for all $s \in S$. The number k is called an* **upper bound** *of S. Similarly a* **lower bound** *can be defined. If S has both an upper bound and a lower bound, then S is* bounded.

Intuitively, a polytope is a geometric object with flat sides called faces.

Definition 3.1.5 (Convex Polytope)**.** *A convex polytope in $\mathbb{R}^d$ (see Fig. 3.1) is the convex hull of a finite set of points. If this finite set is minimal the points are called vertices.*

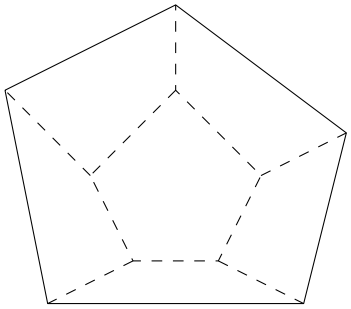


Figure 3.1: Convex polytope

An n-dimensional convex polytope C, briefly *n-polytope*, is bounded by (n-1)-polytopes, which in turn are bounded by (n-2)-polytopes, and so forth. These polytopes are all faces of the n-polytope. Specially, a 0-face is called a **vertex**; a 1-face is called an **edge**, and an (n-1)-face is called a **facet**.

3.1.1 Mathematical representation

A convex polytope is usually an infinite set of points. In order to computationally deal with a convex polytope, we need to have a finite representation. Typically a convex polytope has two different types of representations: vertex representation and half-space representation.

Definition 3.1.6 (Vertex representation)**.** *A convex polytope C can be represented as a convex hull of a finite set of points, where the finite set must contain the set of extreme points (or vertices) of the polytope. Such a definition is called a vertex representation or $\mathcal{V}$-representation of C. We denote the set of vertices of the polytope C by $V(C)$.*

Definition 3.1.7 (Half-space representation)**.** *A convex polytope C is the feasible region of a series of linear inequalities. Each inequality represents a certain half-space in $\mathbb{R}^d$. The minimal set of inequalities is called half-space representation or $\mathcal{H}$-representation.*

There is a fundamental theorem [24] that these representations of C are equivalent.

3.2 Linearly inseparable SVM

Linearly inseparable SVM problems are a set of SVM classification problems where the two datasets are linearly inseparable. While the goal of the linearly

separable SVM problem is to find a margin or largest "gap", the goal of the linearly inseparable SVM is to find an overlap of smallest "intersection".

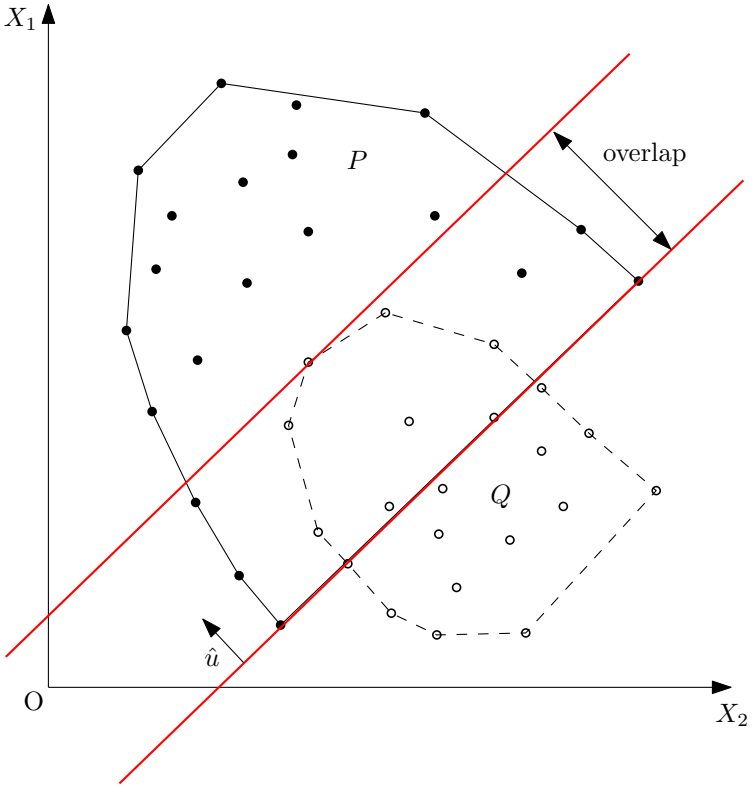


Figure 3.2: Datasets represented as polytopes

As in Fig. 3.2, polytopes P and Q are convex hulls of two data sets, $\hat{u}$ is a unit vector which is orthogonal to the separation hyperplane, and p and q are arbitrary points of P and Q. Thus in the linear separable case, the margin is defined as:

$$\text{margin} = \sup_{\forall \hat{u}} \inf_{p \in P, q \in Q} \langle p - q, \hat{u} \rangle \tag{3.1}$$

here $\hat{u}$ is any unit vector in the given space.

This is called a Nearest Point Problem [22, 12, 11, 13].

Similarly in linearly inseparable cases,

$$\text{overlap} = \inf_{\forall \hat{u}} \sup_{p \in P, q \in Q} \langle p - q, \hat{u} \rangle \tag{3.2}$$

3.3 Minkowski polytope

In order to solve equation (3.2), Cui [6] applied a method of creating a Minkowski polytope [21, 9].

Definition 3.3.1 (Minkowski Sum)**.** *Given two convex polytopes P and Q, define the Minkowski Sum of P and Q as $P+Q = \{p+q \mid p \in P \text{ and } q \in Q\}$.*

Correspondingly, we can define the Minkowski Difference as

$$P - Q = \{p - q \mid p \in P \text{ and } q \in Q\} \tag{3.3}$$

Since both P and Q are convex polytopes, we can describe $P - Q$ as the convex hull of $\{p - q \mid p \in V(P), q \in V(Q)\}$ (see Fig. 3.3).

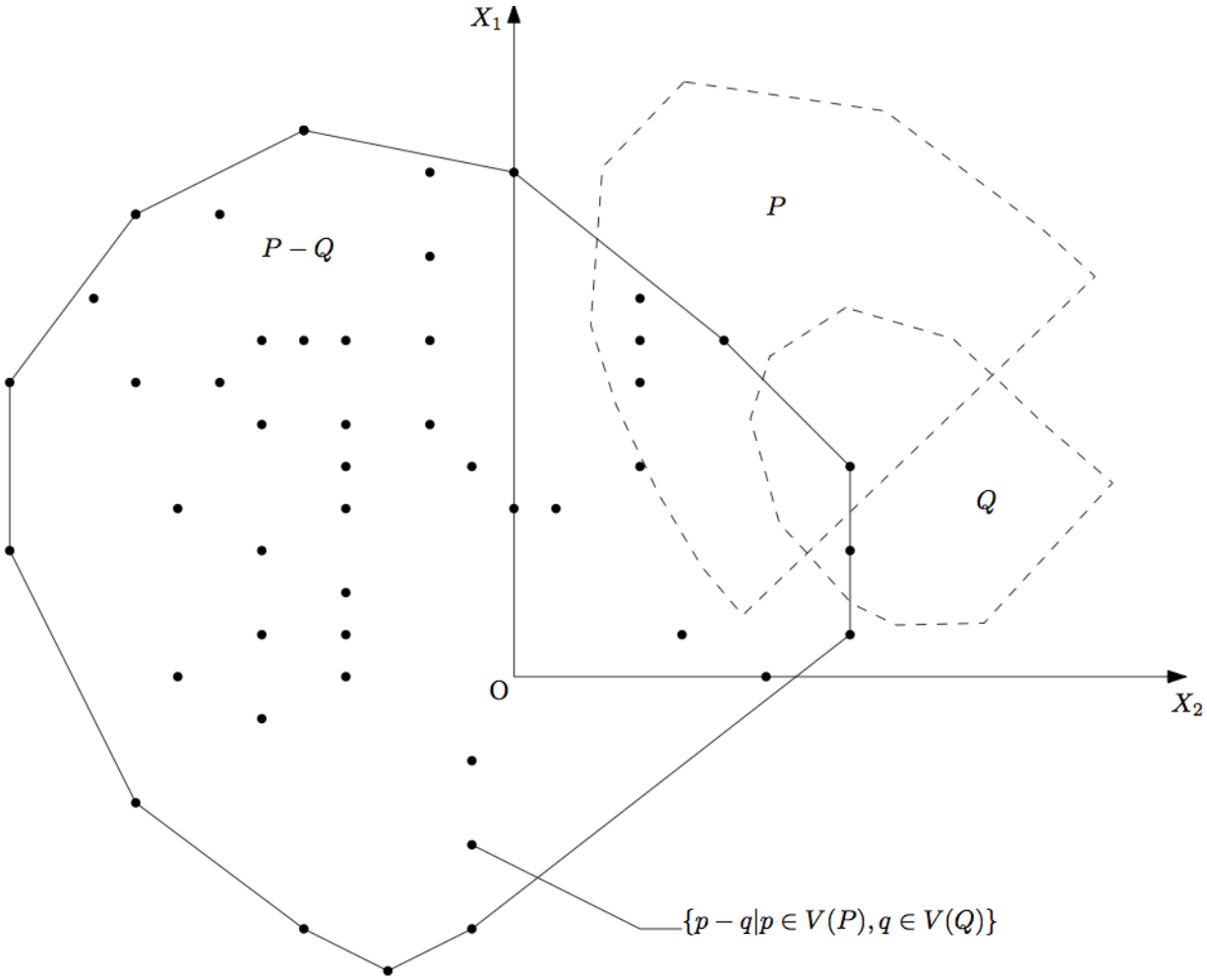


Figure 3.3: Minkowski difference

$P - Q$ is itself a polytope, thus formula (3.2) can be rewritten as:

$$\min \|r\| \tag{3.4}$$

$$\text{such that } r \in \text{bd}(P - Q).$$

Here the boundary $\text{bd}(C)$ of a polytope C is the union of all facets of C.

The proof of this transformation can be found in Cui [6], (3.2) is called a Norm Minimization problem [15].

3.4 Brute-force approach

If the origin $\mathbb{O}$ locates outside of $P - Q$, the "*bd*" part of (3.4) can be eliminated, making it a convex program. However, in the linearly inseparable case, $\mathbb{O}$ always lies inside $P - Q$. We then are dealing with the boundary itself, thus making the constraints of (3.4) become a disjunction of equalities in addition to inequalities.

Since $P - Q$ is generated from the two data sets of the SVM problem, it is obvious to be represented as a convex combination of its vertices. In order to get the $\mathcal{H}$-representation of $P - Q$, we need to enumerate the facets of $P - Q$

However, there is no obvious relationship between the number of vertices (v) and facets (f) for an arbitrary convex polytope. For example we know that for a cross polytope $f = 2^{v/2}$ and for a hypercube $f = 2\log_2 v$, but for $P - Q$ generated from two unknown polytopes P and Q, it might take a long time to enumerate all facets of $P - Q$. The facet numbers are unpredictable.

3.5 Main approach

Since the problem we are dealing with is specifically derived from SVM problems, we can infer that the volume of $P \cap Q$ is typically much smaller than both the volumes of P and Q, which means that the centroid of $P - Q$ is highly shifted from $\mathbb{O}$. In other words, most parts of the boundary of $P - Q$ are farther away than other parts. With this property, it wastes a lot of time doing the calculation of the vertices and facets which are far away from $\mathbb{O}$. And after trimming $P - Q$ into a much smaller set of vertices, it will be faster to enumerate the remaining facets. In the following chapters, two different approaches will be introduced.

Chapter 4

Cutting-plane Method

4.1 Basic Idea

The very basic idea is to find a *cutting sphere* S of a polytope P where P is the previously discussed Minkowski Polytope. In order to make S a cutting sphere, it should meet the following requirement: neither $S \subset P$ nor $P \subset S$ should apply. By applying a cutting sphere with a carefully selected radius, the vertices and facets which are relatively far away from the origin $\mathbb{O}$ will be eliminated. However, in order to construct a new model, the remaining vertices are not necessarily enough. In order to form a new polytope, we must introduce the intersection points of S and P.

Although we ultimately apply this method to $P - Q$, for noting convenience, we simply describe the method as applied to the polytope P.

Since S is non-linear, there is no fast way to find the intersection of P and

S. In order to simplify the question, the cutting-plane method is introduced. The basic idea of cutting-plane method (see Fig. 4.1) is to construct a supporting hyperplane H of the sphere S and use H to cut off the suboptimal region of P. The new model will be represented by two sets of vertices:

- vertices of P which are in the feasible region of P (V_1);
- points intersected by P's edges and H (V_2).

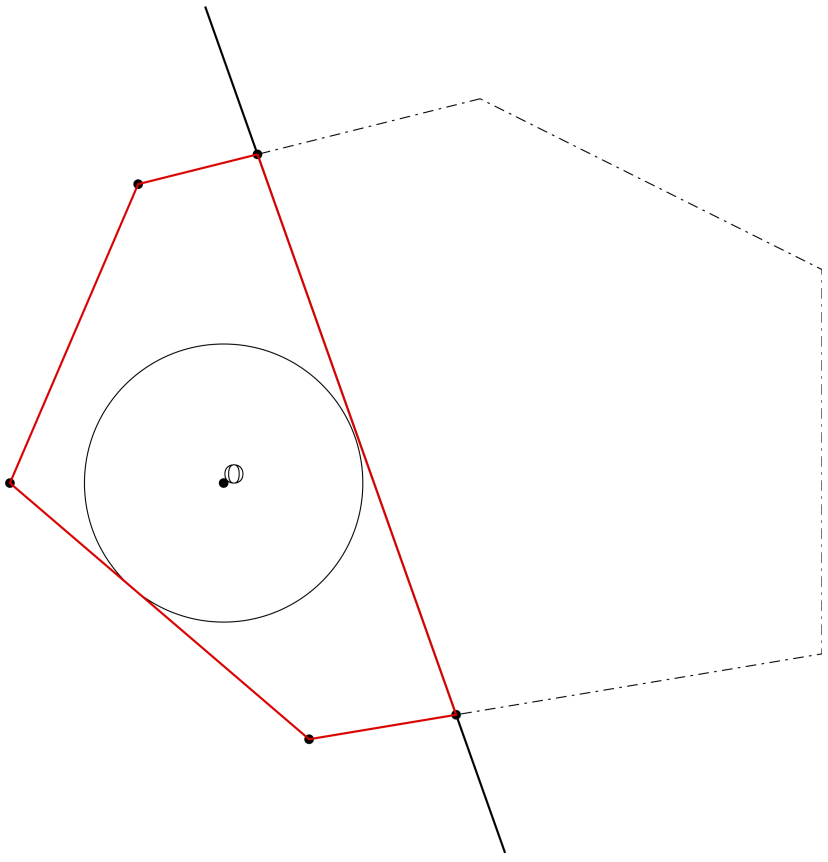


Figure 4.1: Applying a cutting plane

This step can be applied for several iterations until further iteration would not provide any further optimization. However, before actually implementing this method, the following issues have to be considered:

- the radius of the cutting sphere should be well selected;
- the direction of the cutting plane tangent to the sphere should be well selected as well;
- in each iteration, the optimal solution of the new model should be the same as the previous optional solution.

4.2 Detailed description

4.2.1 Choose a radius

We need to choose a suitable radius of the cutting sphere as the first step of each iteration. The ideal radius is the smallest distance from the boundary of P ($bd(P)$) and $\mathbb{O}$, which is actually just one more step to the final result, but is not so easy to get from a $\mathcal{V}$-representation.

A simple workaround is to sample a random set of facets and find the smallest distance from that sample. In most cases this will generate a small enough sphere compared to the size of P.

In practice, the lrs library has already provided a method to generate the $\mathcal{H}$-representation from the $\mathcal{V}$-representation of P. That takes a relatively

long time to perform the whole conversion, but we can stop the process after say the first hundred facets and use these facets as the sample set.

4.2.2 Choose a direction

Compared to the previous issue, choosing a good direction is rather easy. Since our model is in $\mathcal{V}$-representation, it takes almost no time to enumerate all vertices and calculate the distance of each from $\mathbb{O}$.

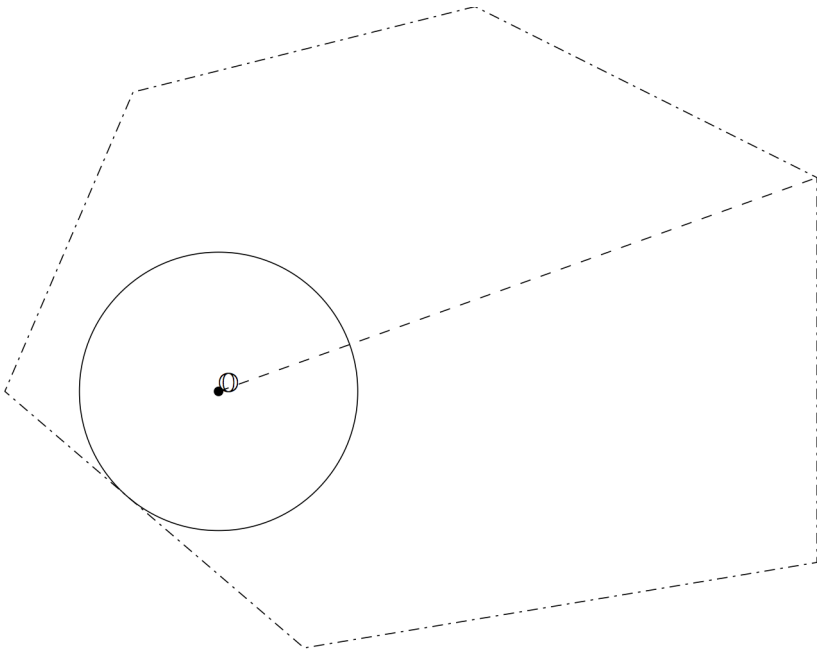


Figure 4.2: A "best" direction

In most cases, the direction of the farthest vertex (see Fig. 4.2) will be a good choice to eliminate more vertices than other directions. The direction is hardly likely to be *the* optimal, but since both the *optimal* direction and the

chosen one are pointing towards the "far" side of the polytope, this should not be a key issue.

4.2.3 Applying the cutting plane

A cutting-plane H will divide the space into two half spaces: H^+ and H^-. In our model, H^+ contains $\mathbb{O}$ and will be the feasible region, while vertices in H^- will be eliminated (see Fig. 4.3).

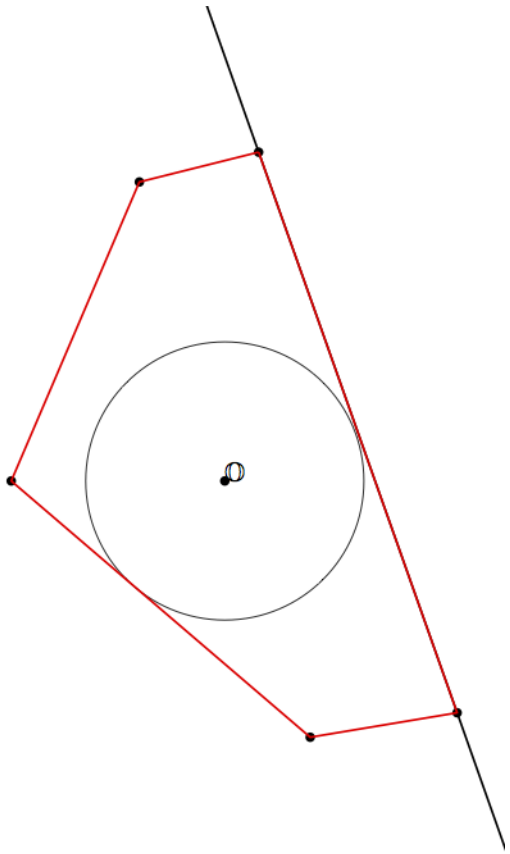


Figure 4.3: After applying the cutting plane

Definition 4.2.1. *Given a polytope P and hyperplane H, let V be the set of vertices of P; define H^+ as the half space with $\mathbb{O} \in H^+$ and H^- as the half space such that $\mathbb{O} \notin H^-$; define $P \cap H^+$ as P^+, $P \cap H^-$ as P^-, and $P \cap H$ as P^0; define $V \cap H^+$ as V^+, $V \cap H^-$ as V^-.*

Definition 4.2.2. *Given a convex polytope P with $\mathbb{O}$ inside P, and choose any facet F of P. Create a sphere S centred at the origin and tangent to the affine hull of F; then S is called a cutting sphere of P.*

Definition 4.2.3. *Given a convex polytope P with $\mathbb{O}$ inside P, define the minimal norm of P as the smallest distance from* $\mathrm{bd}(P)$ *to $\mathbb{O}$, denoted* $\mathrm{mnorm}(P) = \min_{p \in \mathrm{bd}(P)} \|p\|$.

Theorem 4.2.1. *Suppose we are given a convex polytope P and a cutting sphere S; create a hyperplane H which supports S. Then* $\mathrm{mnorm}(P^+) = \mathrm{mnorm}(P)$.

Proof. First, because P is a closed subset of $\mathbb{R}^d$, there exists a point $p_0 \in bd(P)$ such that $\|p_0\| = \mathrm{mnorm}(P)$, which means that $\forall p \in bd(P), \|p\| \geq \|p_0\|$. After applying H, $bd(P)$ is the union of three types of facets: F^+, F^0 and F^-, where:

F^+: $\forall p \in F^+$, $p \in H^+$;

F^-: $\forall p \in F^-$, $p \in H^-$;

F^0: $\exists p_1$ and $p_2 \in F^0$, such that $p_1 \in H^+$ and $p_2 \in H^-$.

Meanwhile, $bd(P^+)$ also consists of three types of facets: F_H^+, F_H^0 and H_H^0, where:

$F_H^+ = F^+$;

$F_H^0 = F^0 \cap H^+$;

$H_H^0 = H \cap P$.

Depending on the choice of H, facets of certain types might not exist.

We know that $\forall p \in F_H^+$, $\|p\| \geq \|p_0\|$, and $\forall p \in F^0$, $\|p\| \geq \|p_0\|$, since $F_H^+ \subset bd(P)$ and $F_H^0 \subset bd(P)$.

And for H_H^0, since H supports S at the minimum norm point p of H_H^0, say we have $\|p\| = r$ where r is the radius of S. From the definition of S, $\exists p' \in bd(P)$, such that $\|p'\| = r$, hence $r \geq \|p_0\|$, therefore p_0 is not cut off by H since it is inside S.

As a conclusion, $\forall p \in bd(P^+)$, $\|p\| \geq \|p_0\|$. Since $p_0 \in S$, we conclude that $\text{mnorm}(P^+) = \text{mnorm}(P)$. □

4.3 Algorithm and Implementation

The algorithm is as follows:

Algorithm 1 Cutting-plane method

repeat
- Determine a reference facet F of polytope P
- Determine a reference direction $\vec{d}$
- Create a cutting sphere S such that S supports F
- Create a cutting plane H such that $H \perp \vec{d}$ and H supports S
- Calculate intersection points of H and P as V'^0
- Use $V' = V'^0 \cup V^+$ to form a new polytope $P' = P \cap H^+$

until There is no obvious improvement from P to P'

4.3.1 Determine a reference facet

An optimal facet should be a facet which is closest to $\mathbb{O}$, unfortunately this optimal facet is just the solution we are trying to get. Instead an alternative approach is to choose a closest facet from a subset of all facets.

A straightforward method is to "randomly" sample a set of facets and choose a smallest distance from them to $\mathbb{O}$. In the following algorithm, we are using the lrs library [1] to enumerate the facets of P using its internal reverse search method [2, 5, 3]. We will not describe the details of the reverse search, however it is generally faster than plain enumeration because the reverse search method goes through only one iteration of samples instead of the backtracking with redundancy.

Algorithm 2 Find a reference facet

Ensure: the best facet of the first 100 samples
  Get first basis of lrs_data
  repeat
    for $i = 0$ to sizeof lrs_dict **do**
      if Sample size has reached 100 **then**
        break to the outer loop
      end if
      if Get the next solution F **then**
        Add 1 to sample size
        Calculate the mnorm(F)
        if mnorm(F) $<$ current smallest distance **then**
          Store mnorm(F) and F
        end if
      end if
    end for
  until not Get the next basis of lrs_data

After sampling first 100 facets, the stored facet F is the reference facet.

4.3.2 Determine a reference direction

In contrast to the reference facet, there is no "optimal" direction. But in most cases, the "longest axis" should be good enough. Here "longest axis" means the direction from a point p in the border of polytope P to $\mathbb{O}$ which has the longest distance from $\mathbb{O}$ than any other points of $\mathrm{bd}(P)$. Here we claim that this point must be some vertex of P. In order to prove this, we need to introduce the concept of "relative interior".

Definition 4.3.1. *The* relative interior *of a convex set is a refinement of the concept of interior. The relatively interior of a convex set contains all points of the set which are not on the relative boundary of the set. For any non-empty convex set $C \subseteq \mathbb{R}^n$, the relative interior* $\mathrm{relint}(C) = \{x \in C \mid \forall y \in C\ \exists \lambda > 1,\ \lambda x + (1-\lambda)y \in C\}$.

Lemma 4.3.1. *Given polytope P, define $V(P)$ as the set of vertices of P; if $\exists p \in \mathrm{bd}(P)$ and $\forall p' \in \mathrm{bd}(P)$, $\|p\| \geq \|p'\|$, then $p \in V(P)$.*

Now we can prove this lemma by contradiction.

Proof. If $p \notin V(P)$, say there exists face $f_0 \subset P$, where $p \in \mathrm{relint}(f_0)$, draw a sphere S with its center of $\mathbb{O}$ and S touches p. Since f_0 itself is a polytope, there must exist some vertex $p' \in f_0$, such that $\|p'\| > \|p\|$, which leads to conflict. □

Since the P itself is represented as a convex hull of vertices, the reference direction can be calculated in $O(n)$ time, where n is the number of vertices of P.

Algorithm 3 Find a reference direction

Ensure: The farthest vertex from $\mathbb{O}$
  for $i = 0$ to number of vertices of P **do**
    Calculate the distance of $V[i]$ to $\mathbb{O}$, $\|V[i]\|$
    if $|V[i]| > d$ **then**
      $d = |P[i]|$, $\vec{v} = P[i]$
    end if
  end for

After this simple algorithm, $\vec{v}$ will be the reference direction.

4.3.3 Generate new polytopes

The cutting sphere S itself does not appear in the code, since we only need its radius. Using this radius r and the reference direction $\vec{v}$, a cutting plane H can be generated.

The new polytope P' is also represented as the convex hull of vertices V'. Note that $V' = V^+ \cup V^0$, where V^+ can be solved by intersecting V and H^+, and V^0 is the intersection of H with the edges of P.

The cdd library provide a function to calculate the edges of a $\mathcal{V}$-polytope P but it is too expensive in time. Here the only edges that we need are those that are intersected with H. The following is another method to get these edges.

In this algorithm we claim that if the middle point of vv' is not strictly inside

Algorithm 4 Find intersection points

```
for all v ∈ V⁺ do
  for all v′ ∈ V⁻ do
    Get middle point v″ of v and v′
    if not v″ strictly inside P then
      Insert H ∩ v⃗v′ into V⁰
    end if
  end for
end for
```

P, then vv' is a candidate edge. However this is not always a true edge because vv' could also be a diagonal of a facet. This does improve performance since many of the pairs could be determined from a linear program instead of an intersection calculation. This however will introduce dummy points to the generated polytope, which does not affect the $\mathcal{V}$-representation, but causes the waste of time in the next iteration.

4.4 Discussion

By applying a cutting hyperplane to the polytope, useless facets and vertices are eliminated from the polytope. After several iterations, the remaining polytope is simpler than the original one, which will reduce the time of enumerating the facets.

4.4.1 Drawbacks

The first drawback of this method is that the reference facet is calculated from a uncontrollable sample, which means that the facet might not be close enough to $\mathbb{O}$, in another words, it might be a bad reference. However we cannot really determine this on the fly.

Another drawback is that during different iterations, many of the "good" vertices and facets remains the same, as well as the dummy points, which means that these vertices will be calculated several times without any changes.

The third drawback is that after several iterations, the remaining polytope will not have an eccentricity as high as the original $P - Q$, thus there won't be much improvement after only a few iterations.

In the next chapter, another method will be introduced to address these drawbacks.

Chapter 5

Cutting-polytope Method

5.1 Basic Idea

The basic idea of the cutting-polytope method (see Fig. 5.1) is to use a polytope to prune $P - Q$ instead of the cutting plane in the previous method. In order to avoid the previously discussed drawbacks of the cutting-plane method, some details must be discussed.

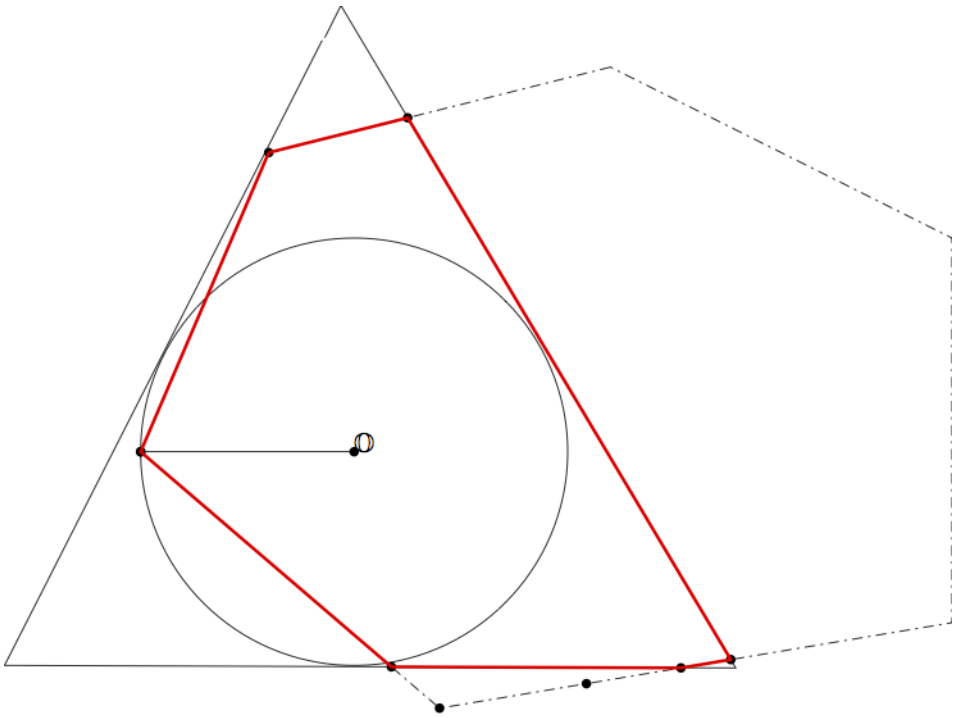


Figure 5.1: Cutting-polytope method

5.1.1 Radius of the cutting sphere

In order to trim off a maximal part of $P - Q$, the goal of choosing the cutting sphere is to make its radius as small as possible; on the other hand it should not be smaller than $\|r\|$ in equation (3.4).

In this method, the radius (see Fig. 5.2) is defined as:

$$\min \|v\| \tag{5.1}$$

$$\text{such that } v \in V,$$

where V is the set of vertices of $P - Q$.

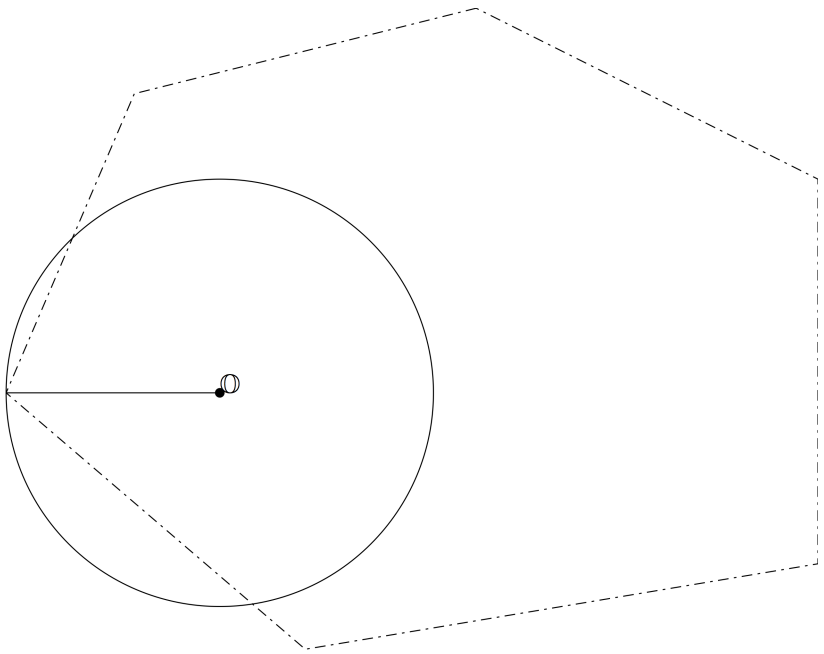


Figure 5.2: Radius of cutting sphere

In some rare cases, this new radius might not prove as useful as the "reference facet", but other than those rare cases, it is likely to be a good choice.

5.1.2 Non-iterative approach

Assuming that the number of facets of the cutting polytope is f, one iteration of the cutting-polytope method can be regarded as "equivalent" to f iterations of cutting-plane method. But since there's only one iteration, much of the duplicated calculation in the former method is avoided.

5.1.3 Aspects of the cutting-polytope

After generating the cutting-polytope R, it will be intersected with $P - Q$. The intersection is also a convex polytope, which is what we need. The intersection's $\mathcal{V}$-representation is a union of three subsets, i.e. $V((P-Q) \cap R) = V_1 \cup V_2 \cup V'$, where $V_1 = \{v \mid v \in V(P-Q) \text{ and } v \in R\}$, $V_2 = \{v \mid v \in V(R) \text{ and } v \in P-Q\}$, and $V' = \{v \mid v \in bd(P-Q) \cap bd(R) \text{ and } v \text{ is extreme point of the intersection}\}$. Here V' can still be represented as $V' = V_3 \cup V_4$, where $V_3 = \{v \mid v \in E(P-Q) \text{ and } v \in \text{bd}(R)\}$, and $V_4 = \{v \mid v \in E(R) \text{ and } v \in \text{bd}(P-Q)\}$. Here $E(C)$ denotes the set of all edges of polytope C.

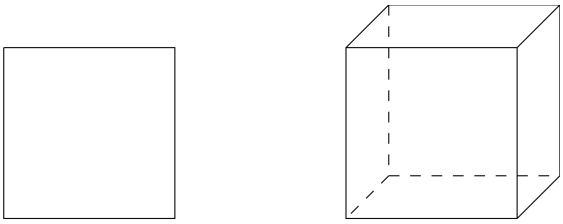
Even though a hypercube is a common choice for a cutting polytope, it is not a good choice for our method. And to choose an appropriate cutting-polytope, two aspect of the polytope should be considered: "*shape*" and "*type*". The following sub sections will explain the reason a hypercube is not a good choice and find a better choice than the a hypercube.

5.1.3.1 "Shape"

Intuitively, from the above discussion we can infer that to simplify calculations, we need to reduce the cardinality of $V(R)$, $E(R)$, and the facet set $F(R)$. Thus the number of vertices, edges and facets of R should be as small as possible. Since $\|E(R)\| \leq \|V(R)\|^2$, only the number of vertices and facets of R will be considered.

Definition 5.1.1 (orthotope)**.** *An orthotope (or a hyperrectangle) (see Fig.*

5.3) is a cartesian product of intervals. It is the generalization of a rectangle to higher dimensions. The special case in which all intervals have equal length, is called the n-hypercube. Clearly orthotopes exist in all dimensions. Each facet of an orthotope is an orthotope of one less dimension.



$n = 2$
$v = 4$
$f = 4$

$n = 3$
$v = 8$
$f = 6$

Figure 5.3: Orthotopes

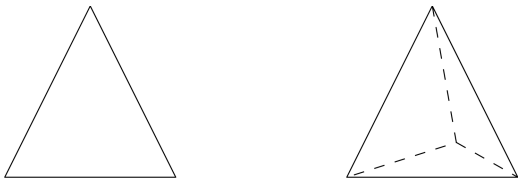
Definition 5.1.2 (orthoplex)**.** *An orthoplex (cross-polytope or hyperoctahedron) (see Fig. 5.4) is a regular convex polytope in of dimension n. Its vertices are all the permutations of* $(\pm 1, 0, 0, \ldots, 0)$*, so the orthoplex is the convex hull of its vertices. It is the generalization of the square and octahedron to higher dimensions. Its facets are simplices of dimension* $n - 1$*.*

$n = 2$
$v = 4$
$f = 4$

$n = 3$
$v = 8$
$f = 6$

Figure 5.4: Orthoplices

Definition 5.1.3 (simplex)**.** *Suppose* $n+1$ *points* $p_0, \ldots, p_n \in \mathbb{R}^n$ *are affinely independent. The n-simplex (see Fig. 5.5) the convex hull of these points:* $C = \{\theta_0 p_0 + \cdots + \theta_n p_n \mid \theta_i \geq 0, 0 \leq i \leq n, \sum_{i=0}^{n} \theta_i = 1\}$*. A simplex is a generalization of the triangle and tetrahedron to higher dimensions. It has the property that any two vertices define an edge of the simplex. Each of the* $n-1$ *facets is itself a simplex of dimension* $n-1$*. A regular simplex is one whose edges have identical lengths.*



$n = 2$
$v = 3$
$f = 3$

$n = 3$
$v = 4$
$f = 4$

Figure 5.5: Simplices

In n dimensions, an orthotope has $2n$ facets and 2^n vertices; an orthoplex has 2^n facets and $2n$ vertices; while a simplex has only $n+1$ facets and $n+1$ vertices. Thus a simplex is a good choice for the cutting polytope.

5.1.3.2 "Type"

Besides randomly generated simplices, two types of simplices can be implemented easily, regular simplices and corner-of-cube.

In its mathematical representation, a corner-of-cube is better because the coordinate values for the vertices have simple formulae. However since the

cutting-polytope is a non-iterative approach, the goal after cutting is to make the center of $(P - Q) \cap R$ as close to $\mathbb{O}$ as possible. Therefore, we choose a regular simplex as the cutting-polytope.

5.2 Algorithm and Implementation

The algorithm of the cutting-polytope method is as follows:

Algorithm 5 Cutting-polytope method

Determine a reference vertex v^*
Create a cutting sphere S with its radius from $\mathbb{O}$ to v^*
Create a regular simplex R with S circumscribed inside
Calculate V_1, V_2, V_3 and V_4.
Calculate $V = \bigcup_{i=1}^{4} V_i$

5.2.1 Determine a reference vertex

As discussed in section 5.1.1, the reference should be the vertex of $P - Q$ which is the closest to $\mathbb{O}$.

Algorithm 6 Determine a reference vertex

Require: V is the set of vertices of $P - Q$
Require: $v^* \in V$
Ensure: $\|v^*\| = \text{mnorm}(V)$
  for all $v \in V$ **do**
    record $\|v\|$ as d
    if $d < \|v^*\|$ **then**
      replace v^* with v
    end if
  end for

5.2.2 Create cutting sphere

This is virtually a dummy step. The only required information about the sphere is its radius, which is $\|v^*\|$.

5.2.3 Create cutting simplex

Since R is a regular simplex with $\mathbb{O}$ as its centroid, it has two properties:

$$\forall v \in V(R),\ \|v\| = |V(R) - 1|\|v^*\| \tag{5.2}$$

and

$$\sum_{i=1}^{|V(R)|} v_i = \vec{0} \tag{5.3}$$

With these two properties we can generate a regular simplex in some fixed rotation. In this implementation, v_1 is always located on the first axis.
Take a 4-simplex supporting an $r = 1$ sphere for example. Here $d = 4, n = 5$, and $l = 4$. The vertices of the simplex are defined by the rows of:

$$S_0 = \begin{bmatrix} x_1 & y_1 & z_1 & w_1 \\ x_2 & y_2 & z_2 & w_2 \\ x_3 & y_3 & z_3 & w_3 \\ x_4 & y_4 & z_4 & w_4 \\ x_5 & y_5 & z_5 & w_5 \end{bmatrix}$$

Algorithm 7 Generate a regular simplex

Require: dimension d
Require: $r = \|v^*\|$
Ensure: R is some regular simplex
  get number of vertices $n = d + 1$
  get the value of l as $l = C = (n-1)r$
  set coordinate of v_1 as $[l, 0, 0, \ldots, 0]$
  for i = 1 to d **do**
    for all v from v_{i+1} to v_n **do**
      set i^{th} coordinate of v as $k = -l/(n-i)$
    end for
    if $i \neq d$ **then**
      let C be the square sum of known coordinates of v_{i+1}
      let $l' = \sqrt{l^2 - C}$
      set $(i+1)^{th}$ coordinate of v_{i+1} as l'
      update l for the next iteration as l'
      set remaining coordinate of v_{i+1} as 0s
    end if
  end for

First, set the coordinates of v_1:

$$S_1 = \begin{bmatrix} 4 & 0 & 0 & 0 \\ x_2 & y_2 & z_2 & w_2 \\ x_3 & y_3 & z_3 & w_3 \\ x_4 & y_4 & z_4 & w_4 \\ x_5 & y_5 & z_5 & w_5 \end{bmatrix}$$

During the first iteration, set the coordinates of each $x_k = -4/(5-i), k =$

$2, 3, 4, 5, i = 1$:

$$S_1' = \begin{bmatrix} 4 & 0 & 0 & 0 \\ -1 & y_2 & z_2 & w_2 \\ -1 & y_3 & z_3 & w_3 \\ -1 & y_4 & z_4 & w_4 \\ -1 & y_5 & z_5 & w_5 \end{bmatrix}$$

and set $y_2 = \sqrt{4^2 - {x_2}^2}$, leaving the other coordinates of v_2 be 0:

$$S_2 = \begin{bmatrix} 4 & 0 & 0 & 0 \\ -1 & \sqrt{15} & 0 & 0 \\ -1 & y_3 & z_3 & w_3 \\ -1 & y_4 & z_4 & w_4 \\ -1 & y_5 & z_5 & w_5 \end{bmatrix}$$

Repeat the same iteration as follows:

$$S_3 = \begin{bmatrix} 4 & 0 & 0 & 0 \\ -1 & \sqrt{15} & 0 & 0 \\ -1 & -\frac{\sqrt{15}}{3} & \frac{2\sqrt{30}}{3} & 0 \\ -1 & -\frac{\sqrt{15}}{3} & z_4 & w_4 \\ -1 & -\frac{\sqrt{15}}{3} & z_5 & w_5 \end{bmatrix}$$

$$S_4 = \begin{bmatrix} 4 & 0 & 0 & 0 \\ -1 & \sqrt{15} & 0 & 0 \\ -1 & -\frac{\sqrt{15}}{3} & \frac{2\sqrt{30}}{3} & 0 \\ -1 & -\frac{\sqrt{15}}{3} & -\frac{\sqrt{30}}{3} & \sqrt{10} \\ -1 & -\frac{\sqrt{15}}{3} & -\frac{\sqrt{30}}{3} & w_5 \end{bmatrix}$$

$$S_5 = \begin{bmatrix} 4 & 0 & 0 & 0 \\ -1 & \sqrt{15} & 0 & 0 \\ -1 & -\frac{\sqrt{15}}{3} & \frac{2\sqrt{30}}{3} & 0 \\ -1 & -\frac{\sqrt{15}}{3} & -\frac{\sqrt{30}}{3} & \sqrt{10} \\ -1 & -\frac{\sqrt{15}}{3} & -\frac{\sqrt{30}}{3} & -\sqrt{10} \end{bmatrix}$$

Now S_5 describes the desired regular simplex.

5.2.4 Intersection points

Here we have V_3 and V_4 represented as $V_3 = \{v \mid v \in E(P - Q)$ and $v \in bd(R)\}$, and $V_4 = \{v \mid v \in E(R)$ and $v \in bd(P - Q)\}$. For V_3, we can use the simplex R's facets to represent $bd(R)$, however we cannot use this method to get V_4. Thus we need a more general method to get the intersection points of two polytopes.

The solution is a mathematical program. For a directional line segment $v\vec{v'}$, the following program is applied:

$$\begin{aligned} & \min t \\ \text{such that } & \lambda_1 x_1 + \lambda_2 x_2 + \cdots + \lambda_n x_n = (1-t)v + tv' \\ & \lambda_1 + \lambda_2 + \cdots + \lambda_n = 1 \\ & \lambda_i \geq 0 \\ & 0 \leq t \leq 1 \end{aligned} \tag{5.4}$$

We can see this is a linear program by rewriting it in a more standard form:

$$\begin{aligned} & \min t \\ \text{such that } & x_1\lambda_1 + x_2\lambda_2 + \cdots + x_n\lambda_n + (v - v')t = 1 \\ & \lambda_1 + \lambda_2 + \cdots + \lambda_n = 1 \\ & \lambda_i \geq 0 \\ & 0 \leq t \leq 1 \end{aligned} \tag{5.5}$$

In this LP x_i are the vertices of a given polytope P_{any}, and λ_i and t are free variables. If this LP has a solution, then either $v \in P_{any}$ (if and only if $t = 0$), or $(1-t)v + tv'$ is one of the intersection points of the line segment vv' and P_{any}. If there exist two intersection points, then the other one can be found via applying the LP (5.5) to $\vec{v'v}$.

In theory, the line segment vv' could be an edge of one polytope, however as

described in section 4.3.3, it is not necessarily a real edge. Besides, compared to the cutting-plane method, the cutting-polytope method is a non-iterative approach, which means that the dummy points of the remaining polytope have less side effect on performance.

5.3 Discussion

Let us compare the current method with that of chapter 4, the facets of the generated simplex serve the same function as the hyperplanes of the cutting-plane method. However, since those hyperplanes are well defined from the start, unnecessary calculation of the infeasible points and facets is avoided. The three drawbacks of the cutting plane method are also mitigated. A detailed experimental comparison of the two methods will be discussed in the next chapter (since this chapter already discusses the theoretical differences)

.

Chapter 6

Experimental Results

6.1 Experimental dataset

The datasets generated for our experiments was prepared in order to make comparisons in three different aspects. In order to reduce the statistical error, we will not use raw data as the input of different algorithms. The preprocessed Minkowski polytopes will be used as the input datasets.

This polytope is the Minkowski Difference of two randomly generated polytope P and Q via equation (3.4); both P and Q are translated with a random vector $\vec{p}$ and $\vec{q}$. We will not describe the process in detail because it is not part of the comparison.

In this chapter, we will make three different comparisons: per method, per dimension, and per number of vertices. To make this comparison feasible, we have generated four datasets of different scales in each dimension from 5

dimension	vertices			
5	548	1362	2203	3928
6	575	1278	2133	4257
7	621	1242	2267	4028
8	593	1303	2267	3814
9	594	1426	2137	3680
10	572	1337	2125	3983

Table 6.1: Size of datasets

to 10.

Each dataset will be represented as vertices, i.e. $\mathcal{V}$-representation. As is shown in the table (6.1), each column of datasets has similar numbers of vertices, in order to make fair comparisons. The number of vertices are not exactly identical, since we have to remove redundant points after generating the Minkowski polytope; during this process, we cannot control the exact number of vertices, but we managed to get similar number of vertices after some trial.

6.2 Experimental method

To regulate the experiment, we compare three different aspects: dimension, method and number of vertices.

6.2.1 Comparison across different dimensions

In the first comparison, we have calculated the average running time in each dimension and illustrated the result as the following chart. As the dimension

goes up, the running time also goes up, and the time increment from one dimension to the next becomes larger: it took over twice the time in 10 dimensions compared to 9 dimensions.

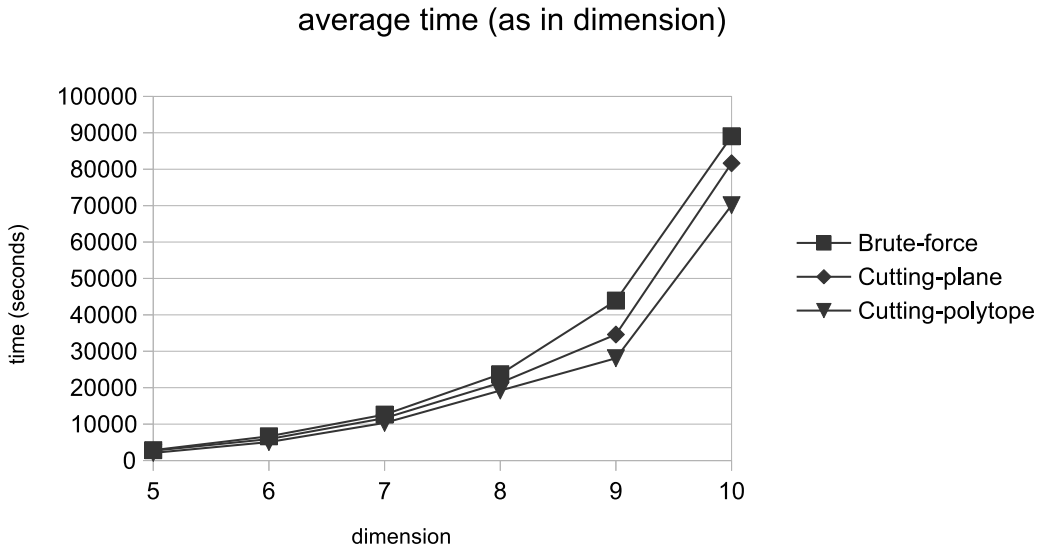


Figure 6.1: Average running time for each dimension

Another result shown on the chart is that in smaller dimensions, the curves are mixed together, but when the dimension goes up, the curves spread out and the cutting polytope method is below both brute-force method and cutting-plane method.

6.2.2 Comparison across different methods

In order to make a more detailed comparison between different methods, we have also illustrated a single chart for each dimension.

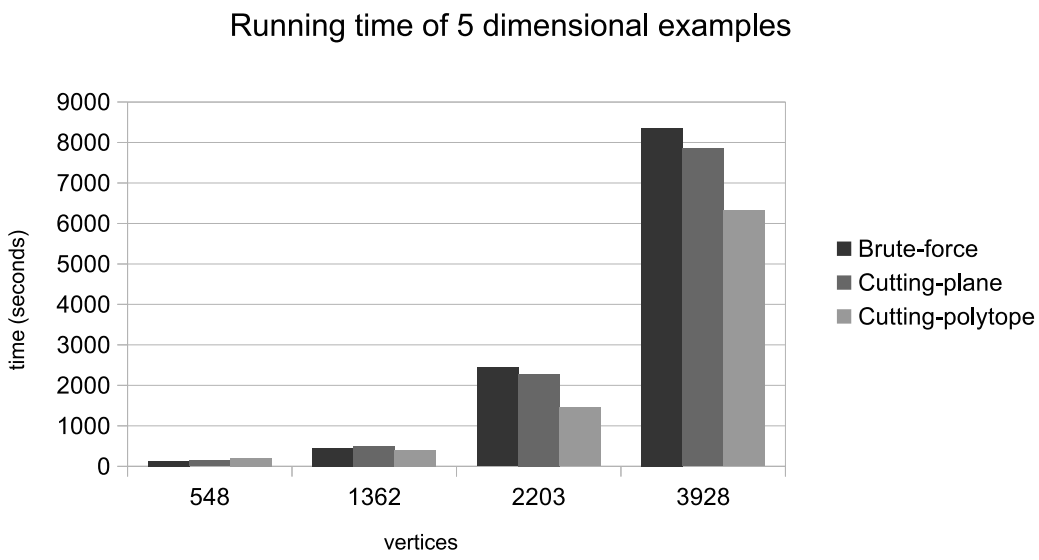


Figure 6.2: Running time for 5 dimensions

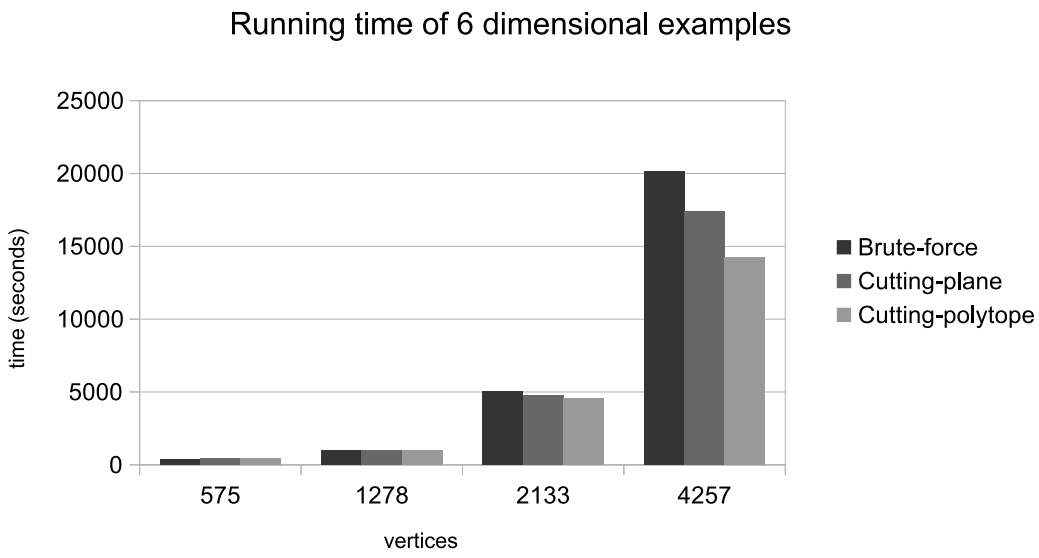


Figure 6.3: Running time for 6 dimensions

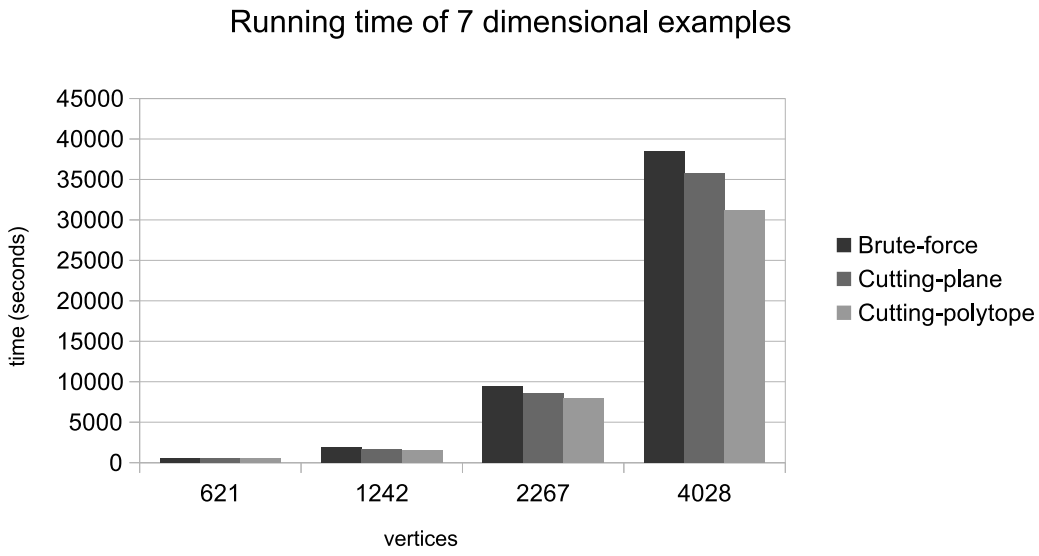


Figure 6.4: Running time for 7 dimensions

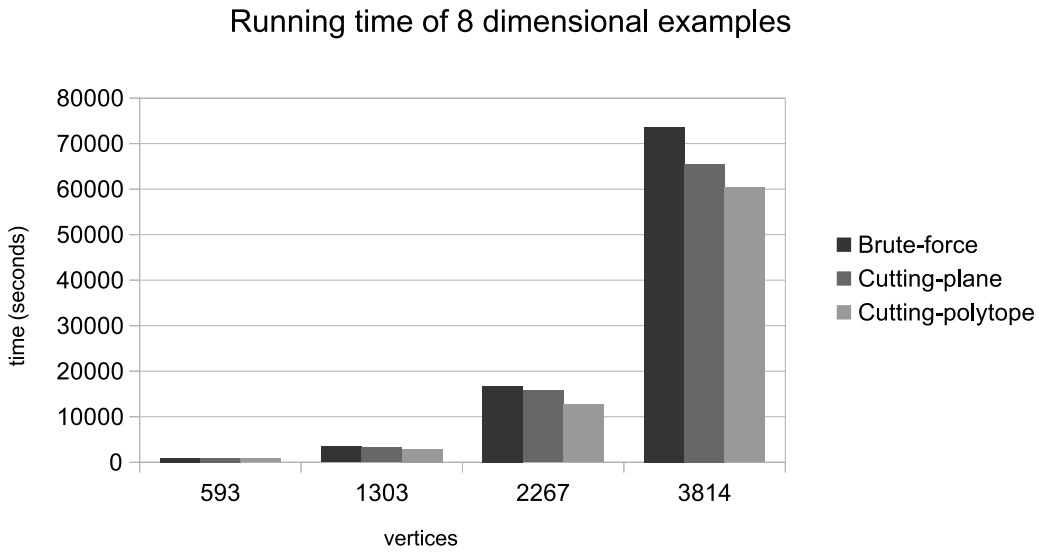


Figure 6.5: Running time for 8 dimensions

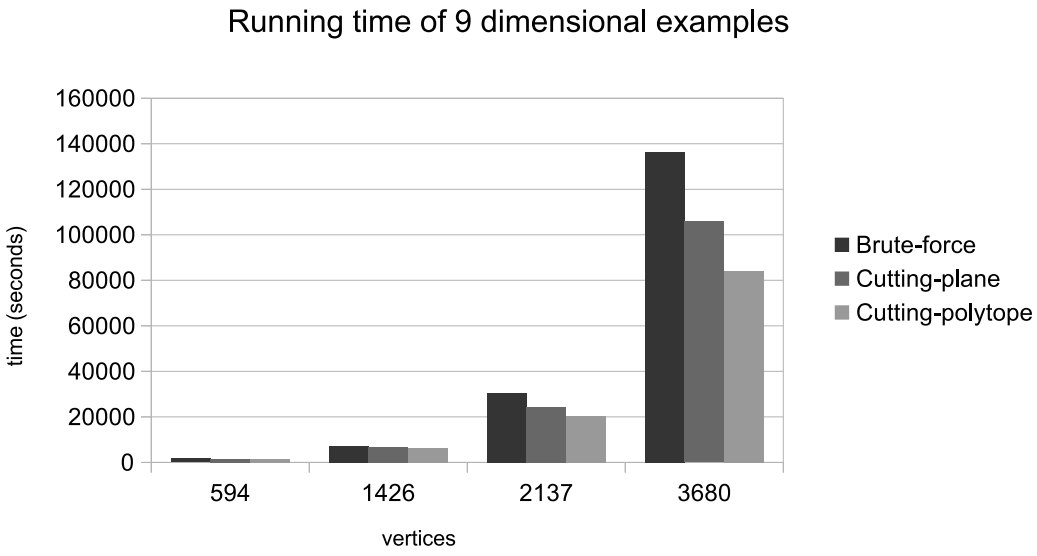


Figure 6.6: Running time for 9 dimensions

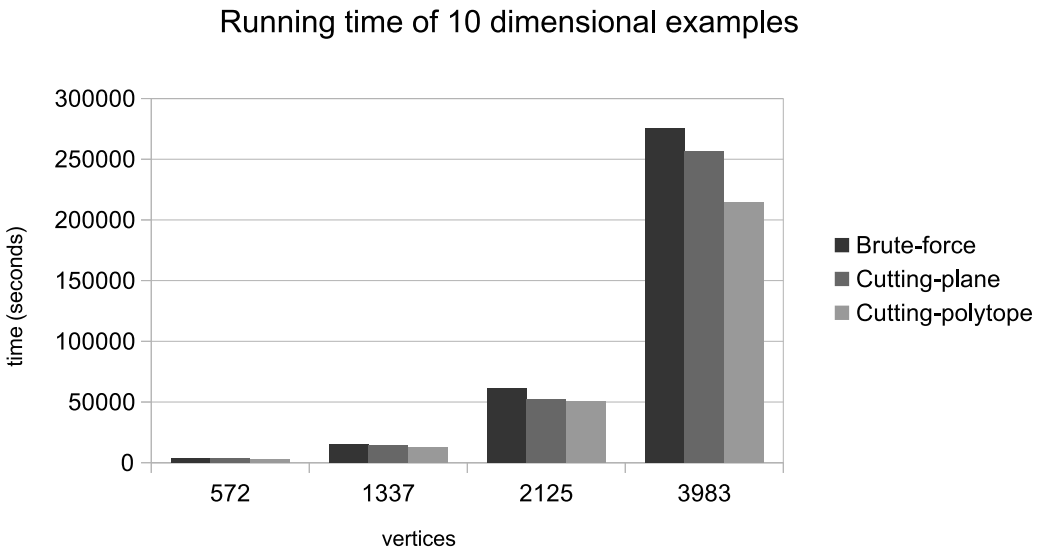


Figure 6.7: Running time for 10 dimensions

From the charts we can see that on large datasets, the brute-force method always took the most time and the cutting polytope method always took the least time. Not all of them have large improvements but in some cases like in 9 dimensions, nearly one third of time is saved from the cutting-polytope method compared to the brute-force method.

6.2.3 Comparison across different vertex numbers

The last experiment is to make a comparison between different vertex numbers. In contrast to the other two experiments, this experiment is only a rough comparison, because it is not easy to make datasets in different dimensions have the same number of vertices. But we managed to make similar number of vertices as shown in each column in table (6.1). We have plotted the following charts for both cutting plane method and cutting polytope method:

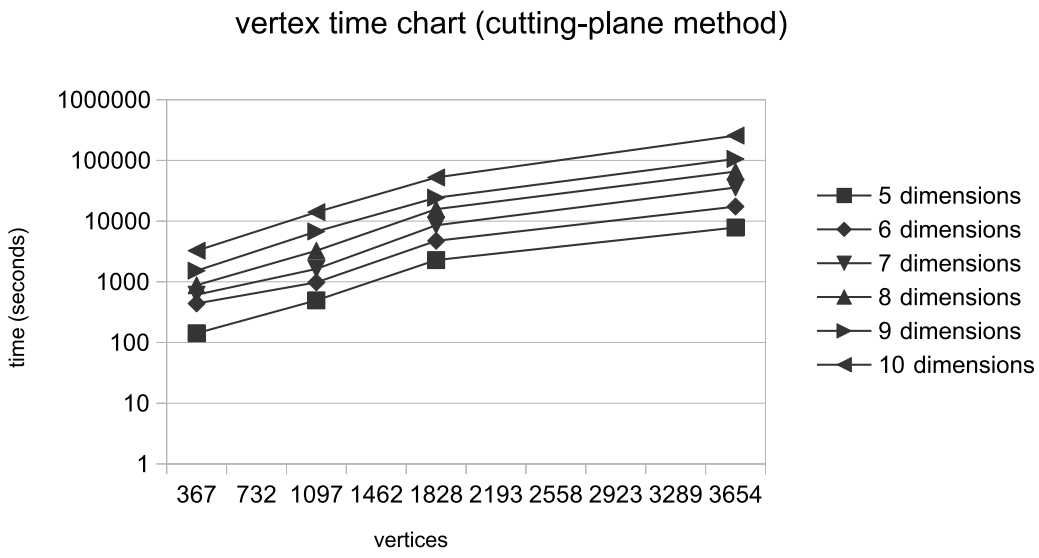


Figure 6.8: Running time for cutting plane method

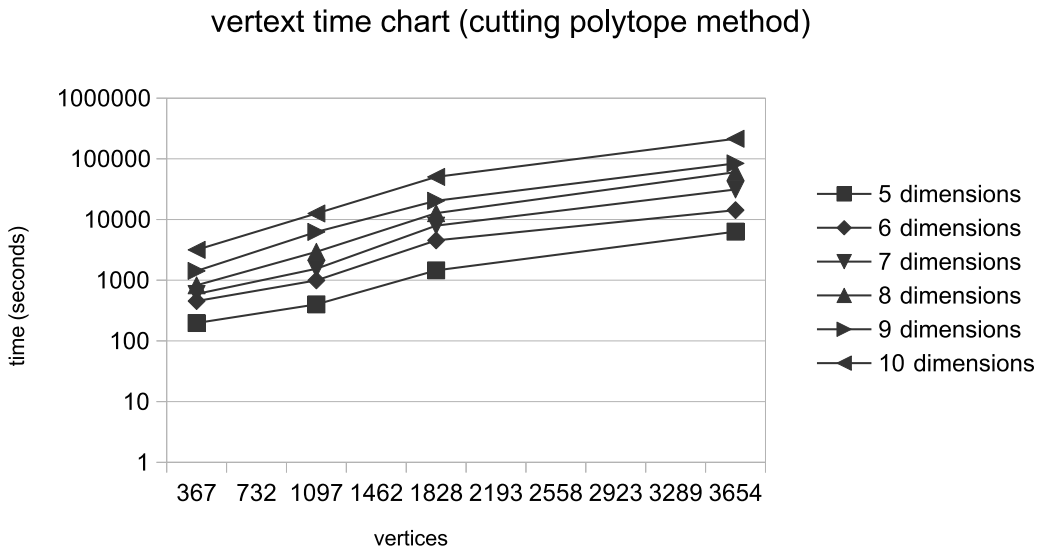


Figure 6.9: Running time for cutting polytope method

Note that the time axis of both charts is in logarithmic scale. From both charts we can see that the six curves are separated clearly. The sub-linear curves show that running time increases more rapidly when the number of vertices becomes larger.

6.3 Results

From the charts shown above we can get the following results:

- The running time increases when either the dimension or number of vertices increases;
- The rate of growth of the running time grows more quickly when either the dimension or vertex numbers increases;
- The brute-force method developed in Cui's [6] thesis takes the most time, the cutting plane method has some improvements, the cutting polytope method is consistently the fastest method;
- All three methods have similar growth curves, meaning they have similar complexity.

Chapter 7

Conclusion and Future Work

7.1 Summary and Conclusion

In this thesis we have discussed how a SVM problem with dataset interference is converted into a geometric problem by converting it into a single polytope via the Minkowski difference and finding the minimum norm point on the boundary of that polytope. We also introduced two different methods for solving the polytope problem: the cutting plane method and the cutting polytope method. Both methods find the facet of the polytope closest to the origin, the difference between the methods is the trimming method. In Chapter 6, we have compared both methods with the brute-force method. From the comparison, we can see that even though the shape of the size-time curve does not change too much, the cutting plane method is faster than brute-force method, and the cutting polytope method is even faster. All

three methods have the same accuracy since the results of all three methods are the same – the closest facet of the polytope to the origin.

As with other computational geometry methods, the algorithms in this thesis share some common advantages and disadvantages. Compared to decomposition methods, they consume more memory, but are relatively faster. This is particularly true for the cutting-polytope method, which does not have the time consuming iterations in both decompositional methods and cutting-plane methods. These algorithms do not have competitive performance compared to those of the approximation methods which are used in most commercial products; but they do have better accuracy, which is a common benefit of all non-approximation methods. Moreover, both the cutting-plane method and cutting-polytope method generally address the drawback of typical geometrical methods, which fail when classification violations are allowed.

7.2 Open Questions and Future Work

In the cutting polytope method, we have applied the simplex as the "cutting model", as it has the least vertices and facets among all polytopes of a given dimension. This simplifies the calculations. However there is another aspect which will affect the calculation time: the simplex has "sharp" angles which will tend to increase the calculation time, since more vertices are left uncut-off in these angles. So one question is to find a polytope or a family of

polytopes which somehow achieve a better balance between size, shape and volume.

In this thesis, both methods do not change the shape of the size-time curve compared to the brute-force method. As a result, if the dimension or the size of vertex numbers is large, it will still take a lot of time in calculation. The reason is that after trimming down enough vertices, the remaining parts will become a polytope whose center is near to the origin. No matter how many more trimming iterations are invoked, there will not be much improvement in the complexity of $P - Q$. Thus we are forced to apply the brute-force enumeration on the remaining parts. If there is an approach which does not rely on facet enumeration, one can expect faster calculations.

Last but not least, the datasets in this thesis are artificially generated. The SVM method, however, is generally a practical method of classification problems. Considering how the datasets are generated in section 6.1, the raw data before calculating the Minkowski Difference are naturally divided into two categories by which polytope they belong to. This is the same characteristic which holds for most practical SVM datasets. Thus the methods introduced in this thesis will have their potential usage in real world applications. However, the algorithms might need some tweaks in order to have better performance and accuracy.

Bibliography

[1] David Avis, *lrslib (reverse search vertex enumeration program/ch package)*, `http://cgm.cs.mcgill.ca/~avis/C/lrs.html`.

[2] ———, *lrs: A revised implementation of the reverse search vertex enumeration algorithm*, 1998.

[3] David Avis and Komei Fukuda, *A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra*, (1992).

[4] Kristin P. Bennett and Erin J. Bredensteiner, *Duality and geometry in svm classifiers*, In Proc. 17th International Conf. on Machine Learning, Morgan Kaufmann, 2000, pp. 57–64.

[5] David Bremner, Komei Fukuda, and Ambros Marzetta, *Primal-dual methods for vertex and facet enumeration*, Discrete and Computational Geometry **20** (1998), 333–357.

[6] Yan Cui, *Training SVMs with classification violations by using Minkowski geometry*, Master's thesis, Universisty of New Brunswick, 2011.

[7] Free Software Foundation, *gmplib (the gnu multiple precision arithmetic library),* `http://gmplib.org/`.

[8] Komei Fukuda, *cddlib (c implemented double description library),* `http://www.ifor.math.ethz.ch/~fukuda/cdd_home/`.

[9] Komei Fukuda, *From the zonotope construction to the Minkowski addition of convex polytopes*, Journal of Symbolic Computation **38** (2004), no. 4, 1261–1272.

[10] G. Fung and O. L. Mangasarian, *Proximal support vector machine classifiers*, Proceedings KDD-2001: Knowledge Discovery and Data Mining, August 26-29, 2001, San Francisco, CA (New York) (F. Provost and R. Srikant, eds.), Asscociation for Computing Machinery, 2001, ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-02.ps, pp. 77–86.

[11] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, *A fast procedure for computing the distance between complex objects in three-dimensional space*, Robotics and Automation, IEEE Journal of **4** (1988), no. 2, 193–203.

[12] Elmer G. Gilbert, *An Iterative Procedure for Computing the Minimum of a Quadratic Form on a Convex Set*, SIAM Journal on Control **4** (1966), no. 1, 61–80.

[13] S. Sathiya Keerthi, Shirish Krishnaj Shevade, Chiranjib Bhattacharyya, and K. R. K. Murthy, *A fast iterative nearest point algorithm for support vector machine classifier design.*, IEEE Trans. Neural Netw. Learning Syst. **11** (2000), no. 1, 124–136.

[14] O. L. Mangasarian and David R. Musicant, *Lagrangian support vector machine classification*, Tech. Report 00-06, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, June 2000, ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-06.ps.

[15] B. Mitchell, V. Dem'yanov, and V. Malozemov, *Finding the Point of a Polyhedron Closest to the Origin*, SIAM Journal on Control (1974).

[16] Thomas M. Mitchell, *Machine learning*, 1 ed., McGraw-Hill, Inc., New York, NY, USA, 1997.

[17] John C. Platt, *Advances in kernel methods*, MIT Press, Cambridge, MA, USA, 1999, pp. 185–208.

[18] Ralph T. Rockafellar, *Convex Analysis (Princeton Landmarks in Mathematics and Physics)*, Princeton University Press, December 1996.

[19] Bernhard Scholkopf and Alexander J. Smola, *Learning with kernels: Support vector machines, regularization, optimization, and beyond*, MIT Press, Cambridge, MA, USA, 2001.

[20] J. A. K. Suykens and J. Vandewalle, *Least squares support vector machine classifiers*, Neural Process. Lett. **9** (1999), no. 3, 293–300.

[21] Christophe Weibel, *Minkowski sums of polytopes: combinatorics and computation*, Ph.D. thesis, Lausanne, 2007.

[22] P. Wolfe, *Finding the nearest point in a polytope*, Math. Progr. **11** (1976), no. 1, 128–149.

[23] Philip Wolfe, *A duality theorem for nonlinear programming*, Quarterly of applied mathematics **19** (1961), no. 3.

[24] Günter M. Ziegler, *Lectures on polytopes*, ch. 2, p. 65, Springer, 1995.

Vita

Candidate's full name: Zhan Gao

University attended (with dates and degrees obtained):
2011 - 2014 : University of New Brunswick, Master of Computer Science
2010 - 2011 : Southeast University (Suzhou campus), joint program with University of New Brunswick
2006 - 2010 : Southeast University, Bachelor of Software Engineering

Conference Presentations:
OPDE 2013, Nearest Facet of a Higher Dimensional Convex Hull to an Inner Point