FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Optimization Methods in Knowledge Engineering
Student:	Vladislav Stankov
Supervisor:	Ing. Tomáš Kalvoda, Ph.D.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2019/20

Instructions

1. Get acquainted with the mathematical formulation of a general optimization problem. Summarize basic classes of these problems (linear, quadratic, convex, non-linear with various constraints put on optimization variables, etc.).

2. Review particular practical tasks belonging to the realm of knowledge engineering and amenable to optimization methods.

3. Review existing mathematical methods for solving optimization problems from point 1 of this assignment.

4. Explore available software tools (software packages, libraries) able to solve optimization problems. Emphasize open-source tools developed in Python and Julia communities. Carry out a benchmark of a selected set of tools on a suitable set of problems.

References

Boyd S., Vandenberghe L., Convex Optimization, Cambridge University Press, 2009
Sra S., Nowozin S., Wright S. J., Optimization for Machine Learning, Massachusetts Institute of Technology, 2012

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague December 2, 2018

Acknowledgements

I am profoundly grateful to my supervisor Tomáš Kalvoda for his inspiration and advice in the writing this thesis. His excellent knowledge of the subject, valuable comments and positive feedback helped me to immensely improve this work.

Declaration

In Prague on May 16, 2019 . . . .

Citation of this thesis

Stankov, Vladislav. *Optimization Methods in Knowledge Engineering.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Tato práce se zabývá problematikou matematické optimalizace a ukazuje, jak ji lze aplikovat na problematiku znalostního inženýrství. Nejdříve se zaměříme na teoretické základy matematické optimalizace a formulujeme základní optimalizační problémy jako lineární programování, kvadratické programování či optimalizaci vektor]u. Dále se zaměříme na vybrané problémy z oblasti znalostního inženýrství a vyjádříme je v rámci optimalizace. Dále prezentujeme metody řešení optimalizačních problém]u, konkrétně metody sestupu, Newtonovu metodu a bariérovou metodu. Nakonec se přesuneme do praktické části, kde demonstrujeme r]uzné nástroje pro řešení optimalizačních úloh.

Klíčová slova optimalizace, konvexní optimalizace, metody konvexní optimalizace, metody sestupu, metoda vnitřního bodu, znalostní inženýrství, strojové učení, vytěžování dát

Abstract

This thesis explore the field of mathematical optimization and show how it can be applied on the problems of knowledge engineering. Firstly, we develop a theoretical background in mathematical optimization and formulate basic optimization problems like linear programming, quadratic programming and vector optimization. Next we select different problems from the domain of knowledge engineering and express them in optimization framework. Then we present methods handling optimization problems, namely descent methods, Newton's method and barrier method. Finally, we move beyond the theoretical part and demonstrate various programming tools for solving optimization tasks.

Keywords optimization, convex optimization, methods of convex optimization, descent methods, interior point method, knowledge engineering, machine learning, data mining

Contents

Citation of this thesis . . . vi

Introduction **1**

1 Mathematical optimization **3**
1.1 Theory . . . 4
1.1.1 Preliminaries . . . 4
1.1.2 Convex sets . . . 6
1.1.3 Convex functions . . . 13
1.2 Convex Optimization . . . 20
1.2.1 Linear programming . . . 23
1.2.2 Quadratic programming . . . 23
1.2.3 Vector optimization . . . 24
1.3 Duality . . . 25
1.4 Non-convex optimization . . . 29
1.5 Reproducing Kernel Hilbert Spaces . . . 29

2 Knowledge Engineering **37**
2.1 Classification . . . 39
2.1.1 Hard SVM . . . 39
2.1.2 Soft SVM . . . 41
2.1.3 Kernel SVM . . . 43
2.2 Regression . . . 44
2.2.1 Ordinary least squares . . . 45
2.2.2 Ridge regression . . . 46
2.2.3 Kernel ridge regression . . . 48
2.2.4 Neural Networks . . . 49
2.3 Clustering . . . 53
2.3.1 K-means learning rule . . . 54
2.3.2 K-means algorithm . . . 56

3 Algorithms **57**
3.1 Unconstrained optimization methods . . . 57
3.1.1 Gradient descent . . . 60
3.1.2 Newton’s method . . . 61
3.2 Equality constrained minimization . . . 67

3.2.1 Newton's method with equality constrains 68
3.3 Inequality constrained minimization 71
3.3.1 Barrier method 71

4 Applications **77**
4.1 Problem specification 77
4.2 Julia 78
4.2.1 Solvers 79
4.2.2 Benchmarks 79
4.3 Python 80
4.3.1 Solvers 81
4.3.2 Benchmarks 81
4.4 Example 83

Conclusion **87**

Bibliography **89**

A Contents of enclosed CD **91**

Introduction

In this thesis, we investigate the connection between optimization and field of knowledge engineering. At first glance the relation looks quite poor, certainly we want to achieve the best results solving any problem, but usually the mathematical part is overlooked. One might think of knowledge engineering as writing domain specific programs solving particular problems, and in fact, there exist tons of such programs based on inexact heuristics, hence showing suboptimal results. Moreover, such heuristic algorithms mostly do not provide us with proof of the solution optimality, so we have to trust the implementation. However, within notion of mathematical optimization framework we would be able to handle these tasks efficiently achieving the guaranteed optimal performance with small computational time.

The thesis contributes to gaining a solid understanding of mathematical optimization methods applied to the demands of knowledge engineering. Thus, motivating us for further development and improvement of computer programs for deriving knowledge based on optimization technics.

Initially, we review mathematical backgrounds of optimization, in particular, convex optimization problems that in general can be solved efficiently. Talking about convex optimization, we formulate specific types of problems like linear optimization, quadratic programming, or vector optimization. Nevertheless, we make some notes on non-convex optimization and show its relation to the convex one. Next, we study various problems in knowledge engineering that can be represented in terms of mathematical optimization. Also, these problems will be assigned to already defined optimization classes. Then, we review existing optimization methods for solving formulated types of problems. In the end, we look at programming constituent, namely on various software tools that can solve optimization tasks, preferably open-source packages. Finally, we take some already defined problems benchmark selected solvers.

Objective

The main goal of this thesis is to show strong relation between two huge fields, namely mathematical optimization and knowledge engineering. Usually, knowledge engineering is considered as writing sophisticated computer programs that cary out non-trivial results, and it turns out, that such programs are mostly derived from mathematical models describing the problem domain.

For our first goal, we formulate such problems in terms of optimization and review mathematical methods that can solve these tasks yielding desired optimal outcomes. We not only analyze the mathematical part, but also show how these problems are solved in terms of the optimization framework. The partial objective is to investigate existing software tools that are able to carry out optimizational computations on earlier formulated problems. As a result, we would eliminate a black box perspective of optimization technics used in knowledge engineering.

Chapter 1

Mathematical optimization

The field of mathematical optimization is surely not a new topic and traces back to the 17th century, when Pierre de Fermat formulated the principle of first derivative vanishment at the function extreme point. Until 20th century optimization problems frequently appeared in physics, for example Newton's minimal resistance problem and principle of least action. As for the algorithms, the first one was least squares method presented in 19th century by Carl Friedrich Gauss and Adrien-Marie Legendre. The main advantage of least squares method is the analytical solution, though in general, optimization problems are solved algorithmically than analytically. Nevertheless, the main optimization theory, namely convex optimization, and algorithms were set on motion in 1950 and by 1970 were quite well developed.

The first widely known general algorithm for solving optimization problems, specifically linear problems, was the simplex method invented by George Dantzig in 1947 at Stanford. The next milestone was the ellipsoid method from 1970s developed by the Soviet and Ukrainian mathematicians. The ellipsoid method was the first algorithm to show that LP (a.k.a. linear optimization a.k.a. linear programming) is solvable in polynomial time. In fact, it was the reason why in late 70s an article related to LP appeared at the front page of New York Times. In 80th an interior method was developed, originally it was used to solve LP problems and later it was realized that this approach can be generalized to convex optimization as well.

Actually, one can notice that the invention of the simplex method coincides with the development of modern digital computers. Since generally optimization problems do not have an analytical solution, they are usually solved by some iterative methods. Thus, potential of solving sophisticated optimization problems is highly correlated with growth of the computer performance.

Although, mathematical optimization is not a new subject, the thing that makes it interesting now is its applications in various fields like finance, circuit design, statistics and machine learning. Moreover, the theory behind convex optimization shows that convex problems can be solved time and space efficiently depending on the input size. As for non-convex problems, it appears that in most cases non-convex problems use convex optimization as its subroutine.

This chapter looks at the theory behind optimization along with most important families of convex and non-convex optimization problems. Then we move on to duality that gives a interesting way of solving even hard problems. Next, some notes on non-convex case will be given. At the end we step aside and introduce theoretical ideas of reproducing kernel Hilbert spaces, that will be also widely used in the next chapter. This chapter is a brief outline of the book [1, Chapters 1-5], the section about Hilbert spaces summarizes [2].

1.1 Theory

1.1.1 Preliminaries

In this subsection we summarize basic theoretical concepts that we will widely use throughout the thesis.

Remark 1.1 (Notation).

- Suppose $x \in \mathbb{R}^n$ is a column vector, we identify vector x by n-tuple of reals as $x = (x_1, \dots, x_n) \in \mathbb{R}^n$. Thus, when we write elements of a column vector in parentheses we do not use the transposition.
- We denote all non-negative and positive real numbers by $\mathbb{R}_+$ and $\mathbb{R}_{++}$, respectively.
- Suppose $\mathbf{X} \in \mathbf{R}^{n \times m}$. We denote the ith columns of $\mathbf{X}$ by $\mathbf{X}_{i,\bullet}$, for the jth row we have $\mathbf{X}_{\bullet,j}$.

$\circ$

Linear algebra

Definition 1.1 (Symmetric matrix)**.** Suppose $\mathbf{A} \in \mathbb{R}^{n \times n}$. We say that $\mathbf{A}$ is symmetric if $\mathbf{A} = \mathbf{A}^T$ and denote this as $\mathbf{A} \in \mathbb{S}^n$ $\circledcirc$

A nice property of symmetric matrices is that they have real eigenvalues.

Definition 1.2 (Positive semidefinite symmetric matrix)**.** Suppose $\mathbf{A} \in \mathbb{S}^n$. If for all $x \in \mathbb{R}^n$ if the following holds

$$x^T \mathbf{A} x \geqslant 0.$$

we call $\mathbf{A}$ a *positive semidefinite* matrix or PSD matrix and denote this vy $\mathbf{A} \in \mathbb{S}^n_+$. If for all non-zeo $x \in \mathbb{R}^n$ we obtain

$$x^T \mathbf{A} x > 0,$$

then $\mathbf{A}$ is called *positive definite* or PD matrix, we denote this by $\mathbf{A} \in \mathbb{S}^n_{++}$. $\circledcirc$

Lemma 1.3 (Characterization)**.** Suppose $\mathbf{A} \in \mathbb{S}^n$.

- $\mathbf{A} \in \mathbb{S}^n_+$ if and only if all eigenvalues of $\mathbf{A}$ are non-negative.
- $\mathbf{A} \in \mathbb{S}^n_{++}$ if and only if all eigenvalues of $\mathbf{A}$ are positive.

Definition 1.4 (Norm)**.** Suppose $\mathcal{X}$ is a vector space. A function $\|\cdot\|_{\mathcal{X}} : \mathcal{X} \to \mathbb{R}$ is called a *norm* if for all $x, y \in \mathcal{X}$ and $\alpha \in \mathbb{R}$ the following holds:

1. Positive definiteness: $\|x\|_{\mathcal{X}} \geqslant 0 \wedge (\|x\|_{\mathcal{X}} = 0 \iff x = 0)$.
2. Positive homogeneity: $\|\alpha x\|_{\mathcal{X}} = |\alpha| \|x\|_{\mathcal{X}}$.
3. Triangle inequality: $\|x + y\|_{\mathcal{X}} \leqslant \|x\|_{\mathcal{X}} + \|y\|_{\mathcal{X}}$. $\circledcirc$

Remark 1.2 (Notation). In the majority of the text our vector space $\mathcal{X}$ will be just $\mathbb{R}^n$, so notation $\|\cdot\|$ will stand for norm defined on $\mathbb{R}^n$, in particular we denote the Euclidean norma as $\|\cdot\|_2$. If we use another vector space $\mathcal{Y}$ it would be explicitly specified by the index, i.e. $\|\cdot\|_{\mathcal{Y}}$ $\circ$

Theorem 1.5 (Cauchy-Schwarz inequality)**.** Suppose $x, y \in \mathbb{R}^n$, then the following inequality holds

$$|x^T y| \leqslant \|x\|_2 \|y\|_2.$$

We have an equality

$$|x^T y| = \|x\|_2 \|y\|_2$$

only if x, y are collinear.

Multivariable functions Lets have a look at basic points related to differentiability of multivariable functions.

Definition 1.6 (Partial derivative)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a real valued function. If $a \in \mathbf{dom}\, f$ and the following limit exits

$$\lim_{h\to 0} \frac{f(a + he_i) - f(x)}{h}, \qquad e_i = (0_1, \ldots, 0_{i-1}, 1_i, 0_{i+1}, \ldots, 0_n),$$

then we say that value of this limit is a *partial derivative of f at a point a* with respect to i-th variable and we always write it as $\frac{\partial f}{\partial x_i}(a)$. ◎

Remark 1.3 (Vector valued functions)*.* Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a vector valued function. Partial derivative of f at a point $a \in \mathbf{dom}\, f$ with respect to x_i is the following vector

$$\frac{\partial f}{\partial x_i}(a) = \Big(\frac{\partial f_1(a)}{\partial x_i}, \cdots, \frac{\partial f_n(a)}{\partial x_m}\Big). \qquad \circ$$

Definition 1.7 (Differentiability)**.** Suppose $f : \mathbb{R}^n \to \mathbb{R}^m$ is a function. We say that f is *differentiable at a point* $a \in \mathbf{dom}\, f, a \neq x$ if and only if there exist a linear map $\mathbf{J} : \mathbb{R}^n \to \mathbb{R}^m$ such that

$$\lim_{x\to a} \frac{\|f(x) - f(a) - \mathbf{J}(x - a)\|_2}{\|x - a\|_2} = 0.$$

We say that f is *differentiable* if its domain is open and it is differentiable at every point of its domain. ◎

Remark 1.4 (The Jacobian matrix and the gradient)*.* In the definition of differentiability of a function we talk about linear mapping $\mathbf{J} : \mathbb{R}^n \to \mathbb{R}^m$, which is defined as a matrix of all first-order partial derivatives of a vector valued function and is called *the Jacobian matrix* $\mathbf{J} \in \mathbb{R}^{n\times n}$, i.e.

$$\mathbf{J}_{ij}(a) = \frac{\partial f_i(a)}{\partial x_j}, \qquad i, j = 1, \ldots, n.$$

If original function f is a scalar valued function, then we talk about vector in $\mathbb{R}^n$. This vector is called *the gradient* of the function f and is denoted as ∇f, i.e.

$$\nabla f(a) = \Big(\frac{\partial f(a)}{\partial x_1}, \cdots, \frac{\partial f(a)}{\partial x_n}\Big). \qquad \circ$$

Definition 1.8 (Second derivative)**.** Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a function. Second derivative or *Hessian matrix* of f at $a \in \mathbf{int}\,\mathbf{dom}\, f$, denoted $\nabla^2 f(a) \in \mathbb{S}^n$ is defined as the following,

$$\big(\nabla^2 f(a)\big)_{ij} = \frac{\partial^2 f(a)}{\partial x_i \partial x_j}, \qquad i, j = 1, \ldots, n.$$

We say that f is twice differentiable if its Hessian exists at each point in the domain of f, which is open. ◎

1.1.2 Convex sets

In this section we start with some basic definitions like affine set, convex set and cones. Next we move on to some important examples like balls, ellipsoids or half-spaces. Then we focus on operations that preserve convexity to formulate so using these operations we can construct new convex sets from other convex sets. After that we look at generalized inequalities or vector inequalities with respect of proper cones.

We begin with important idea of convex sets and other related concepts.

Definition 1.9 (Line segment)**.** Suppose $a, b \in \mathbb{R}^n$ and $a \neq b$. The set $\emptyset \neq C \subseteq \mathbb{R}^n$ such that

$$C = \{y \mid y = \theta a + (1-\theta)b,\ 0 \leqslant \theta \leqslant 1\}$$

is called a *line segment* between a and b. If $\theta \in \mathbb{R}$ then C is called the *line* passing through a and b, ⊚

Definition 1.10 (Convex combination)**.** Let $x_1, \dots, x_m \in \mathbb{R}^n$ and $\theta_1, \dots, \theta_n \in \mathbb{R}_+$ such that $\sum_i \theta_i = 1$. Point y of the from

$$y = \sum_{i=1}^{m} \theta_i x_i,$$

is called a *convex combination* of the points $x_1, \dots, x_n$. If we let θ be any real number we obtain an affine combination. ⊚

In fact, affine and convex combinations are special cases of linear combination with some constrains on the coefficients.

Definition 1.11 (Convex set)**.** Suppose $\emptyset \neq C \subseteq \mathbb{R}^n$. If for every $a, b \in C$ and $a \neq b$ the following condition holds

$$\{y \mid y = \theta a + (1-\theta)b,\ 0 \leqslant \theta \leqslant 1\} \subseteq C,$$

then we call C is a *convex set.* If we let $\theta \in \mathbb{R}$ then C is an affine set. ⊚

The definition tells us that if set C contains every line segment between its two points then it is convex, this situation is illustrated in the Figure 1.1.

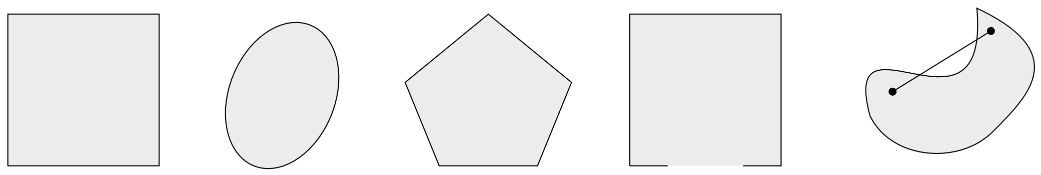


Figure 1.1: Three sets on the left hand side are convex, others are not.

Now we move on to convex cones and conic combinations.

Definition 1.12 (Conic combination)**.** Let $a, b \in \mathbb{R}^n$ and $C \subseteq \mathbb{R}^n$. Any point x of the following form:

$$x = \alpha a + \beta b, \qquad \alpha, \beta \in \mathbb{R}_+$$

is called a *conic combination* of a and b. If C contains all such x, then we call it *convex cone.* ⊚

Geometrically we have a sort of pie slice. Points on boundaries are just non-negative multiples of u or v and points inside the slice are positive multiples of $u + v$, to see this consider the Figure 1.2.

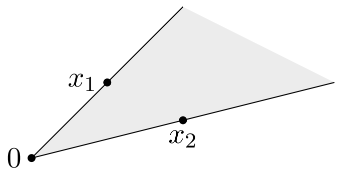


Figure 1.2: Convex cone illustration.

Lets look at some important sets that we will use later in analysis of the domains of optimization problems.

Example 1.5 (Solution set of linear equations)**.** Let $C = \{x \mid \mathbf{A}x = b\}$, where $\mathbf{A} \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$, i.e. C is the *solution set of a system of linear equations.* To show that C is affine, suppose $u, v \in C,\ u \neq v$ and $\theta \in \mathbb{R}$. Lets consider a line passing u, v, so we have

$$\mathbf{A}(\theta u + (1-\theta)v) = \theta\mathbf{A}u + (1-\theta)\mathbf{A}v = \theta b + (1-\theta)b = b. \qquad \bigcirc$$

Example 1.6 (Hyperplanes and half-spaces)**.** Let $C \subseteq \mathbb{R}^n$ be a solution set of a single linear equation, i.e. set of the form

$$C = \{x \in \mathbb{R}^n \mid a^T x = b\}, \qquad a \neq 0.$$

The set C is called a *hyperplane*, with a normal vector a. Notice, when we move around with b we get parallel hyperplanes to the first one.

A solution set of a single linear inequality, i.e. set of the form

$$\{x \in \mathbb{R}^n \mid a^T x \leqslant b\}, \qquad a \neq 0$$

is called a *half-space.*

Both hyperplanes and half-spaces are affine hence convex. Now lets pick a $\theta \in \mathbb{R}$ and verify the definition of an affine set.

$$a^T(\theta u + (1-\theta)v) = \theta a^T u + (1-\theta)a^T v = \theta b + (1-\theta)b = b$$

An example of a half-space and hyperplane is illustrated in the Figure 1.3. $\bigcirc$

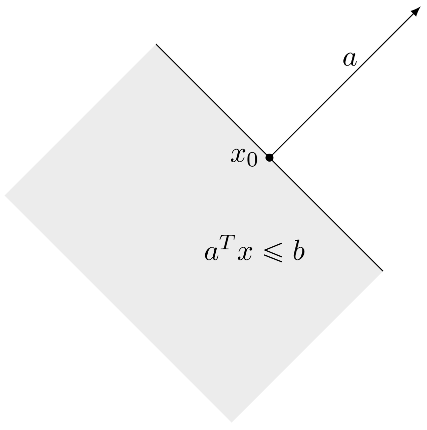


Figure 1.3: Hyperplane $a^T x_0 = b$ and a half-space $a^T x \leqslant b$.

Example 1.7 (Polyhedron)**.** Suppose $\preccurlyeq$ is a component-wise inequality and $a, b \in \mathbb{R}^n$, i.e.

$$a \preccurlyeq b \iff a_i \leqslant b_i, \qquad i \in \{1, \ldots, n\}.$$

A set $P \subseteq \mathbb{R}^n$ defined as a solution set of a finite number of linear inequalities and equations

$$P = \{x \in \mathbb{R}^n \mid \mathbf{A}x \preccurlyeq b,\ \mathbf{B}x = d\}, \qquad \mathbf{A} \in \mathbb{R}^{m\times n},\ \mathbf{B} \in \mathbb{R}^{p\times n},\ b \in \mathbb{R}^m,\ p \in \mathbb{R}^p$$

is called a *polyhedron.* Notice, that a polyhedron can be both bounded and unbounded.Using the fact that intersection preserve convexity (will be shown later) we can say that polyhedron is a convex set. A example of a polyhedron is illustrated in the figure 1.4 $\bigcirc$

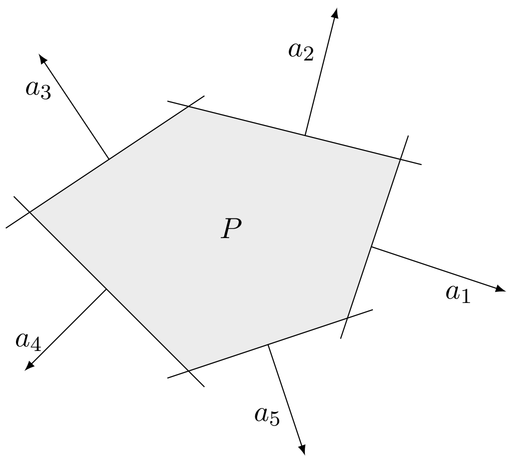


Figure 1.4: Polyhedron.

Example 1.8 (Balls and ellipsoids)**.** The set $B(x_c, r)$ of the form

$$B(x_c, r) = \{x \mid \|x - x_c\|_2 \leqslant r\}$$

is called a *Euclidean ball* with the center x_c and radius r. Generalization of a ball is an *ellipsoid*, i.e. set of the form

$$E = \{x \in \mathbb{R}^n \mid (x - x_c)^T \mathbf{P}^{-1}(x - x_c) \leqslant 1\}, \qquad \mathbf{P} \in \mathbb{S}^n_{++}. \tag{1.1}$$

To see that a ball is a special case of an ellipsoid consider the following

$$\|x - x_c\|_2 \leqslant r \iff \frac{1}{r^2}(x - x_c)^T \mathbf{I}(x - x_c) \leqslant 1.$$

Thus, we have $\mathbf{P}^{-1} = 1/r^2\mathbf{I}$ or equivalently $\mathbf{P} = r^2\mathbf{I}$.

Let us derive an equivalent formulation of an ellipsoid. Since $\mathbf{P} \in \mathbb{S}^n_{++}$ by the spectral theorem for symmetric matrices we have $\mathbf{P} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$, where $\mathbf{Q}$ is an orthogonal matrix, i.e. $\mathbf{Q}\mathbf{Q}^T = \mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ and $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues of $\mathbf{P}$, such that eigenvalues λ_i of $\mathbf{P}$ are positive. Now we define $\mathbf{P}^{1/2} \in \mathbb{S}^n_{++}$ as $\mathbf{P}^{1/2} = \mathbf{Q}\mathbf{\Lambda}^{1/2}\mathbf{Q}^T$. Therefore, $\mathbf{P}$ can be expressed as $\mathbf{P} = \mathbf{P}^{1/2}\mathbf{P}^{1/2}$. Moreover, since $\mathbf{P}$ and $\mathbf{P}^{1/2}$ are positive definite, the inverses exist and we have

$$\mathbf{P}^{-1} = \left(\mathbf{P}^{1/2}\mathbf{P}^{1/2}\right)^{-1} = \mathbf{P}^{-1/2}\mathbf{P}^{-1/2}, \qquad \mathbf{P}^{-1/2}\mathbf{P}^{1/2} = \mathbf{I}.$$

The condition in (1.1) can be reformulated as

$$(x - x_c)^T\mathbf{P}^{-1}(x - x_c) \leqslant 1 \iff \left\|\mathbf{P}^{-1/2}(x - x_c)\right\|_2 \leqslant 1.$$

Now we say that $\mathbf{P}^{-1/2}(x - x_c) = u$ for some $u \in \mathbb{R}^n$ hence $\|u\|_2 \leqslant 1$. Thus, we get

$$x = x_c + \mathbf{P}^{1/2}u.$$

Therefore, taking $\mathbf{A} = \mathbf{P}^{1/2}$ an ellipsoid can be expressed as

$$E = \{x_c + \mathbf{A}u \mid \|u\|_2 \leqslant 1\}, \qquad \mathbf{A} \in \mathbb{S}^n_{++}.$$

In the same way we can represent a ball

$$B(x_c, r) = \{x_c + ru \mid \|u\|_2 \leqslant 1\}.$$

To show that ellipsoid is convex consider the definition and two points
$x_c + \mathbf{A}u,\ x_c + \mathbf{A}v \in E$. Lets check that $\theta(x_c + \mathbf{A}u) + (1-\theta)(x_c + \mathbf{A}v) \in E$:

$$\begin{aligned}\theta(x_c + \mathbf{A}u) + (1-\theta)(x_c + \mathbf{A}v) &= x_c + \theta\mathbf{A}u + (1-\theta)\mathbf{A}v \\ &= x_c + \mathbf{A}(\theta u + (1-\theta)v).\end{aligned}$$

Now we verify that $\|\theta u + (1-\theta)v\|_2 \leqslant 1$. Because $x_c + \mathbf{A}u,\ x_c + \mathbf{A}v \in E$ we have

$$\|\theta u + (1-\theta)v\|_2 \leqslant \theta + (1-\theta) = 1$$

○

Example 1.9 (Normed cone)**.** A set $C \subseteq \mathbb{R}^{n+1}$ of a form

$$C = \{(x,t) \in \mathbb{R}^{n+1} \mid t \geqslant \|x\|\}$$

is called a *normed cone.* If we think in terms of functions and interpret norm as a function, then C represents an epigraph of a norm (epigraphs will be formally defined in the next subsection).

To see that a normed cone is a convex cone consider $(a,u),\ (b,v) \in C$ and $\alpha, \beta \in \mathbb{R}_+$:

$$\alpha(a,u) + \beta(b,v) = (\alpha a + \beta b,\ \alpha u + \beta v)$$

and by assumption and triangle inequality we have

$$\|\alpha a + \beta b\|_2 \leqslant \alpha\|a\|_2 + \beta\|b\|_2 \leqslant \alpha u + \beta v$$

so normed cone is a convex cone. ○

Example 1.10 (Positive semidefinite cone)**.** A set $\mathbb{S}^n_+$ of the following form

$$\mathbb{S}^n_+ = \{\mathbf{A} \in \mathbb{R}^{n \times \in} \mid \mathbf{A} \succcurlyeq \mathbf{0}\} = \{\mathbf{A} \in \mathbb{R}^{n \times \in} \mid x^T\mathbf{A}x \geqslant 0,\ x \in \mathbb{R}^n\},$$

is called a *positive semidefinite cone.*

To show that $\mathbb{S}^n_+$ is a convex cone suppose $\mathbf{A}, \mathbf{B} \in \mathbb{S}^n_+,\ x \in \mathbb{R}^n$ and $\alpha, \beta \geqslant 0$:

$$x^T(\alpha\mathbf{A} + \beta\mathbf{B})x = \alpha x^T\mathbf{A}x + \beta x^T\mathbf{B}x \geqslant 0$$

so $\mathbb{S}^n_+$ is actually a convex cone.

Similarly it can be shown that positive definite cone $\mathbb{S}^n_{++}$:

$$\mathbb{S}^n_{++} = \{\mathbf{A} \in \mathbb{R}^{n \times \in} \mid \mathbf{A} \succ \mathbf{0}\} = \{\mathbf{A} \in \mathbb{R}^{n \times \in} \mid x^T\mathbf{A}x > 0,\ 0 \neq x \in \mathbb{R}^n\},$$

is a convex cone. ○

Now we move on to operations that preserve convexity, i.e. calculus of convex sets. The motivation to learn about convex calculus is that we will need to determine wether a set is convex or not. In most cases showing convexity by the definition is non-trivial, but with convex calculus we are able to establish convexity, because it was constructed from convex sets by operations that preserve convexity.

In fact, this approach is very useful. If we want to write a program that determines wether a set is convex, using convex calculus we will need to create a tree such that its leaves will be basic convex sets, its nodes will be operations that preserve convexity and the root will be the originally given set.

Theorem 1.13 (Intersection of convex sets)**.** If $\emptyset \neq A, B \subseteq \mathbb{R}^n$ are convex sets, then $A \cap B$ is convex.

Proof. Suppose $u, v \in A \cap B$ and $\theta \in [0,1]$.

$$\begin{aligned} u, v \in A \cap B &\iff (u, v \in A) \wedge (u, v \in B) \\ &\iff (\theta u + (1-\theta)v \in A) \wedge (\theta u + (1-\theta)v \in B) \\ &\iff (\theta u + (1-\theta)v) \in A \cap B \end{aligned}$$ □

Theorem 1.14 (Affine mapping of convex sets)**.** Let C be a convex sets and $f : \mathbb{R}^n \to \mathbb{R}^m$ an affine function defined as $x \mapsto \mathbf{A}x + b$ with $\mathbf{A} \in \mathbb{R}^{m \times n},\ b \in \mathbb{R}^m$. The image of C under f,

$$f(C) = \{f(x) \in \mathbb{R}^m \mid x \in C\}$$

is convex.

Likewise,the inverse image of A under f,

$$f^{-1}(C) = \{x \in \mathbb{R}^n \mid f(x) \in C\}$$

is convex.

Proof. Suppose $f(u), f(v) \in f(C)$, $\theta \in [0,1]$ and C is convex,

$$\begin{aligned} f(u), f(v) \in f(C) \iff u, v \in C &\iff \theta u + (1-\theta)v \in C \\ &\iff \mathbf{A}(\theta u + (1-\theta)v) + b \in f(C) \\ &\iff \mathbf{A}(\theta u + (1-\theta)v) + b + \theta b - \theta b \in f(C) \\ &\iff \theta(\mathbf{A}u + b) + (1-\theta)(\mathbf{A}v + b) \in f(C), \end{aligned}$$

so $\theta f(u) + (1-\theta)f(v) \in f(C)$ and $f(C)$ is convex.
Now consider $p, q \in f^{-1}(C)$, $\beta \in [0,1]$ and C is convex,

$$\begin{aligned} p, q \in f^{-1}(C) \iff f(p),\ f(q) \in A &\iff \beta f(p) + (1-\beta)f(q) \in A \\ &\iff \beta(\mathbf{A}p + b) + (1-\beta)(\mathbf{A}q + b) \in A \\ &\iff \mathbf{A}(\beta p + (1-\beta)q) + b \in A \\ &\iff \beta p + (1-\beta)q \in f^{-1}(C) \end{aligned}$$ □

Remark 1.11 (Balls and ellipsoids). As our formulation of balls and ellipsoids suggests, they are just affine transformations of a unit ball, which is obviously convex. ○

Now we focus on generalized inequalities. As name suggests we generalize the idea of a familiar inequality on $\mathbb{R}$ to another context like matrix inequalities, or vector inequalities.

Definition 1.15 (Proper cone)**.** A convex cone K is a *proper cone* if

1. K is closed (contains its boundary)
2. K is solid (has nonempty interior, i.e. is full dimensional with respect to a given vector space)
3. K is pointed (contains no line, i.e. $x \in K, -x \in K \implies x = 0$) ⊚

Definition 1.16 (Generalized inequality)**.** Let K be a proper cone. We define a partial ordering associated with a proper cone K as

$$x \preccurlyeq_K y \iff y - x \in K, \qquad x, y \in K.$$

This ordering is called *generalized inequality* defined by convex cone K. ⊚

Remark 1.12. Strict generalized inequality can be defined as following,

$$x \prec_K y \iff y - x \in \textbf{int } K, \qquad x, y \in K. \qquad \circ$$

Example 1.13 (Nonnegative orthant)**.** An intuitive example of a proper cone is the nonnegative orthant $\mathbb{R}^n_+$, where vectors with some zero components form its boundary, interior is all component-wise positive vectors. So for $x, y \in \mathbb{R}^n_+$ we have component-wise inequality

$$x \preccurlyeq_{\mathbb{R}^n_+} y \iff y - x \in \mathbb{R}^n_+ \iff x_i \leqslant y_i,\ i \in \{1, \dots, n\}.$$

Notice that we used this inequality in the to express a polyhedron. ◯

Example 1.14 (Semidefinite cone)**.** Another example will be a positive semidefinite cone. Interior is formed by positive definite matrices, boundary contains all matrices with some zero eigenvalue, i.e. all singular positive semidefinite matrices and positive semidefinite with all eigenvalues equal to zero forms the origin. Now we can define matrix inequality with respect to $\mathbb{S}^n_+$

$$\mathbf{A} \preccurlyeq_{\mathbb{S}^n_+} \mathbf{B} \iff \mathbf{B} - \mathbf{A} \in \mathbb{S}^n_+ \iff \mathbf{B} - \mathbf{A} \succcurlyeq \mathbf{0} \qquad ◯$$

Notice, that unlike $\leqslant$ a generalized inequality $\preccurlyeq_K$ is a partial ordering as we can have vectors that are incomparable, i.e. $x \not\preccurlyeq_K y$ and $x \not\succcurlyeq_K y$. That's why idea of the smallest element splits into two parts the minimum and the minimal element.

Definition 1.17 (Minimum and minimal)**.** Let $\preccurlyeq_K$ be a generalized inequality with respect to a proper cone K and S as an arbitrary set. Element $x \in S$ such that,

$$x \ \preccurlyeq_K \ y, \qquad \forall y \in S,$$

is called *the minimum element* of S with respect to $\preccurlyeq_K$.

Element $u \in S$ such that,

$$\forall v \in S,\ v \ \preccurlyeq_K \ u \qquad \Longrightarrow \qquad v = u,$$

is called *a minimal element* of S with respect to $\preccurlyeq_K$. ◎

Remark 1.15. Minimal and minimum elements can be described in set notation. The point $x \in S$ is the minimum of S if and only if

$$S \subseteq x + K.$$

Here $x + K$ is a set of all points that are comparable and grater or equal to x. Notice that if x is minimum it follows that $S \subseteq K$.

The point $x \in S$ is minimal of S if and only if

$$S \cap x - K = \{x\}.$$

Here $x - K$ determines a set of all points that are comparable and less or equal to x. Thus, if x is minimal then $S \not\subseteq K$. ∘

The intuitive difference between the minimum and a minimal is the following, a point $x \in S$ is the minimum, when all other points are larger then x with respect to cone K, and if $y \in S$ is a minimal, then there is no points that are less then y.

Now we look at some concepts that will help us to gain a useful characterization of minimum and minimal elements, that will be used in vector optimization problems.

Definition 1.18 (Dual cone)**.** Let $K \subseteq \mathbb{R}^n$ be a cone. The set

$$K^* = \{x \in \mathbb{R}^n \mid u^T x \geqslant 0,\ \forall u \in K\}$$

is called *the dual cone* of K. ◎

Lets look at the following lemma that tells us that the duals of proper cones can define generalized inequalities.

Lemma 1.19. Let K be a cone. Then

1. K^* is a convex cone.
2. If K is closed, i.e. $\mathbf{cl}\ K = K$, then the dual cone of the dual cone is the original cone, i.e. $K^{**} = K$.
3. If K is proper, then the dual cone K^* is a proper cone.

Proof.

1. Using the definition of the dual cone we see that for every $x \in K$ the set $\{y \mid x^T y \geqslant 0\}$ is a half-space, so we express K^* as the following,
$$K^* = \bigcap_{x \in K} \{y \mid x^T y \geqslant 0\}.$$
Since we use non-strict inequality $x^T y \geqslant 0$ it follows that the dual cone is closed.

2. We start with the fact that closure of any cone K can be expressed as an intersection of all homogeneous half-spaces containing K.Thus, for closed cones the statement $\mathbf{cl}\ K = K$ holds. With the previous point we have:
$$\mathbf{cl}\ K = \bigcap_{x \in K^*} \{y \mid x^T y \geqslant 0\} = \{y \mid x^T y \geqslant 0,\ \forall y \in K^*\} = K^{**}.$$

3. Suppose that K is a proper cone. To see that K^* is pointed we use the fact that the original cone K has nonempty interior. Assume that K^* is not pointed, i.e.
$$\begin{aligned} 0 \neq u, -u \in K^* &\Rightarrow x^T u \geqslant 0,\ x^T(-u) \geqslant 0, \qquad \forall x \in K \\ &\Rightarrow x^T u \geqslant 0,\ x^T u \leqslant 0 \\ &\Rightarrow K = \{x \mid x^T u = 0, u \neq 0\}, \end{aligned}$$
so K is affine, hence does not contain nondegenerate ball from the original vector space ($\mathbb{R}^n$) and so has an empty interior.

   Now we show that K^* has nonempty interior assuming K is pointed. Consider the following:
$$\begin{aligned} \mathbf{int}\ K^* = \emptyset &\Rightarrow \exists x \neq 0 : K^* = \{y \mid x^T y = 0,\ (-x)^T y = 0\} \\ &\Rightarrow x, -x \in K^{**} = \{u \mid a^T u \geqslant 0,\ a \in K^*\}, \end{aligned}$$
hence K^{**} contains a line and using point 2 we have that K is not pointed. $\square$

Remark 1.16. In the proof of the previous lemma we used characterization of the interior point of a set from topology, so for $S \subseteq \mathbb{R}^n$ and $x \in S$, the following holds
$$x \in \mathbf{int}\ S \iff \{x + ru \mid \|u\|_2 \leqslant 1,\ u \in \mathbb{R}^n,\ r \neq 0\} \subseteq S, \qquad \circ$$

By now we obtained important properties associated with relation between generalized inequalities and their duals.

Remark 1.17. If K is a convex proper cone and $\preccurlyeq_K$ is a generalized inequality with respect to K, then its dual K^* is also a convex proper cone, hence defines a generalized inequality $\preccurlyeq_{K^*}$. Let $x,\ y \in K$ and $\lambda \succcurlyeq_{K^*} 0$. The important relations between K and K^* are the following:

1. $x \preccurlyeq_K y \iff \lambda^T x \leqslant \lambda^T y$
2. $x \prec_K y \iff \lambda^T x < \lambda^T y,\ \lambda \neq 0.$ ○

Here we present characterization of minimum and minimal using the dual cones, however we omit the proof, since it is based on other non-trivial theorems.

Theorem 1.20 (The minimum element characterization)**.** Let $S \subseteq \mathbb{R}^n$ be any set and $x \in S$. The element x is the minimum element of S with respect to $\preccurlyeq_K$, if and only if for every $\lambda \succ_{K^*} 0$, x is the unique minimizer of $\lambda^T u$ over $u \in S$.

In case of a minimal characterization splits into two different parts.

Theorem 1.21. Let $S \subseteq \mathbb{R}^n$ be any set and $x \in S$. If $\lambda \succ_{K^*} 0$ and x is a unique minimizer of $\lambda^T u$ over all $u \in S$ then x is minimal.

However for the converse implication we need to assume that S is a convex set, as in general $x \in S$ can be a minimal element but not the minimizer of $\lambda^T u, u \in S$ if S is not convex.

Theorem 1.22. Let S be a convex set. If $x \in S$ is minimal and there exists $\lambda \succcurlyeq_{K^*}$ such that x is the unique minimizer of $\lambda^T u$ over all $u \in S$.

The proofs for these theorems can be found in the book [1, p. 46-57].

1.1.3 Convex functions

In this subsection we extend the meaning of convexity to the context of functions. At first, we say what a convex function is and illustrate this definition on some examples. Establishing convexity of a function plays critical role in optimization problem as, roughly speaking, convex problems can be efficiently solved. Then we explain reasons why convexity plays a crucial role in optimization and how with local information about the gradient we obtain global results. Next we move on to operations that preserve convexity and form the basics of the convex calculus in terms of functions.

Definition 1.23 (Convex function)**.** Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a function, such that $\mathbf{dom}\, f$ is a convex set. If for any $x, y \in \mathbf{dom}\, f$ the following holds:

$$f(\theta x + (1-\theta)y) \leqslant \theta f(x) + (1-\theta)f(y), \qquad \theta \in [0,1]$$

then we say that f is a *convex function.* If the strict inequality holds then f is called a *strictly convex function.* If there holds $\geqslant$ instead of $\leqslant$ then f is called *concave function.* ◎

The illustration of a convex function can be viewed in the Figure 1.5.

Actually, we have already seen points of this type $\theta a + (1-\theta)b$, it is nothing else but a point on a line segment between a and b. The definition says that if you evaluate function at a point in a line segment between a and b you get smaller value than convex combination of the end points. In terms of graph we say that line segment connecting $(x, f(x))$ and $(y, f(y))$ lies above the graph of the function.

One extremely useful property of convex functions that directly follows from the definition is that if we restrict the domain of a convex function to a line, then f will be below that line. In some cases it will be worth plotting a function before formal verification of convexity, if the function value is higher then convex weighted sum of the end points, then we are done, and the function is not convex. However, if no such line was founded we have to check convexity by hand.

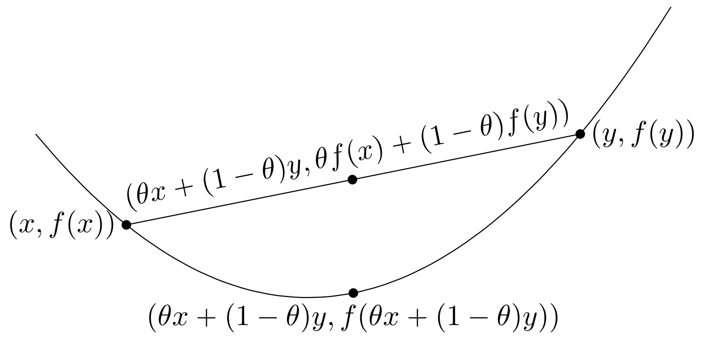

$(\theta x + (1-\theta)y, \theta f(x) + (1-\theta)f(y))$
$(y, f(y))$
$(x, f(x))$
$(\theta x + (1-\theta)y, f(\theta x + (1-\theta)y))$

Figure 1.5: Convex function illustration.

Remark 1.18 (Restriction to a line). The function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if the function $g : \mathbb{R} \to \mathbb{R}$ such that,

$$g(t) = f(x + tv), \qquad \mathbf{dom}\, g = \{t \mid x + tv \in \mathbf{dom}\, f\},\ x \in \mathbf{dom}\, f,\ v \in \mathbb{R}^n$$

is convex in t. ○

Remark 1.19 (Extended-value extension). Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. We define function $\tilde{f} : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ such that,

$$\tilde{f}(x) = \begin{cases} f(x), & x \in \mathbf{dom}\, f, \\ +\infty, & \text{otherwise}, \end{cases}$$

and call it *extended-value extension* of f. ○

This extension will simplify the notation now we can write for any $a, b \in \mathbb{R}^n$ the function is convex if and only if

$$\tilde{f}(\theta a + (1-\theta)b) \leqslant \theta \tilde{f}(a) + (1-\theta)\tilde{f}(b).$$

Notice, that now we are using extended arithmetic and ordering. In the rest of the thesis extended-value notation of functions will be used explicitly instead of the ordinary one.

Now we look at theorems that will help us to establish convexity of a given function.

Theorem 1.24 (First order condition)**.** Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a differentiable function with convex domain. The function f is convex if and only if for $\forall u, v \in \mathbf{dom}\, f$ the following holds

$$f(u) \geqslant f(v) + \nabla f(v)^T (u - v).$$

Proof. The proof can be found in [1, p. 70]. □

Remark 1.20 (First order Tylor approximation). Note, that expression on the right hand side is a *first order Tylor approximation* of a function f at a point v and is, actually, affine in u. ○

The main assertion of the first order condition is, that the function is has higher value everywhere than the first order Tylor approximation. As it is a global result now we can claim that our solution is the optimal and no other can do better.

Theorem 1.25 (Second order condition)**.** Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a twice differentiable function with convex domain. We say that f is convex if and only if the following condition holds

$$\nabla^2 f(u) \succcurlyeq \mathbf{0}, \qquad \forall u \in \mathbf{dom}\, f.$$

Proof. The proof can be found in the literature [1, p. 71]. □

Corollary 1.26 (Strict convexity)**.** If the Hessian matrix is positive definite, i.e. $\nabla^2 f(u) \succ \mathbf{0}$, $\forall u \in \mathbf{dom}\, f$, then we call f *strictly convex.* However, the converse is not generally true as for $f(x) = x^2$ second derivative at $x = 0$ is zero, although it is a strictly convex function.

Corollary 1.27 (Concavity)**.** Using negation of second order condition we can say that

$$f \text{ is concave} \iff \mathbf{dom}\, f \text{ is convex, } \nabla^2 f(x) \preccurlyeq \mathbf{0},\ x \in \mathbf{dom}\, f.$$

We use these conditions to establish convexity of the basic functions, that will be used as cornerstones of calculus of convex functions. Now lets have a look at some examples.

Example 1.21 (Examples on $\mathbb{R}$)**.** Consider the following functions:

1. *Affine*: $f(x) = ax+b$, $\mathbf{dom}\, f = \mathbb{R}$, $a, b \in \mathbb{R}$ is convex and concave, as $a(\theta x+(1-\theta)y)+b = \theta(ax + b) + (1 - \theta)(ay + b)$.

2. *Exponential*: $f(x) = e^{ax}$, $\mathbf{dom}\, f = \mathbb{R}$, $0 \neq a \in \mathbb{R}$ is convex, because second derivative is always positive.

3. *Powers*: $f(x) = x^a$, $\mathbf{dom}\, f = \mathbb{R}_{++}$. If $a \in [0, 1]$ function is concave, and if $a \in \mathbb{R} \setminus (0, 1)$ function is convex. According to the second order condition we have $a(a - 1)x^{a-2} \geqslant 0$, because power of x is always positive and $a(a - 1) \geqslant 0 \iff a \in \mathbb{R} \setminus (0, 1)$.

4. *Powers of absolute values*: $f(x) = |x|^a$, $\mathbf{dom}\, f = \mathbb{R}$, $a \geqslant 1$ is convex, by the previous point.

5. *Logarithms*: $f(x) = \log x$, $\mathbf{dom}\, f = \mathbb{R}_{++}$ is concave, as $f'' < 0$.

6. *Negative entropy*: $f(x) = x \log x$, $\mathbf{dom}\, f = \mathbb{R}_{++}$ is convex, by the second order condition.

○

Now we move on to more interesting and practical examples on $\mathbb{R}^n$.

Example 1.22 (Norms)**.** Let $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ be a norm. To see that any norm is convex, let $\theta \in [0, 1]$ and using homogeneity property and triangle inequality we obtain the following:

$$\|\theta a + (1 - \theta)b\|_2 \leqslant \theta\|a\|_2 + (1 - \theta)\|b\|_2.$$

○

Example 1.23 (Max function)**.** Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a function defined as the following

$$f(x) = \max\{x_1, \dots, x_n\}.$$

We call f a *max function.* To see that max function is convex, let $\theta \in [0, 1]$ and consider the following

$$\max_i\, (\theta a_i + (1 - \theta)b_i) \leqslant \theta \max_i\, a_i + (1 - \theta) \max_i\, b_i.$$

The inequality holds as the sum of maximum elements is at least maximum of sums. Another way to interpret max function is an infinity norm on $\mathbb{R}^n$, i.e. $\|\cdot\|_\infty$ ○

Example 1.24 (Quadratic function)**.** Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a function defined as the following

$$f(x) = (1/2)x^T\mathbf{P}x + q^T x + r, \qquad \mathbf{P} \in \mathbb{S}^n_+.$$

We call f a *quadratic function.* To show that f is convex we look at the Hessian.

$$\nabla f = \mathbf{P}x + q \qquad\qquad \nabla^2 f = \mathbf{P}.$$

So a quadratic function is convex if and only if $\mathbf{P} \in \mathbb{S}^n_+$ ○

Example 1.25 (leastsquares objective)**.** Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a function defined as the following

$$f(x) = \|\mathbf{A}x - b\|_2^2, \qquad \mathbf{A} \in \mathbb{R}^{m\times n}.$$

We call f a *leastsquares objective* function. We use the second order condition to see that f is convex.

$$\nabla f = \mathbf{A}^T(\mathbf{A}x - b) \qquad\qquad \nabla^2 f = 2\mathbf{A}^T\mathbf{A}.$$

Now we show that the Hessian is positive semidefinite. Suppose $x \in \mathbb{R}^n$ and consider the following equations

$$x^T\mathbf{A}^T\mathbf{A}x = (\mathbf{A}x)^T\mathbf{A}x = \|\mathbf{A}x\|_2^2 \geqslant 0 \qquad \bigcirc$$

Example 1.26 (Quadratic-over-linear function)**.** Suppose $f : \mathbb{R}^2 \to \mathbb{R}$ is a function defined as the following

$$f(x) = f(x,y) = x^2/y, \qquad \mathbf{dom}\, f = \mathbb{R} \times \mathbb{R}_{++}.$$

We call f a *quadratic-over-linear function.* To see that f is convex we use the second coincident, so for the gradient we have

$$\nabla f = \left(\frac{2x}{y}, \, -\frac{x^2}{x^2}\right).$$

Now we can work out the Hessian

$$\nabla^2 f = \begin{bmatrix} 2/y & -2x/y^2 \\ -2x/y^2 & 2x^2/y^3 \end{bmatrix} = \frac{2}{y^3}\begin{bmatrix} y^2 & -xy \\ -xy & x^2 \end{bmatrix} = \frac{2}{y^3}\begin{bmatrix} y \\ -x \end{bmatrix}\begin{bmatrix} y \\ -x \end{bmatrix}^T \geqslant \mathbf{0}.$$

To see that outer product of any vector $x \in \mathbb{R}^n$ with itself is positive semidefinite lets take an arbitrary $y \in \mathbb{R}^n$. Now consider the following

$$y^Txx^Ty = (x^Ty)^Tx^Ty = \|x^Ty\|_2^2 \geqslant 0. \qquad \bigcirc$$

function

Example 1.27 (Log-sum-exp function)**.** Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a function such that

$$f(x) = \log(1^Te^x), \qquad e^x = (e^{x_1}, \ldots, e^{x_n}),\ 1 = (1_1, \ldots, 1_n).$$

The function f is called *log-sum-exp function.*

We show that log-sum-exp function is convex using the second order condition. For the gradient we have

$$\nabla f = \frac{1}{1^Tz}z, \qquad z = e^x.$$

Before computing the Hessian lets take partial derivatives

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = -\frac{z_iz_j}{(1^Tz)^2} \qquad\qquad \frac{\partial^2 f}{\partial^2 x_i} = \frac{z_i(1^Tz) - z_i^2}{(1^Tz)^2}.$$

Now we can work out the Hessian.

$$\nabla^2 f = \frac{1}{(1^Tz)^2}\begin{bmatrix} z_1(1^Tz) - z_1^2 & \cdots & z_1z_j & \cdots & z_1z_n \\ \vdots & \ddots & & & \vdots \\ z_jz_1 & \cdots & z_j(1^Tz) - z_j^2 & \cdots & z_jz_n \\ \vdots & & & \ddots & \vdots \\ z_nz_1 & \cdots & z_nz_j & \cdots & z_n(1^Tz) - z_n^2 \end{bmatrix}.$$

So in the matrix notation we have

$$\nabla^2 f = \frac{1}{(1^T z)^2}((1^T z)\,\mathbf{diag}(z) - zz^T).$$

Now we verify that $\nabla^2 f \succ \mathbf{0}$ i.e.

$$v^T \nabla^2 f v = \frac{1}{(1^T z)^2}\left(\Big(\sum_i z_i\Big)\Big(\sum_i v_i z_i\Big) - \Big(\sum_i v_i z_i\Big)^2\right) \geqslant 0$$

the inequality holds by Cauchy-Schwarz inequality $(p^T p)(q^T q) \geqslant (p^T q)^2$ when we take $p_i = v_i\sqrt{z_i}$ and $q_i = \sqrt{z_i}$, hence log-sum-exp function is convex. ◯

Now we show the connection between convex sets and convex functions that will be useful in transforming optimization problems in equivalent forms.

Definition 1.28 (Epigraph)**.** Let f be a function $f : \mathbb{R}^n \to \mathbb{R}$. The set denoted as the following

$$\mathbf{epi}\, f = \{(x,t) \in \mathbb{R}^{n+1} \mid x \in \mathbf{dom}\, f, t \geqslant f(x)\},$$

is called *epigraph* of f. ◎

Remark 1.28 (Epigraphs of convex functions). The connection between convex functions and convex sets is the epigraph. The function f is convex if and only if $\mathbf{epi}\, f$ is convex. The claim is, actually, obvious as the border of f, that is equal to graph of f, is convex and $\mathbf{epi}\, f$ has solid interior, hence contains every line segment between its two arbitrary points. ○

Remark 1.29 (Hypograph). Similarly, if $f : \mathbb{R}^n \to \mathbb{R}$ is a concave function, then we define its *hypograph* as the following,

$$\mathbf{hypo}\, f = \{(x,t) \in \mathbb{R}^{n+1} \mid x \in \mathbf{dom}\, f, t \leqslant f(x)\}.$$

We see that f is concave if and only if $\mathbf{hypo}\, f$ is convex. ○

So far we have discovered some basic techniques for establishing convexity of a function, like the Hessian, the gradient and lines. However, these methods are quite complicated and they may be the last resort to verify convexity, as in most cases we can calculus of convex functions. We have already seen a bunch of basic convex functions so lets look at methods to combine these atoms that will give us an easier way to examine convexity.

We start with some elementary operations moving on to complicated one.

Lemma 1.29 (Nonnegative weighted sum)**.** If h and g are convex functions and $\alpha \in \mathbb{R}_{++}$, then $\alpha f_1 + f_2$ is convex. This idea can be generalized to n convex functions,

$$g = \sum_i w_i f_i, \qquad w_i \geqslant 0.$$

Proof. Follows from the definition of convex function. □

Lemma 1.30 (Composition with affine function)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. The function $g : \mathbb{R}^m \to \mathbb{R}$ defined as the following,

$$g(x) = f(\mathbf{A}x + b), \qquad \mathbf{A} \in \mathbb{R}^{n\times m}, b \in \mathbb{R}^n$$

is convex.

Proof. Suppose $\theta \in [0,1]$,

$$f(\mathbf{A}(\theta x + (1-\theta)y) + b) = f(\theta(\mathbf{A}x + b) + (1-\theta)(\mathbf{A}y + b)) \leqslant \theta f(\mathbf{A}x + b) + (1-\theta) f(\mathbf{A}y + b)$$ □

Remark 1.30 (Norm of affine function). By the previous statement we have that for any norm precomposed with affine mapping, i.e. $\|\mathbf{A}x + b\|_2$ is convex. ◦

Theorem 1.31 (Point-wise supremum)**.** Let f be a function such that $f(x, y)$ is convex in x for all $y \in A$, where A is a completely arbitrary set. Then function g defined as the supremum over A, i.e.

$$g(x) = \sup_{y \in A} f(x, y)$$

is a convex function of x with domain

$$\mathbf{dom}\, g = \{x \mid (x, y) \in \mathbf{dom}\, f, \forall y \in A,\ \sup_{y \in A} f(x, y) < +\infty\}.$$

Proof. To see that g is a convex function we use the epigraph characterization of convex functions and say that

$$\mathbf{epi}\, g = \bigcap_{y \in A} \mathbf{epi}\, f(\cdot, y).$$

So the epigraph of g is convex, hence g is convex. □

Remark 1.31 (Point-wise infimum of concave functions). Similarly, we say that point-wise infimum over concave functions is concave. ◦

Example 1.32 (leastsquares cost as a function of weights)**.** Suppose we have a weighted least-squares problem, so we minimize the objective

$$\sum_i w_i (a_i^T x - b_i)^2, \qquad a_i \in \mathbb{R}^m.$$

Here w_i denote weights and can be negative, so function can be unbounded below. We define the *optimal list-squares cost function* g as the following,

$$g(w) = \inf_x \sum_i w_i (a_i^T x - b_i)^2,$$

where domain is defined as

$$\mathbf{dom}\, g = \{x \mid g(x) > -\infty\}.$$

In the matrix form we obtain

$$g(w) = \inf_x (\mathbf{A}x - b)\, \mathbf{diag}(w) (\mathbf{A}x - b).$$

By definition g if infimum over linear functions in w, hence is concave. ○

Example 1.33 (Maximum eigenvalue of a symmetric matrix)**.** Let $f : \mathbb{S}^n \to \mathbb{R}$ be a function such that,

$$f(\mathbf{X}) = \lambda_{\max}(\mathbf{X}) = \sup_{\|y\|_2 = 1} y^T \mathbf{X} y.$$

According to the definition of f as a supremum over linear functions in $\mathbf{X}$ it follows that f is convex. ○

Notice that in the Theorem 1.31 there is no requirement of any kind how the variable you maximizing over enters the function, the only requirement is that $f(x, y)$ is convex for all $y \in A$. In the case of minimizing a convex function we find a noticeable asymmetry in the assumptions, so we have a much stronger condition.

Theorem 1.32 (Minimization)**.** If g is jointly convex in (x, y) and $C \neq \emptyset$ is a convex set, then the function f defined as

$$f(x) = \inf_{y \in C} g(x, y), \qquad \mathbf{dom}\, f = \{x \mid (x, y) \in \mathbf{dom}\, g, y \in C,\ f(x) > -\infty\}$$

is a convex function.

Proof. To show this we use the characterization of a convex function via convexity of its epigraph,

$$\mathbf{epi}\, f = \{(x, t) \mid (x, y, t) \in \mathbf{epi}\, f, y \in C\}.$$

Epigraph of f is a convex set as it is a projection of a convex set $\mathbf{epi}\, f$ onto y. □

Let's consider more general ideas like function composition. The thing that needs to be said is that most previously defined operations that preserve convexity, can be formulated as special cases of the composition rule, however, they are worth mentioning as they are easy to understand and remember.

Before looking at the formal definition of the composition rule consider the following straightforward example.

Assume that $g : \mathbb{R} \to \mathbb{R}$ and $h : \mathbb{R} \to \mathbb{R}$, both functions are twice differentiable and $\mathbf{dom}\, h = \mathbf{dom}\, g = \mathbb{R}$. Now we define $f : \mathbb{R} \to \mathbb{R}$ such that $f(x) = h(g(x))$. By the second order condition convexity of f reduces to $f''(x) \geqslant 0$, $\forall x \in \mathbb{R}$, roughly speaking only the sign of the second derivative of f matters. Next step will be applying the chain rule twice, so we get

$$f''(x) = h''(g(x))g'(x)^2 + h'(g(x))g''(x).$$

Now suppose that g is concave, i.e. $g'' \leqslant 0$, in this case we need to have $h' \leqslant 0$, i.e. h should be non-increasing, so the second term is positive. The first term is positive only if h is convex, i.e. $h'' \geqslant 0$. As the result we have made up our first composition rule

$$f \text{ is convex if } g \text{ is concave, } h \text{ is convex and non-increasing.}$$

Theorem 1.33 (The composition rule)**.** Let $g : \mathbb{R}^n \to \mathbb{R}$ and $h : \mathbb{R} \to \mathbb{R}$ be any functions. We define a function $f : \mathbb{R}^n \to \mathbb{R}$ as

$$f(x) = h(g(x)), \qquad \mathbf{dom}\, f = \{x \in \mathbf{dom}\, g \mid g(x) \in \mathbf{dom}\, h\}.$$

The function f is convex if

- g are convex, h is convex and $\tilde{h}$ is non-decreasing,
- g are concave, h is convex and $\tilde{h}$ is non-increasing.

Here $\tilde{h}$ is an extended-value extension of h.

Proof. The proof can be found in the book [1, p. 85,86]. □

1.2 Convex Optimization

In this section we look at optimization problems in general and related concepts like feasibility, constrains set. Then we study the problem of convex optimization and common families of related problems.

Definition 1.34 (Optimization problem in standard form). An *optimization problem in standard form* is given by

$$\begin{aligned} \text{minimize } & f_0(x) \\ \text{subject to } & f_i(x) \leqslant 0, \qquad i = 1, \dots, m, \\ & h_j(x) = 0, \qquad j = 1, \dots, p. \end{aligned} \tag{1.2}$$

where

- $x \in \mathbb{R}^n$ is the optimization variable
- $f_0 : \mathbb{R}^n \to \mathbb{R}$ is the objective function
- $f_i : \mathbb{R}^n \to \mathbb{R}$ are the inequality constrains
- $h_j : \mathbb{R}^n \to \mathbb{R}$ are the equality constrains ⊚

In the definition we use word “minimize” which is not the same as mathematical operator min. In our context “minimize” and “subject to” can be viewed as attributes of an optimization problem.

Remark 1.34 (Terminology). Assume that we have an optimization problem in standard form 1.2.

- The set of all points where the objective function and constrains are defined, i.e.

  $$\mathcal{D} = \bigcap_{i=0}^{m} \mathbf{dom}\, f_i \cap \bigcap_{i=1}^{p} \mathbf{dom}\, h_i,$$

  is called the *domain* of an optimization problem (1.2).

- The point $x \in \mathbb{R}^n$ that lies in the domain $\mathcal{D}$ of an optimization problem and satisfies the constrains it is called a *feasible point.* The set of all feasible points x is called the *feasible set.*

- If an optimization problem has no constrains, we call it an *uncostrained* problem.

- If an optimization problem has an empty feasible set it is call *infeasible*, otherwise it is *feasible.*

- If there exists a sequence of feasible points x_ℓ such that $f_0(x_\ell) \to -\infty$, as $\ell \to \infty$ then the optimization problem is called *unbounded below.*

- The *optimal value* $p^\star$ of the optimization problem is defined as

$$p^\star = \begin{cases} \inf_x \{ f_0(x) \mid f_i(x) \leqslant 0,\ h_j(x) = 0,\, i = 1, \dots, m,\, j = 1, \dots, p \}, & \text{problem is feasible} \\ +\infty, & \text{problem is infeasible} \\ -\infty, & \text{problem is unbounded} \end{cases}$$

- If $x^\star$ is feasible and solves the optimization problem, i.e. $f(x^\star) = p^\star$, we call $x^\star$ an *optimal point.* The set of all optimal points, denoted as X_{opt}, is called the *optimal set.*
- A feasible point x is called *locally optimal*, if there exists $R > 0$ such that
$$f_0(x) = \inf_x \{f_0(y) \mid y \text{ is feasible}, \ \|y - x\|_2 \leqslant R\}.$$
- The constrain is called *redundant*, if it does not change the feasible set. ○

Now we look at some straightforward examples to illustrate defined terminology.

Example 1.35 (Examples)**.** Assume we have an unconstrained problem such that $f_0 : \mathbb{R} \to \mathbb{R}$, $\mathbf{dom}\, f_0 = \mathbb{R}_{++}$.

- $f_0(x) = -\log x$, $p^\star = -\infty$ and the problem is unbounded below.
- $f_0(x) = x \log x$, $p^\star = -1/e$, $x^\star = 1/e$.

If we add a constraint $x < 0$, where the objective function is given by $f_0(x) = \log x$ then this problem is infeasible. ◯

Now we look at one variation on optimization problems that is called feasibility problem. Actually, before rushing into solving an optimization problem, we need to check wether the feasible set is not empty and if it is not, we start searching the solution.

Definition 1.35 (Feasibility problem)**.** The optimization problem of the following form
$$\begin{array}{rll} \text{find} & x & \\ \text{subject to} & f_i(x) \leqslant 0, & i = 1, \dots, m \\ & h_j(x) = 0, & j = 1, \dots, p \end{array}$$
is called a *feasibility problem.* ◎

Notice that feasibility problem corresponds to optimization a constant function subject to some constraints. Finally we get to the definition of a convex optimization problem.

Definition 1.36 (Convex optimization problem)**.** The optimization problem defined as
$$\begin{array}{rll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leqslant 0, & i = 1, \dots, m, \\ & a_i^T x = b_i, & i = 1, \dots, p \end{array} \tag{1.3}$$
where $f_0, \dots, f_m$ are convex functions, is called a *convex optimization problem.* ◎

Moreover, another equivalent definition of convex optimization can be given, but we stay with 1.3. We now look at the stunning fact that makes convex optimization so important.

Theorem 1.37 (Local and global optima)**.** Suppose we have an optimization problem (1.3). If x is locally optimal, then x is (global) optimal.

Proof. By the definition of the local optimum we have
$$\|z - x\|_2 \leqslant r \implies f_0(x) \leqslant f_0(z), \qquad z \text{ is feasible}, \ r > 0.$$
Now we suppose that x is not global optimum, so there exists y such that $f_0(y) \leqslant f_0(x)$ and $\|y - x\|_2 > r$. We take z and θ as the following
$$z = \theta y + (1 - \theta)x, \qquad \theta = \frac{r}{2\|y - x\|_2}.$$

Then we have $\|z - x\|_2 = r/2 < r$ and z is feasible since feasible set is convex. As f_0 is a convex function we get

$$f_0(z) \leqslant \theta f_0(y) + (1-\theta) f_0(x) = f_0(x) - \theta(f_0(x) - f_0(y)) < f_0(x),$$

which contradicts with local optimality of x. □

Roughly speaking, what this theorem says is this, if we take a point x and look around in its arbitrary close neighborhood and it is better then all the others that means that its better then all the other points that are outside the neighborhood.

Theorem 1.38 (Optimality for differentiable objective)**.** Suppose we have a convex optimization problem with differentiable objective. A feasible point $x \in \mathbf{dom}\, f_0$ is optimal if and only if the following inequality holds.

$$\nabla f_0(x)^T (y - x) \geqslant 0, \qquad \forall y \text{ feasible}.$$

Proof. The proof can be found in the literature [1, p. 139, 140]. □

Corollary 1.39 (Unconstrained problem)**.** Suppose we have an unconstrained convex optimization problem with differentiable objective. A feasible point $x \in \mathbf{dom}\, f_0$ is optimal if and only if the following holds.

$$\nabla f_0(x) = 0$$

Proof. The proof can be found in the book [1, p. 140]. □

We now show basic transformations that preserve convexity of an optimization problem.

Equivalent transformations

- Suppose we have a convex optimization problem (1.3) with equality constraints, given by $\mathbf{A}x = b,\ \mathbf{A} \in \mathbb{R}^{p \times n},\ x \in \mathbb{R}^n,\ b \in \mathbb{R}^p$. We can eliminate these constraints by passing expressing the variable x as $\mathbf{F}z + \bar{x}$, where column vectors of $\mathbf{F} \in \mathbb{R}^{n \times\ (n-p)}$ span the null space of $\mathbf{A}$ and $\bar{x}$ is a particular solution of $\mathbf{A}x = b$. New optimization problem is given by

$$\begin{aligned} \underset{z}{\text{minimize}}\ & f_0(\mathbf{F}z + \bar{x}) \\ \text{subject to}\ & f_i(\mathbf{F}z + \bar{x}) \leqslant 0, \qquad i = 1, \dots, m. \end{aligned}$$

  Since we only precomposed convex functions with an affine one convexity remains.

- For the problem (1.3) we can formulate an equivalent epigraph problem as the following

$$\begin{aligned} \underset{(x,t)}{\text{minimize}}\ & t \\ \text{subject to}\ & f_0(x) \leqslant t, \\ & f_i(x) \leqslant 0, \qquad i = 1, \dots, m, \\ & \mathbf{A}x = b. \end{aligned}$$

1.2.1 Linear programming

Linear programming or LP is in some sense the simplest optimization problem, where every function involved is affine. Moreover, it is totally not a new topic as even Fourier had a paper about this [3]. However modern methods for solving LP trace back to 1950 to Stanford where the SIMPLEX method was invented by George Dantzig.

Definition 1.40 (Linear program)**.** The optimization problem of the following form is called a *linear program* or just LP

$$\begin{aligned} &\text{minimize } c^T x \\ &\text{subject to } \mathbf{G}x \leqslant h, \qquad \mathbf{G} \in \mathbb{R}^{m\times n} \\ &\qquad\qquad \mathbf{A}x = b, \qquad \mathbf{A} \in \mathbb{R}^{p\times n}. \end{aligned}$$ ◎

In linear programs the feasible set is a polyhedron. Although, linear program seems to be a poor model as every function are affine it can be applied on problems that does not seem linear at all. To illustrate this we look at an interesting example called Chebychev center of a polyhedron, avoiding "Hello World" historical one called the diet problem.

Example 1.36 (Chebychev center of a polyhedron)**.** Suppose we have a polyhedron $P = \{x \in \mathbb{R}^n \mid a_i^T x = b,\, i = 1, \dots, m\}$. The task is to find the largest ball that lies in the polyhedron P, here the center of the ball is called the *Chebyshev center.* We represent an Euclidean ball as $B(x_c, r) = \{x_c + u \mid \|u\|_2 \leqslant r\}$. Thus, the variables are x_c, r, and we want to find maximum r such that $B(x_c, r) \subseteq P$. At first glance this problem is way not linear, so it will illustrate that LP can be applied in very unobvious cases.

The simplest possible case is that the polyhedron is just a half-space defined by $(a, b) \in \mathbb{R}^{n+1}$, hence we want to have

$$\|u\|_2 < r \Rightarrow a^T(x_c + u) \leqslant b.$$

By the Cauchy-Schwarz inequality we obtain

$$\sup\{a^T u \mid \|u\|_2 \leqslant r\} = r\|u\|_2.$$

Now for every $a_i, b_i,\ i = 1, \dots, m$ that form the polyhedron we obtain

$$a_i^T x_c + r\|a_i\|_2 \leqslant b_i.$$

Notice that here $\|a_i\|_2$ is a constant, hence we can formulate a corresponding linear program as

$$\begin{aligned} &\underset{r,x_x}{\text{maximize}}\ r \\ &\text{subject to } a_i^T x_c + r\|a_i\|_2 \leqslant b_i, \qquad i = 1, \dots, m. \end{aligned}$$ ○

1.2.2 Quadratic programming

Generally quadratic programming came up in middle fifties of the 20th century, it was first generalization of LP, where objective is not linear but a convex quadratic function, although feasible set reminds a polyhedron.

Definition 1.41 (Quadratic program)**.** The optimization problem of the following form is called a *quadratic program* or just QP

$$\begin{aligned} &\text{minimize } (1/2)x^T\mathbf{P}x + q^T x + r \\ &\text{subject to } \mathbf{G}x \geqslant h \\ &\qquad\qquad \mathbf{A}x = b. \end{aligned}$$

Here matrices are defined as

$$\mathbf{P} \in \mathbb{S}^n_+,\ \mathbf{G} \in \mathbb{R}^{m\times n},\ \mathbf{A} \in \mathbb{R}^{p\times n}$$ ⊚

Notice, if we take $\mathbf{P} = \mathbf{0}$ we get an LP. Also, if $\mathbf{P}$ is not a positive semidefinite the problem goes from being extremely straightforward to NP-hard.

One thing that we immediately see, that for LP in most cases we get the solution on a vertex (if there is more than one solution, for example all the points on the edge minimize the objective, we still can pick an end point of this edge, which is a vertex). In case of QP the solution does not have to be in vertex, as level curves of the objective are now ellipsoids.

Now let have a look at some examples, now we are not ignoring "Hello World" example called leastsquares.

Example 1.37 (leastsquares)**.** The leastsquares problem corresponds to the following unconstrained QP problem

$$\text{minimize}\ \ \|\mathbf{A}x - b\|_2^2.$$

Here $\mathbf{A} \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ ○

Next generalization is to introduce convex quadratic inequalities instead of linear.

Definition 1.42 (Quadratically constrained quadratic program)**.** The optimization problem of the following form is called a *quadratically constrained quadratic program* or QCQP

$$\begin{aligned} \text{minimize}\ & (1/2)x^T\mathbf{P}_0x + q_0^Tx + r_0 \\ \text{subject to}\ & (1/2)x^T\mathbf{P}_ix + q_i^Tx + r_i \leqslant 0, \qquad i = 1,\dots,m \\ & \mathbf{A}x = b. \end{aligned}$$

Here matrices used in quadratic functions are positive semidefinite, i.e.
$\mathbf{P}_i \in \mathbb{S}^n_+,\ i = 0,\dots,m$ and $\mathbf{A} \in \mathbb{R}^{p\times n}$. ⊚

1.2.3 Vector optimization

Now we introduce the extension of the objective to be vector valued function.

Definition 1.43 (Vector optimization problem)**.** The problem of the following form is called *vector optimization problem*

$$\begin{aligned} \underset{\text{w.r.t. } K}{\text{minimize}}\ & f_0(x) \\ \text{subject to}\ & f_i(x) \leqslant 0, \qquad i = 1,\dots,m \\ & \mathbf{A}x = b. \end{aligned} \tag{1.4}$$

Here

- $f_0 : \mathbb{R}^n \to \mathbb{R}^q$ is K convex.
- $K \subseteq \mathbb{R}^q$ is a proper cones.
- $\mathbf{A} \in \mathbb{R}^{p\times n}$.

⊚

Notice that if we have a scalar valued objective and two points that are feasible and have the same objective value we can not prefer one point to another, if we do then our model is not valid. Now the objective returns a vector, so we do not have total ordering and we have to decide.

To actually compare objective values we introduce the following concepts.

Remark 1.38 (Terminology). Suppose $\mathcal{O}$ is a set of all feasible objective value, i.e.

$$\mathcal{O} = \{f_0(x) \mid x \text{ is feasible}\}.$$

- Point x is *optimal* if $f(x)$ is the minimum value of $\mathcal{O}$ (1.17).
- Point x is *Pareto optimal* if $f(x)$ is a minimal value of $\mathcal{O}$. ○

Definition 1.44 (Multicretarion optimization). Consider the problem of the form (1.4). If $K = \mathbb{R}^q_+$ we call such problem *multicretarion optimization* problem as the components of the objective can be interpreted as the following

$$f_0(x) = (F_1(x), \ldots, F_q(x)).$$

We can say that $F_i(x)$ are separate scalar objectives, and we want each of the to be minimized.

- The point $x^\star$ is optimal if

  $$y \text{ is feasible} \implies f_0(x^\star) \preccurlyeq_K f_0(y).$$

  If the optimal points exist, then the objectives do not compete.
- The point x^{op} is Pareto optimal if

  $$y \text{ is feasible} \wedge f_0(y) \preccurlyeq_K f_0(x^{\text{op}}) \implies f_0(x^{\text{op}}) = f_0(y)$$

  In these case we say that there exists a trade-off between the objective functions. ◎

Now we can define the semantics of solving a multicretarion problem as finding a Pareto optimal point, or even exploring all the Pareto optimal points. We can interpret it like having a joystick that we are allowed to move around on the Pareto optimal surface.

Definition 1.45 (Scalarization). Assume a problem of the form (1.4). To solve such a problem means to choose $\lambda \succ_{K^*} 0$ and solve a new problem called *scalar* optimization problem of the following form

$$\begin{aligned} \text{minimize } & \lambda^T f_0(x) & \\ \text{subject to } & f_i(x) \leqslant 0, & i = 1, \ldots, m \\ & \mathbf{A}x = b, & \mathbf{A} \in \mathbb{R}^{p \times n}. \end{aligned}$$

◎

Remark 1.39 (Pareto optimal). If point x is the optimal for a scalar problem, then x is a Pareto-optimal point for the original vector problem. ○

1.3 Duality

The next topic is duality. Duality can be viewed as an organized way of forming highly non-trivial bounds on optimization problems, even on hard ones that are not convex. Furthermore, when the problem is convex the bound is usually tight. Another interpretation corresponds to passing the constraints directly to the objective, roughly speaking, making an unconstrained problem.

Remark 1.40 (Terminology). In this section we call the original optimization problem the *primal problem.* ○

We start with the the definition of the Lagrangian.

Definition 1.46 (Lagrangian)**.** The *Lagrangian* $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ of an optimization problem in standard form (1.2) is given by

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x), \qquad \mathbf{dom}\, L = \mathcal{D} \times \mathbb{R}^m \times \mathbb{R}^p.$$

Vectors λ, ν are called *dual variables.* ⊚

Roughly speaking, we take the objective and add a linear combination of the inequality and equality constraints.

Definition 1.47 (Lagrange dual function)**.** The *Lagrange dual function* is defined as

$$g(\lambda, \nu) = \inf_{x \in \mathcal{D}} L(x, \lambda, \nu).$$ ⊚

Remark 1.41 (Convexity). Notice that the Lagrange dual function is concave in λ, ν regardless the convexity of the original problem, since it is an infimum over affine functions (affine functions by the definition are both convex and concave). ○

This brings us to what we call the Lagrange dual problem. Intuition is this, for any optimization problem we want to construct the best possible lower bound on the optimal value established by the Lagrange dual function, in other words we are interested in the maximum lower bound.

Definition 1.48 (Lagrange dual problem)**.** Suppose we have a primal optimization problem in the standard form (1.2). The *Lagrange dual problem* is given by

$$\begin{aligned} &\underset{\lambda, \nu}{\text{maximize}}\ g(\lambda, \nu) \\ &\text{subject to}\ \ \lambda \succcurlyeq 0. \end{aligned}$$

The *optimal value* of the Lagrange dual problem is denoted by $g^\star$ and $(\lambda^\star, \nu^\star)$ is the *solution of the dual problem.* ⊚

Remark 1.42 (Dual). In the remaining text instead of writing Lagrange dual problem we write dual problem or just dual. By now we have primal and dual optimization problems. ○

Remark 1.43 (Unboundness below). Since Lagrange dual function is infimum over linear function, it is unbounded below. As we can see, when the hyperplane has any slope, for every given point we can find a point with smaller vallue of the Lagrangian. Furthermore, such lower bound is useless. The whole thing changes when hyperplane has no slope, in this case the minimum takes a real value. One way to handle unboundness is to define the domain of the Lagrange dual *implicitly* as the following

$$\mathbf{dom}\, g = \{(\lambda, \nu) \in \mathbb{R}^m \times \mathbb{R}^p \mid \inf_{x \in \mathcal{D}} L(x, \lambda, \nu) > \infty\}.$$

However, in case of an optimization problems, we rather make the domain constrains *explicit* and write directly to the subject to attribute of the problem. ○

Remark 1.44 (Dual feasible points). Notice that the only case when g actually defines a lower bound is when $\lambda \succcurlyeq 0$, as by the definition of the standard form (1.2) inequality constrains are non-positive. So any vector $(\lambda, \nu) \in \mathbf{dom}\, g$ with non-negative λ is called *dual feasible point.* ○

Lemma 1.49 (Lower bound property)**.** Suppose $(\lambda, \nu) \in \mathbf{dom}\, g$ and $\lambda \succcurlyeq 0$. Then the following is always true

$$g(\lambda, \nu) \leqslant p^\star.$$

Here $p^\star$ is the optimal value of the given optimization problem.

Proof. Suppose z is primal feasible. Now since $f_i(x) \leqslant 0,\ i = 1,\dots,p$ and $h_i(x) = 0,\ i = 1,\dots,m$ for every feasible x and $\lambda \succcurlyeq 0$ we obtain

$$\begin{aligned} g(\lambda,\nu) &= \inf_{x\in\mathcal{D}} \left(f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \right) \\ &\leqslant f_0(z) + \sum_{i=1}^{m} \lambda_i f_i(z) + \sum_{i=1}^{p} \nu_i h_i(z) \\ &\leqslant f_0(z). \end{aligned}$$

□

Combining the dual problem with lower bound property we get an idea of weak duality.

Definition 1.50 (Weak duality)**.** Suppose we have primal and dual problems with optimal values $p^\star$ and $g^\star$ respectively. Then by (1.49)

$$g^\star \leqslant p^\star.$$

This property is called *weak duality* and it always holds. The difference between the dual optimal value and the primal optimal value

$$p^\star - g^\star$$

is called the *optimal duality gap.* ◎

One may ask, if weak duality is actually useful. The answer is yes. As we saw the dual problem is convex, hence efficiently solvable, so we use weak duality to find a lower bound on a very difficult problem and then talk about suboptimal points. Moreover, the lower bound can be used to define some non-trivial heuristics for solving the original hard problem.

Now we look at strong duality, when the dual optimal value is equals to the primal optimal value.

Definition 1.51 (Strong duality)**.** Suppose we have primal and dual problems with optimal values $p^\star$ and $g^\star$ respectively and the optimal duality gap is zero, i.e.

$$g^\star = p^\star,$$

then we say that *strong duality* holds. ◎

Generally, strong duality does not hold, however there exists non-convex problems with zero duality gap. More interesting fact is this, usually for convex problems strong duality is attained.

Remark 1.45 (Constraint qualifications). Conditions under which strong duality holds are called *constraint qualifications.* ○

Remark 1.46 (Relative interior). Notice that interior of an affine set C is an empty set, so we define a *relative interior* of an affine set as the following

$$\mathbf{relint}\, C = \{x \in C \mid B(x,r) \cap \mathbf{aff}\, C,\ \mathbf{aff}\, C \subseteq C, r > 0\}.$$

Here $\mathbf{aff}\, C$ is the set of all affine combinations (1.10) of C and $B(x,r)$ is a ball with center x and radius r (1.8). ○

Theorem 1.52 (Slater's condition)**.** If we have a convex optimization problem (1.3) and there exists a strictly feasible point z, i.e.

$$\exists z \in \mathbf{relint}\, \mathcal{D} :\ f_i(z) < 0,\ \mathbf{A}z = b, \qquad i = 1,\dots,m,$$

then strong duality holds. We call this condition Slater's condition.

Remark 1.47 (Variation on Slater's condition)*.* If we have a convex optimization problem with k affine constrains, then we have a relaxed version of Slater's condition

$$f_j(z) \leqslant 0,\ f_i(z) < 0,\ \mathbf{A}z = b, \qquad j = 1, \dots, k,\ i = k+1, \dots, m. \qquad \circ$$

Proof. The proof of this condition can be found in the book [1, Section 5.3.2]. □

Now we observe the situation with zero duality gap.

Remark 1.48 (Complementary slackness)*.* Let $x^\star$ and $(\lambda^\star, \nu^\star)$ be solutions for primal and dual problems respectively. If strong duality holds, then we obtain the following

$$\begin{aligned} f_0(x^\star) = g(\lambda^\star, \nu^\star) &= \inf_{x \in \mathcal{D}} \left(f_0(x) + \sum_{i=1}^m \lambda_i^\star f_i(x) + \sum_{i=1}^p \nu_i^\star h_i(x) \right) \\ &\leqslant f_0(x^\star) + \sum_{i=1}^m \lambda_i^\star f_i(x^\star) + \sum_{i=1}^p \nu_i^\star h_i(x^\star) \\ &\leqslant f_0(x^\star), \end{aligned}$$

hence equality holds. This observation leads to important consequences. Firstly, we get this

$$\sum_{i=1}^m \lambda_i^\star f_i(x^\star) = 0. \tag{1.5}$$

The statement above is called *complementary slackness* and implies the following

$$\lambda_i^\star > 0 \implies f_i(x^\star) = 0 \qquad\qquad f_i(x^\star) < 0 \implies \lambda_i^\star = 0.$$

The second point is that for fixed $(\lambda^\star, \nu^\star)$ point $x^\star$ minimizes the Lagrangian $L(x, \lambda^\star, \nu^\star)$, hence the gradient vanishes

$$\nabla f_0(x^\star) + \sum_{i=1}^m \lambda_i^\star \nabla f_i(x^\star) + \sum_{i=1}^p \nu_i^\star \nabla h_i(x^\star) = 0. \qquad \circ$$

This brings us to necessary optimality condition for any optimization problem.

Corollary 1.53 (Karush–Kuhn–Tucker conditions)**.** Suppose we have a primal problem with differentiable objective and constraint functions. If strong duality holds and $x^\star$ and $(\lambda^\star, \nu^\star)$ are primal and dual solutions then the following conditions must be satisfied

- Primal constrains: $f_i(x^\star) \leqslant 0,\ h_j(x^\star) = 0;\ i = 1, \dots, m,\ j = 1, \dots, p.$
- Dual constrains: $\lambda^\star \geqslant 0$.
- Complementary slackness: $\sum_{i=1}^m \lambda_i^\star f_i(x^\star) = 0$.
- Gradient of the Lagrangian with respect to $x^\star$ is zero

$$\nabla f_0(x^\star) + \sum_{i=1}^m \lambda_i^\star \nabla f_i(x^\star) + \sum_{i=1}^p \nu_i^\star \nabla h_i(x^\star) = 0.$$

These conditions are called *Karush-Kuhn-Tucker conditions* or the KKT conditions.

Moreover, for convex problems these conditions are also sufficient, so we have an equivalence.

Corollary 1.54 (KKT conditions for convex problems)**.** Suppose primal problem is convex and the strong duality holds. Let $\tilde{x}$ and $\tilde{\lambda}, \tilde{\nu}$ be points that satisfy the KKT conditions, then they are primal and dual optimal points respectively.

Proof. By complementary slackness we have $f_0(\tilde{x}) = L(\tilde{x}, \tilde{\lambda}, \tilde{\nu})$. Since the primal is convex and the gradient vanishes we get $g(\tilde{\lambda}, \tilde{\nu}) = L(\tilde{x}, \tilde{\lambda}, \tilde{\nu})$, hence we reach the lower bound, i.e. $f_0(\tilde{x}) = g(\tilde{\lambda}, \tilde{\nu})$ and $\tilde{x}, \tilde{\lambda}, \tilde{\nu}$ are optimal. □

Notice that since the Slater's condition implies the strong duality we get the following characterization of the primal solution.

Corollary 1.55 (KKT conditions and Slater's condition)**.** Suppose we have a convex optimization problem with differentiable objective and constraint functions that satisfies Slater's condition. Then $\tilde{x}$ is optimal if and only if there exist $\tilde{\lambda}, \tilde{\nu}$ such that $\tilde{x}, \tilde{\lambda}, \tilde{\nu}$ satisfy the KKT conditions.

1.4 Non-convex optimization

Let us make some notes on non-convex optimization, i.e. when we can not transform the problem to the convex form 1.3. In general, it can be shown that solving a non-convex problem is NP-hard [4]. The intuition is this, in convex setting we knew that local optimum is a global one, but in non-convex case there exists more than one local optimal points, often exponentially many local minima, so to prove optimality we have to check all of them. Thus, in general, there is no an effective method suitable for every task, and typically every problem uses its own approach. Some basic scenarios of dealing with non-convexity are the following

- Formulate a convex relaxation of the original problem and solve the relaxation. i.e. relaxing some of the constraints of the original problem and extend the objective to the larger space. Thus, all feasible points of the original non-convex problem are still reachable, but the optimal value is now a lower or upper bound on the original optimal value [5, Chapter 6].
- Use a domain specific heuristics, derived from the problem structure. We would see one such approach in the later chapter.
- Run convex optimization algorithm [1] more than once with different initial points and hope for sensible results.

1.5 Reproducing Kernel Hilbert Spaces

In this section we introduce useful concepts from functional analysis related to reproducing kernel Hilbert spaces or RKHS, like inner product spaces and functionals. This section may seem to be outlying from the general stream of the text, however, it has immediate applications in machine learning as it can enormously strengthen linear models to cope with non-linear data.

We start with general Hilbert spaces without reproducing property and related concepts.

Definition 1.56 (Inner product)**.** Assume $\mathcal{F}$ is a vector space over $\mathbb{R}$. A function $\langle \cdot, \cdot \rangle_{\mathcal{F}} : \mathcal{F} \times \mathcal{F} \to \mathbb{R}$ is called an *inner product* on $\mathcal{F}$ if for every $f, g, h \in \mathcal{F}$ and all $\alpha, \beta \in \mathbb{R}$ the following holds:

- $\langle \alpha f + \beta g, h \rangle_{\mathcal{F}} = \alpha \langle f, h \rangle_{\mathcal{F}} + \beta \langle g, h \rangle_{\mathcal{F}}$

[1]Often algorithm is slightly adapted to a given problem.

- $\langle f, g\rangle_{\mathcal{F}} = \langle g, f\rangle_{\mathcal{F}}$
- $(\langle f, f\rangle_{\mathcal{F}} \geqslant 0) \wedge (\langle f, f\rangle_{\mathcal{F}} = 0 \iff f = 0)$ ◎

Remark 1.49 (Norm). Within a specific inner product that maps every two elements to the real numbers we can also define norm as the following

$$\|f\|_{\mathcal{F}} = \sqrt{\langle f, f\rangle_{\mathcal{F}}}.$$

It can be shown that using Cauchy-Schwarz inequality and properties of the inner product this norm satisfies the norm Definition 1.4. ○

Remark 1.50 (Dot product). If our vector space is $\mathbb{R}^n$ the inner product takes the following form

$$\langle u, v\rangle_{\mathbb{R}^n} = u^T v.$$ ○

Now the only one ingredient remains to define a Hilbert space. It is so-called completeness, the reason we need this is, we want limits of every Cauchy sequence of space elements be in the vector space.

Remark 1.51 (Metric). Using the definition of a norm (1.4) with respect to a vector space $\mathcal{F}$, we can define the distance between the vectors $g, h \in \mathcal{F}$ as $d(g, h) = \|g - h\|_{\mathcal{F}}$. This concept enables us to examine convergence of vector sequences. ○

Definition 1.57 (Convergent sequence)**.** Suppose $\mathcal{F}$ is a normed vector space and $\{f_n\}_{n=1}^{\infty}$ is a sequence of vectors in $\mathcal{F}$. We say that $\{f_n\}_{n=1}^{\infty}$ *converges* to $f \in \mathcal{F}$ if

$$\forall \epsilon > 0,\ \exists n_0 = n_0(\epsilon) \in \mathbb{N} : \forall n \geqslant n_0 \qquad \|f_n - f\|_{\mathcal{F}} < \epsilon.$$ ◎

Remark 1.52 (Notation). Here we use notation with parentheses to emphasise the dependency of n_0 on ϵ . ○

Definition 1.58 (Cauchy sequence)**.** Suppose $\mathcal{F}$ is a normed vector space and $\{f_n\}_{n=1}^{\infty}$ is a sequence of vectors in $\mathcal{F}$. We say that $\{f_n\}_{n=1}^{\infty}$ is a *Cauchy sequence* if

$$\forall \epsilon > 0,\ \exists n_0(\epsilon) \in \mathbb{N} : \forall n, m \geqslant n_0 \qquad \|f_n - f_m\|_{\mathcal{F}} < \epsilon.$$ ◎

Remark 1.53 (Convergence and Cauchy sequences). Note that by triangle inequality every convergent sequence is Cauchy. To see this, we take $\epsilon/2$ and find n_0 that for all $n, m > n_0$ we have $\|f_n - f\|_{\mathcal{F}} < \epsilon/2$ and $\|f_m - f\|_{\mathcal{F}} < \epsilon/2$. Finally, we obtain

$$\|f_n - f_m\|_{\mathcal{F}} \leqslant \|f_n - f\|_{\mathcal{F}} + \|f_n - f\|_{\mathcal{F}} < \epsilon.$$

However the converse does not holds. Take a normed space $(\mathbb{Q}, |\cdot|)$ and the sequence 3.1, 3.14, 3.141, 3.1415, ..., which is not convergent since $\pi \notin \mathbb{Q}$. ○

Now we are ready to generalize concepts from finite dimensional space to infinite ones namely to Hilbert spaces. A common example, that will be widely used in this section, is a vector space of functions.

Definition 1.59 (Hilbert space)**.** Let $\mathbb{H}$ be a vector space over $\mathbb{R}$ with associated inner product, such that every Cauchy sequence converges. We call $\mathbb{H}$ a *Hilbert space.* ◎

Let us look at a bunch of concepts from functional analysis tightly associated with reproducing kernel Hilbert spaces.

Definition 1.60 (Linear operator)**.** Let $\mathcal{F}$ and $\mathcal{G}$ be normed vector spaces over $\mathbb{R}$. We call a function $A : \mathcal{F} \to \mathcal{G}$ a *linear operator* if it meets homogeneity and additivity, i.e.

$$A(\alpha f + g) = \alpha A f + A g, \qquad \forall f, g \in \mathcal{F},\ \alpha \in \mathbb{R}.$$ ◎

Definition 1.61 (Linear functional)**.** Suppose $\mathcal{F}$ is a vector spaces over $\mathbb{R}$. A linear operator $A : \mathcal{F} \to \mathbb{R}$ is called a *linear functional* on $\mathcal{F}$. ⊚

Example 1.54. Suppose $\mathcal{F}$ is a vector spaces over $\mathbb{R}$ with an inner product. We can define a linear functional A_g mapping form $\mathcal{F}$ to $\mathbb{R}$ for a specific $g \in F$ as $A_g(f) = \langle f, g \rangle_{\mathcal{F}}$ ○

Definition 1.62 (Continuity)**.** Suppose $\mathcal{F}$ and $\mathcal{G}$ are normed vector spaces over $\mathbb{R}$. A function $A : \mathcal{F} \to \mathcal{G}$ is called *continuous at a point* $h \in \mathcal{F}$ if

$$\forall \epsilon > 0,\ \exists \delta = \delta(f_0, \epsilon) > 0 : \qquad \|f - h\|_{\mathcal{F}} < \delta \implies \|Af - Ah\|_{\mathcal{G}} < \epsilon.$$

We say that A is *continous* on $\mathcal{F}$ if, it is continuous at every $h \in \mathcal{F}$. ⊚

The definition above tells us that a continuous function always maps a convergent sequence to another convergent sequence in its domain.

Definition 1.63 (Lipschitz continuity)**.** Suppose $\mathcal{F}$ and $\mathcal{G}$ are normed spaces over $\mathbb{R}$. A function $A : \mathcal{F} \to \mathcal{G}$ is called *Lipschitz continuous* if

$$\exists C > 0 : \forall f, h \in \mathcal{F} \qquad \|Af - Ah\|_{\mathcal{G}} \leqslant C\|f - h\|_{\mathcal{F}}.$$ ⊚

Notice that Lipschitz continuity is a stronger condition than ordinary continuity, as we can have $\delta = \epsilon/C$ so δ depends only on ϵ.

Remark 1.55 (Continuity of linear functionals)*.* A linear functional (1.54) defined as a function of $f \in \mathcal{F}$ is continuous by Cauchy-Schwarz inequality

$$\|A_g(f) - A_g(h)\|_{\mathbb{R}} = |A_g(f) - A_g(h)| = |\langle f - h, g \rangle_{\mathcal{F}}| \leqslant \|g\|_{\mathcal{F}} \, \|f - h\|_{\mathcal{F}}.$$ ○

Definition 1.64 (Bounded operator)**.** Suppose $\mathcal{F}$ and $\mathcal{G}$ are normed spaces over $\mathbb{R}$ and $A : \mathcal{F} \to \mathcal{G}$ is a linear operator. The *norm of the operator* A is defined as

$$\|A\| = \sup_{0 \neq f \in \mathcal{F}} \frac{\|Af\|_{\mathcal{G}}}{\|f\|_{\mathcal{F}}}.$$

We say that A is *bounded operator*, if $\|A\| < \infty$. ⊚

The following lemma shows the relation between operator boundness and continuity.

Lemma 1.65 (Boundness and continuity)**.** Assume $\mathcal{F}$ and $\mathcal{G}$ are normed spaces over $\mathbb{R}$ and $A : \mathcal{F} \to \mathcal{G}$ is a linear operator. A is bounded if and only if A is continuous at some point of $\mathcal{F}$.

Proof. The proof for the lemma above can be found in the literature [2, p. 6]. □

As we have seen linear functionals on a normed space $\mathcal{F}$ can be defined via an inner product by fixing one argument. The next theorem states that, every linear functional can be represented as an inner product with some vector from $\mathcal{F}$.

Theorem 1.66 (Riesz representation)**.** Assume $\mathcal{F}$ is a Hilbert spaces over $\mathbb{R}$. All continuous linear functionals take the form of $\langle \cdot, g \rangle_{\mathcal{F}}$ for some $g \in \mathcal{F}$.

Proof. The proof of this theorem can be found in the literature[6, Theorem 4.12]. □

Now we are ready to define a reproducing kernel Hilbert space of functions from some set $\mathcal{X}$ to $\mathbb{R}$.

Definition 1.67 (Evaluational functional)**.** Suppose $\mathbb{H}$ is a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X} \neq \emptyset$ and $x \in \mathcal{X}$. A mapping $\delta_x : \mathbb{H} \to \mathbb{R}$, such that $f \mapsto f(x)$ is called the *evaluational functional.* ⊚

Remark 1.56 (Linearity of evaluational functional). To see that evaluational functionals are indeed linear operators we recall the definition of a sum of functions

$$(\alpha f + g)(x) = \alpha f(x) + g(x), \qquad f, g \in \mathbb{H},\ \alpha \in \mathbb{R}.$$

And now for $\delta_x : \mathbb{H} \to \mathbb{R}$ we have

$$\delta_x(\alpha f + g) = (\alpha f + g)(x) = \alpha \delta_x f + \delta_x g. \qquad \circ$$

The desired feature of evaluational functional is boundness or continuity. Roughly speaking, if we have, in some sense, similar functions f, g and evaluate these functions at a some point x, images of x under f, g will also be similar. This intuition leads to the concept of reproducing kernel Hilbert spaces.

Definition 1.68 (Reproducing kernel Hilbert spaces)**.** Assume $\mathbb{H}$ is a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is not empty. Hilbert space $\mathbb{H}$ is called the *reproducing kernel Hilbert spaces* or RKHS if for all $x \in \mathcal{X}$ evaluational functional δ_x is continuous. ⊚

This definition brings us to a pretty property of RKHS like point-wise convergence.

Lemma 1.69 (Point-wise convergence)**.** Assume $\mathbb{H}$ is a RKHS of functions $f : \mathcal{X} \to \mathbb{R}$ and $\mathcal{X} \neq \emptyset$. If a sequence of functions $\{f_n\}_{n=1}^{\infty}$ converges to some $f \in \mathbb{H}$ in the norm $\|\cdot\|_{\mathbb{H}}$, then this sequence converges at every point x from the domain, i.e.

$$\lim_{n\to\infty} \|f_n - f\|_{\mathbb{H}} = 0 \implies \lim_{n\to\infty} f_n(x) = f(x), \qquad \forall x \in \mathcal{X}.$$

Proof. Since for all $x \in \mathcal{X}$ we have the following

$$|f_n(x) - f(x)| = |\delta_x f_n - \delta_x f| \leqslant \|\delta_x\| \, \|f_n - f\|_{\mathbb{H}}.$$

The inequality holds, as by the definition of RKHS evaluational functional δ_x is continuous, i.e. bounded (1.65). □

Remark 1.57 (Point-wise convergence in Hilbert spaces). However, if a given Hilbert space is not RKHS, convergence does not imply point-wise convergence. In other words, although sequence of functions converges to some other function f, there exists at least one point u such that, the sequence of function values at u does not get close to the image of u under the f. ◦

Even though we are talking about reproducing kernel Hilbert spaces, by now we have no idea of reproducing kernels and what do they actually reproduce.

Definition 1.70 (Reproducing kernel)**.** Let $\mathbb{H}$ be a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$ and $\mathcal{X} \neq \emptyset$. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that, for all $x \in X$ and all $f \in \mathbb{H}$ the following holds

- $k(\cdot, x) \in \mathbb{H}$,
- $f(x) = \langle f, k(\cdot, x)\rangle_{\mathbb{H}}$ (*the reproducing property*),

is called a *reproducing kernel* of $\mathbb{H}$. Particularly, for all $x, y \in X$ we have

$$k(x, y) = \langle k(\cdot, x), k(\cdot, y)\rangle_{\mathbb{H}} \qquad ⊚$$

As we can see a reproducing kernel is tightly related to an inner product on some Hilbert space of functions. However, it also yields questions like how reproducing kernels are connected with RKHS, or why we need reproducing property?

Lemma 1.71 (Uniqueness)**.** Let $\mathbb{H}$ be a Hilbert space of function $f : \mathcal{X} \to \mathbb{R}$. If it has a reproducing kernel k then k is unique.

Proof. Assume k, h are kernels of $\mathbb{H}$. Now for all $f \in \mathbb{H}$ and all $x \in X$ we have

$$\langle f, h(\cdot, x) - k(\cdot, x)\rangle_{\mathbb{H}} = f(x) - f(x) = 0.$$

Taking $f = h(\cdot, x) - k(\cdot, x)$ we get

$$\|h(\cdot, x) - k(\cdot, x)\|_{\mathbb{H}}^2 = 0, \qquad \forall x \in \mathcal{X}.$$

Hence $h = k$. □

The next theorem shows the relation between reproducing kernels and RKHS.

Theorem 1.72 (Existence of the reproducing kernel in RKHS)**.** Let $\mathbb{H}$ be a Hilbert space of function $f : \mathcal{X} \to \mathbb{R}$. Hilbert space $\mathbb{H}$ is RKHS if and only if $\mathbb{H}$ has a reproducing kernel.

Proof. $\Longleftarrow$ Suppose we have a Hilbert space $\mathbb{H}$ with reproducing kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, then by Cauchy-Schwarz inequality and reproducing property we obtain

$$\begin{aligned}|\delta_x f| = |f(x)| &= |\langle f, k(\cdot, x)\rangle_{\mathbb{H}}| \\ &\leqslant \|f\|_{\mathbb{H}}\|k(\cdot, x)\|_{\mathbb{H}} \\ &= \|f\|_{\mathbb{H}}\sqrt{\langle k(\cdot, x), k(\cdot, x)\rangle_{\mathbb{H}}} \\ &= \|f\|_{\mathbb{H}}\sqrt{k(x, x)}.\end{aligned}$$

So $\delta_x f$ is bounded, hence continuous, therefore $\mathbb{H}$ is RKHS.

$\Longrightarrow$ Now suppose we have a RKHS $\mathbb{H}$ with a continuous evaluational functional δ_x. By Riesz theorem (1.66) there exists $f_{\delta_x} \in \mathbb{H}$ and for all $f \in \mathbb{H}$ the following is true

$$\delta_x f = \langle f, f_{\delta_x}\rangle_{\mathbb{H}}.$$

Now for all $x \in \mathcal{X}$ we introduce a function $k(\cdot, x) : \mathcal{X} \to \mathbb{R}$ defined as

$$k(x', x) = f_{\delta_x}(x'), \qquad \forall x' \in \mathcal{X},$$

so $k(\cdot, x) = f_{\delta_x} \in \mathbb{H}$ and now

$$\langle f, k(\cdot, x)\rangle_{\mathbb{H}} = \delta_x f = f(x), \qquad \forall f \in \mathbb{H}. \tag{1.6}$$

Hence k meets the conditions of reproducing kernel (1.70). □

Remark 1.58 (Representer and inner products in RKHS). The theorem above brings us to very important properties of RKHS:

- By (1.6) we can call $k(\cdot, x)$ a *representer of a evaluation* at point x.
- Evaluation of f in RKHS $\mathbb{H}$ can be viewed as taking inner product of f with the representer of x. ○

Now we look at an important property of reproducing kernels called positive definiteness.

Definition 1.73 (Positive definite function)**.** Suppose $h : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a function. If h is symmetric add for every $(a_1, \ldots, a_n) \in \mathbb{R}^n$ and for all $(x_1, \ldots, x_n) \in \mathbb{R}^n$ the following is true

$$\sum_{j,i=1}^{n} a_i a_j h(x_i, x_j) \geqslant 0,$$

we call h a *positive definite function*[2]. ◎

[2]If we represent the sum in matrix form we get $a^T\mathbf{H}a \geqslant 0$, in this case matrix $\mathbf{H}$ is positive *semi*definite and not positive definite. However, we would call function h positive definite to stay consistent with machine learning literature.

Lemma 1.74 (Positive definiteness and inner products)**.** Suppose $\mathbb{H}$ is a Hilbert space and $\phi : \mathcal{X} \to \mathbb{H}$ is a mapping from non empty set $\mathcal{X}$ to the Hilbert space $\mathbb{H}$. Then for all $x, y \in \mathcal{X}$ inner product $\langle \phi(x), \phi(y) \rangle_{\mathbb{H}}$ is a positive definite function.

Proof. For all $a \in \mathbb{R}^n$ we obtain

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j \langle \phi(x_i), \phi(x_j) \rangle_{\mathbb{H}} &= \sum_{i=1}^{n}\sum_{j=1}^{n} \langle a_i\phi(x_i), a_j\phi(x_j) \rangle_{\mathbb{H}} \\ &= \left\langle \sum_{i=1}^{n} a_i\phi(x_i), \sum_{j=1}^{n} a_j\phi(x_j) \right\rangle_{\mathbb{H}} \\ &= \left\langle \sum_{i=1}^{n} a_i\phi(x_i), \sum_{i=1}^{n} a_i\phi(x_i) \right\rangle_{\mathbb{H}} \geqslant 0. \end{aligned}$$ □

Corollary 1.75 (Positive definiteness and reproducing kernels)**.** If we take $k(\cdot, x) = \phi(x)$ we see that reproducing kernels are positive definite functions.

Remark 1.59 (Converse implication). Actually, it can be shown that converse also holds and all positive definite functions are related to inner products as positive definite functions meet Cauchy-Schwarz inequality. ○

We now define a kernel, as a function that is an inner product in some Hilbert space.

Definition 1.76 (Kernel)**.** Suppose $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a function, with $\mathcal{X} \neq \emptyset$. We call k a *kernel* if there exists a Hilbert space $\mathbb{H}$ over $\mathbb{R}$ and a function $\phi : \mathcal{X} \to \mathbb{H}$ such that for all $x, y \in \mathcal{X}$ the following holds

$$k(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathbb{H}}.$$ ◎

Remark 1.60 (Kernels, reproducing kernels and positive definiteness). Notice in that we dropped word reproducing and stuck with kernel focusing only on the inner product in some Hilbert space which is not necessary RKHS. However, by the definition (1.70) every reproducing kernel is also a kernel. Since we derived the definition of kernel from the inner product by the definition kernels are positive definite. ○

Now the only question is, why we have to know about RKHS and reproducing kernels. The next theorem will give us the answer, more precisely, how to construct RKHS from a given kernel (not a reproducing kernel).

Remark 1.61 (Moore-Aronszajn). Let k be a kernel, such that $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $\mathcal{X} \neq \emptyset$. Then there exists a unique Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$ for which k is a reproducing kernel. ○

Proof. Complete prof can be found in the literature[2, p. 11 - 18]. □

Corollary 1.77 (Reproducing kernels and PD functions)**.** Every positive definite function is a reproducing kernel.

By the given above corollary reproducing kernels, positive definite functions and kernels are the same concepts!

This leads to operations with kernels, that gives us a free hand in constructing our own kernels.

Lemma 1.78 (Operations on kernels)**.** Suppose k, h are kernels on $\mathcal{X}$ and $\mathcal{Y}$, respectively, and $\alpha \in \mathbb{R}_+$. Then for all $a, b \in \mathcal{X}$ and $u, v \in \mathcal{Y}$ the following functions

1. $\alpha k(a, b)$,

2. $k(a,b) + h(u,v)$,
3. $k(a,b)h(u,v)$

are kernels.

Proof. First and second points follow from positive definiteness of kernels.

For the last point we use the fact that Gram matrices $\mathbf{K}, \mathbf{H} \in \mathbb{S}^n_+$ associated with inner products defined by kernels k, h are positive semidefinite. Applying Cholesky decomposition on $\mathbf{K}$ we obtain $\mathbf{X}\mathbf{X}^T$ such that $\mathbf{K} = \mathbf{X}\mathbf{X}^T$. Now for any $c \in \mathbb{R}^n$ consider the following

$$\sum_{i,j=1}^{n} c_i c_j \left(\sum_{k=1}^{n} \mathbf{X}_{i,k} \mathbf{X}_{j,k} \right) \mathbf{H}_{i,j} = \sum_{k=1}^{n} \sum_{i,j=1}^{n} c_i c_j \mathbf{X}_{i,k} \mathbf{X}_{j,k} \mathbf{H}_{i,j} = \sum_{k=1}^{n} z_k^T \mathbf{H} z_k \geqslant 0.$$

With $z_k = (c_1 \mathbf{X}_{1,k}, \ldots, c_n \mathbf{X}_{n,k})$. □

By the lemma above we can construct some useful kernels.

Example 1.62 (Kernels)**.** Suppose $\mathcal{X} = \mathbb{H} = \mathbb{R}^n$. The simplest case would be a *linear kernel* that is just an inner product

$$k_{\text{lin}}(u,v) = \langle u, v \rangle, \qquad \forall u, v \in \mathbb{R}^n.$$

For polynomials with non-negative coefficients $p(x) = \sum_{i=0}^{n} a_i x^i$ we can define the following *polynomial kernel*

$$k_{\text{poly}}(u,v) = (\langle u, v \rangle + c)^d, \qquad \forall u, v \in \mathbb{R}^n,\ c \in \mathbb{R}_+.$$

We can also use infinite sums with non-negative coefficients, for example Tylor series. This brings us to *exponential kernel*

$$k_{\text{exp}}(u,v) = \exp\left(\frac{\langle u, v \rangle}{\sigma^2} \right), \qquad \forall u, v \in \mathbb{R}^n,\ \sigma > 0.$$

Now we define a feature map $\phi : \mathbb{R}^n \to \mathbb{R}$ as

$$\phi(x) \exp\left(-\frac{\|x\|_2^2}{2\sigma^2} \right), \qquad \forall u, v \in \mathbb{R}^n,\ \sigma > 0,$$

so here the feature space is just $\mathbb{R}$. From feature map we work out a kernel $h : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, as the following

$$h(u,v) = \langle \phi(u), \phi(v) \rangle_{\mathbb{R}} = \phi(u)\phi(v) = \exp\left(-\frac{\|u\|_2^2 + \|v\|_2^2}{2\sigma^2} \right).$$

Next for all $u, v \in \mathbb{R}^n$ we introduce a new *Gaussian kernel*

$$\begin{aligned} k_{\text{Gauss}}(u,v) &= h(u,v) k_{exp}(u,v) \\ &= \exp\left(-\frac{\|u\|_2^2 + \|v\|_2^2 - 2\langle u, v \rangle}{2\sigma^2} \right) \\ &= \exp\left(-\frac{\|u - v\|_2^2}{2\sigma^2} \right). \qquad (1.7) \end{aligned}$$

○

Chapter 2

Knowledge Engineering

This chapter outlines basic tasks in the field of knowledge engineering. Here knowledge engineering is viewed in broader sense of extracting non-trivial insights about the given domain. We mainly use popular data based approach instead of expert driven one and focus on problems that can be represented in our mathematical optimization framework. As the result, we formulate various tasks as optimization problems, so solving them provides us with hidden patterns in the data. This mathematical approach yields a great advantage over computer systems, since we can measure quality of knowledge obtained. In other words, we want our mathematical models, or better machines, to learn some knowledge from raw data. Thus, in this chapter we use widely developed framework of machine learning and data mining.

Two main scenarios will be considered: supervised and unsupervised learning. We now present terminology related to supervised case and after this make make some notes to unsupervised one.

Next terminology outlines summarize [7, Chapter 1, 2].

Supervised learning setting In the supervised learning scenario problems usually take the following form, we have encoded some information and now based on this information we want to make some decisions. This intuition describes a *prediction task*.

- We receive a *training set* $S = ((x_1, y_1), \ldots, (x_m, y_m))$, that is used to find the best model describing the data. For every element (x_i, y_i) we call x_i an *input* and y_i a *label* or *output*. Inputs and outputs come from some input space and output space $\mathcal{X}$ and $\mathcal{Y}$, respectively. We assume that there exists some unknown distribution $\mathcal{Z}$ over $\mathcal{X}$ and x_i are drawn i.i.d. from $\mathcal{Z}$, we denote this by $x_i \sim \mathcal{Z}$.

- Suppose there exists some *target mapping* $f : \mathcal{X} \to \mathcal{Y}$ that maps inputs to the outputs. However, the exact representation of f is not known. We have only some examples of evaluation f on the training data, i.e. $y_i = f(x_i)$, where $(x_i, y_i) \in S$.

- We now construct a space of mappings $h : \mathcal{X} \to \mathcal{Y}$. Usually we do not include every function that maps $\mathcal{X}$ to $\mathcal{Y}$, i.e. h are chosen according to our assumptions about f. The space of all h is denoted by $\mathcal{H} = \{h : \mathcal{X} \to \mathcal{Y}\}$ and is called *hypothesis space*. Each $h \in \mathcal{H}$ is called a *hypothesis*.

- Depending on the problem we introduce a *loss function* $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ that measures the difference between predicted labels $h(x_i)$ and true labels y_i.

- The *learning rule* is to pick a $h \in \mathcal{H}$ that minimizes the mean difference between $h(x_i)$ for $x_i \sim \mathcal{Z}$ and corresponding y_i. We write this problem as the following

$$\underset{h \in \mathcal{H}}{\text{minimize}} \ \mathbb{E}_{x_i \sim \mathcal{Z}}[L(h(x_i), y_i)].$$

  Here the objective is called the *generalization error.* Since the the distribution $\mathcal{Z}$ is not known we rather minimize *empirical error* over the training set given by

$$\underset{h \in \mathcal{H}}{\text{minimize}} \ \frac{1}{|S|} \sum_{(x_i, y_i) \in S} L(h(x_i), y_i).$$

Unsupervised learning setting In unsupervised learning we do not have any labels so the training set looks like $S \subset \mathcal{X}$. Thus, we can hardly speak about loss functions, hence we do not minimize the generalized error. Since there is no labels our goal is to find some structure in the data. The space of such structures can be viewed as hypothesis set in the supervised scenario. Additionally, the problem domain usually suggests some criterion of the structure quality. The learning rule is to find a pattern in the data that optimizes the given criterion.

Hyperparameters Often when we introduce the learning rule we have some free parameters. These are called *hyperparametrs.* To give an example, consider clustering problem, in k-means optimization problem the hyperparameter k corresponds to the number of clusters. Usually to find suitable hyperparameters we use *cross-validation.* We create ℓ validation sets $V_j, j = 1, \dots, \ell$ from the training and then ℓ times train the model on $S \setminus V_j$ and measure the error ε_j on V_j. Then we calculate cross-validation error $\varepsilon = (1/\ell) \sum_j \varepsilon_j$ and select the hyperparameters with the lowest error.

Kernels Before we dive in, let us present usage of kernel functions in machine learning based on the Corollary 1.77 and the Remark 1.60.

Remark 2.1 (Kernel related terminology).

- A function that maps original data to some RKHS, i.e. $\phi : \mathcal{X} \to \mathbb{H}$ is called *feature mapping.*
- A symmetric positive definite function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a *kernel function.* Kernel function is determined by the feature map ϕ. and represents evaluation of inner product on mapped vectors, i.e. for $x, y \in \mathcal{X}$

$$k(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathbb{H}}. \qquad \circ$$

- An image of a data point $x \in \mathcal{X}$ under a feature mapping, i.e. $\phi(x) \in \mathbb{H}$ is called a *feture vector.*
- A RKHS, where the original data points are mapped is called *feature space* and denoted by $\mathbb{H}$. Quite often $\mathbb{H}$ represents a space of functions and has infinite dimension, consider the Gaussian kernel 1.7.

Since every kernel has reproducing property, i.e. for all $f \in \mathbb{H}$, and all $x \in \mathcal{X}$ we obtain

$$f(x) = \langle k(x, \cdot), f \rangle_{\mathbb{H}},$$

and also for every $x \in \mathcal{X}$

$$k(x, \cdot) \in \mathbb{H}.$$

In other words, assuming that $\mathbb{H}$ is a function space $\phi(x)$ is a function such that $\phi(x)(\cdot) : \mathbb{H} \to \mathbb{R}$! We can also write this as

$$\phi(x)(\cdot) = k(x, \cdot).$$

Such notation is called *canonical feature map* representation. Summing up reproducing property and canonical representation, now for every $x, y \in \mathcal{X}$ we have a canonical representations in a Hilbert space, and by reproducing property we can always evaluate an inner product of these two representations. This fact allows us to fix some $x \in \mathcal{X}$ and iterate over other data points $z_i \in \mathcal{X}$ and compute inner products

$$\langle k(x, \cdot), k(z_i, \cdot)\rangle_{\mathbb{H}}.$$

Remark 2.2 (Kernel matrix). It is common practice to represent kernel functions via Gram matrix $\mathbf{K}$ of inner products in the feature space. So for every x_i, x_j from the training set we have

$$\mathbf{K}_{i,j} = k(x_i, x_j) = \langle \phi(x_i), \phi(x_j)\rangle_{\mathbb{H}}. \qquad \circ$$

Now we move on to specific problems. Basically, every subsection that follows can be divided in two these parts

- problem formulation, for example linear classification or regression,
- determining some solving strategy and working out a hypothesis space,
- working out a learning rule, for example SVM for classification or least squares for regression.

2.1 Classification

In this section we look at special type of supervised learning problems called classification, namely binary classification. The setting is the following

- Training set is denoted by $S \in (\mathcal{X} \times \mathcal{Y})^m$ such that $y_i = f(x_i)$, where $\mathcal{X} \subset \mathbb{R}^n$ is an input space and $\mathcal{Y} = \{\pm 1\}$ is an output space.
- The learning problem is to find hypothesis from the hypothesis space $h \in \mathcal{H}$ with the smallest number of misclassifications of data points $x \in \mathcal{X}$.

We introduce most popular learning rule called support vector machines.

2.1.1 Hard SVM

Here we present a support vector machine learning rule that handles an intuitive approach of margin maximization. Furthermore, in conjunction with kernel methods this learning approach is comparable to neural network one. This section summarizes [7, Chapters 5,6] and [8].

Definition 2.1 (Liner separability). Let $S \in (\mathcal{X} \times \mathcal{Y})^m$ be a training set. If there exists a hyperplane

$$H_0 = \{x \in \mathbb{R}^n \mid w^T x = b\}, \qquad w \in \mathbb{R}^n, b \in \mathbb{R}$$

such that for all $(x_i, y_i) \in S$

$$\begin{aligned} w^T x_i + b > 0, &\quad y_i = 1 \\ w^T x_i + b < 0, &\quad y_i = -1, \end{aligned}$$

then S is called *linearly separable* and the hyperplane H_0 is called a *separating hyperplane*. ⊚

Remark 2.3 (Hypothesis space). Assuming linear separability we formulate the hypothesis space as the following

$$\mathcal{H} = \{h(x) = \operatorname{sign}(w^T x + b) \mid y(w^T x + b) > 0, w \in \mathbb{R}^n, b \in \mathbb{R}, (x, y) \in S\}. \qquad \circ$$

Notice that in case of linear separability we have infinitely many such hyperplanes, hence hypothesis space $\mathcal{H}$ has infinitely many elements. That leeds us to the question, how to select the best one. Intuitively, we want such hyperplane to be the "safest" one, i.e. if we wiggle a bit with our data points we want our hyperplane be still a valid classifier.

Remark 2.4 (Marginal hyperplanes). Now we replace H_0 with two hyperplanes H_+ and H_- parallel to H_0

$$H_\pm = \{x \in \mathbb{R} \mid w^T x + b = \pm\delta\}, \ w \in \mathbb{R}^n, \ b, \delta \in \mathbb{R}.$$

For simplicity reasons we define $\delta = 1$. Hyperplanes $H_\pm$ are called *marginal*, since there is a margin between them. Now the data points with positive label are in the half-space defined by H_+ and negative ones in H_-, i.e. for H_+ we have

$$w^T x_i + b \geqslant +1, \quad y_i = 1, \tag{2.1}$$

and for H_-

$$w^T x_i + b \leqslant -1, \quad y_i = -1. \tag{2.2}$$

○

Remark 2.5 (Liner separability modification). Instead of two inequalities it is handy to have the only one. Multiplying both (2.1) and (2.2) by corresponding y_i we get

$$y_i(w^T x_i + b) \geqslant 1, \quad (x_i, y_i) \in S \qquad \circ$$

Since we have a notion of marginal hyperplanes let us also precisely derive a margin between them.

Remark 2.6 (Margin). Assume $x_+ \in H_+$, so $w^T x_+ + b = 1$. We want to find a perpendicular distance ρ from x_+ to H_-. Since H_+ and H_- are parallel, they share same (w, b), so moving x_+ ρ times unit step in the negative direction given by w we make x_+ lie on H_-. Now consider the following

$$\begin{aligned} w^T(x_+ - \rho\frac{w}{\|w\|_2}) + b &= -1 \\ w^T x_+ - \rho\frac{w^T w}{\|w\|_2} + b &= -1 \\ w^T x_+ + b + 1 &= \rho\|w\|_2 \\ \rho &= \frac{2}{\|w\|_2} \end{aligned}$$

The last step follows by $x_+ \in H_+$. ○

By now we have precisely formulated "safeness" of classification given by a hyperplane. Now we want the "safest" classifier, i.e. with maximum margin. The optimization problem looks like

$$\begin{aligned} &\underset{w,b}{\text{maximize}} \ \frac{2}{\|w\|_2} \\ &\text{subject to} \ y_i(w^T x_i + b) \geqslant 1, \qquad i = 1, \dots, m. \end{aligned}$$

Equivalently in familiar minimization form we get the following.

Definition 2.2 (Hard SVM learning rule)**.** Let $S \in (\mathcal{X} \times \mathcal{Y})^m$ be a training set. Assuming the linearly separable case 2.1 the optimization problem given by

$$\begin{aligned} &\underset{w,b}{\text{minimize}} \ \frac{1}{2}\|w\|_2^2 \\ &\text{subject to} \ \ y_i(w^T x_i + b) \geqslant 1, \qquad i = 1, \dots, m, \end{aligned}$$

is called the *hard margin SVM* learning rule or just hard SVM. ◎

Remark 2.7 (Support vectors). Since hard SVM is a differentiable problem and inequalities constraints are affine by Slater's condition (1.52) strong duality holds. The Lagrangian takes the form

$$L(w, b, \lambda) = \frac{1}{2}\|w\|_2^2 - \sum_{i=1}^{m} \lambda_i (y_i(w^T x_i + b) - 1).$$

By (1.55) we get that $(w^\star, b^\star)$ is optimal if and only if there exists Lagrange dual variable $\lambda^\star$ such that the KKT conditions hold.

- Since $(w^\star, b^\star)$ minimizes $L(w^\star, b^\star, \lambda^\star)$ the gradient $\nabla_{w,b} L$ turns zero, i.e.

$$\nabla_w L(w^\star, b^\star, \lambda^\star) = w^\star - \sum_{i=1}^{m} \lambda_i^\star y_i x_i = 0 \qquad \Rightarrow \quad w^\star = \sum_{i=1}^{m} \lambda_i^\star y_i x_i, \tag{2.3}$$

$$\nabla_b L(w^\star, b^\star, \lambda^\star) = -\sum_{i=1}^{m} \lambda_i^\star y_i = 0 \qquad \Rightarrow \quad \sum_{i=1}^{m} \lambda_i^\star y_i = 0,$$

- Dual and primal feasibility

$$\lambda^\star \geqslant 0 \qquad\qquad y_i({w^\star}^T x_i + b^\star) \geqslant 1, \quad i = 1, \dots, m.$$

- Complementary slackness, i.e for $i = 1, \dots, m$ the following holds

$$\lambda_i^\star (y_i({w^\star}^T x_i + b) - 1) = 0 \quad \Rightarrow \quad \lambda_i^\star = 0 \vee y_i({w^\star}^T x_i + b^\star) = 1. \tag{2.4}$$

By (2.3) we see that the solution $w^\star$ takes the form of linear combination of data points x_i. Furthermore, by (2.4) x_i appears in the summation only when it lies in the the marginal hyperplanes $H_\pm$. Such x_i are called *support vectors.* Thus, $w^\star$ depends only on support vectors, while other x_j have no effect on the solution. ○

2.1.2 Soft SVM

Now we consider more realistic situation, when the training data are not linearly separable. This technique can be also viewed as a regularized version of hard margin SVM, when we want produce a larger margin, thus our model is less influenced by outliers.

Remark 2.8 (Non-separable case). If the training points (x_i, y_i), $i = 1, \dots, m$ are not linearly separable there does not exist a hyperplane defined by $w \in \mathbb{R}^n, b \in \mathbb{R}$ such that

$$y_i(w^T x_i + b) \geqslant 1.$$

Thus, for every $(x_i, y_i) \in S$ we introduce *slack variables* ξ_i indicating how much x_i violates linear separability, i.e.

$$y_i(w^T x_i + b) \geqslant 1 - \xi_i, \qquad i = 1, \dots, m.$$ ○

This approach brings us to an optimization problem that minimizes total slack amount and maximizes the margin. However, by reducing slack we always have a thinner margin, or conversely by maximization the margin we get more total slack, so we need to establish a trade-off between slack amount and margin thickness.

Definition 2.3 (Soft SVM learning rule)**.** Let $S \in (\mathcal{X} \times \mathcal{Y})^m$ be a linearly non-separable training set, $C \geqslant 0$ and $p \geqslant 1$. The optimization problem of the following form

$$\begin{aligned} \underset{w,b,\xi}{\text{minimize}} \ & \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^m \xi_i^p \\ \text{subject to} \ & y_i(w^T x_i + b) \geqslant 1 - \xi_i, \qquad i = 1, \dots, m, \\ & \xi_i \geqslant 0. \end{aligned}$$

is called the *soft margin SVM* learning rule or just soft SVM with trade-off C and slack penalization rate p. ◎

Remark 2.9 (Loss function). Setting penalization rate $p = 1$ we get a so called the *hinge loss*. For $p = 2$ we obtain the *quadratic hinge loss*. For the rest of this section we stick with the hinge loss. ○

Remark 2.10 (Soft SVM convexity). Since ξ_i is non-negative, ξ_i^p is a convex function, hence the objective is convex. Inequality constrains are affine in w, b, ξ, thus are convex. Thus, soft SVM is also a convex problem, in particular QP. ○

Remark 2.11 (Support vectors of soft SVM). The Lagrangian of soft SVM is given by

$$L(w, b, \xi, \lambda, \lambda') = \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^m \xi_i - \sum_{i=1}^m \lambda_i(y_i(w^T x_i + b) - 1 + \xi_i) - \sum_{i=1}^m \lambda'_i \xi_i.$$

Notice that soft margin SVM is a convex problem and has differentiable objective and constrains, and constrains are affine hence qualified (1.47). Now by the Corollary 1.55 $(w^\star, b^\star, \xi^\star)$ is optimal if and only if there exist $(\lambda^\star, \lambda'^\star)$ such that the KKT conditions hold. Let $L \equiv L(w^\star, b^\star, \xi^\star, \lambda^\star, \lambda'^\star)$. The KKT conditions are

- Primal solution $(w^\star, b^\star, \xi^\star)$ minimizes the Lagrangian

$$\nabla_w L = w^\star - \sum_{i=1}^m \lambda_i^\star y_i x_i = 0 \qquad \Rightarrow \qquad w^\star = \sum_{i=1}^m \lambda_i^\star y_i x_i, \tag{2.5}$$

$$\nabla_b L = -\sum_{i=1}^m \lambda_i'^\star y_i = 0 \qquad \Rightarrow \qquad \sum_{i=1}^m \lambda_i'^\star y_i = 0, \tag{2.6}$$

$$\nabla_{\xi_i} L = C - \lambda_i^\star - \lambda_i'^\star = 0 \qquad \Rightarrow \qquad \lambda_i^\star + \lambda_i'^\star = C. \tag{2.7}$$

- Feasibility

$$(\lambda^\star, \lambda'^\star) \geqslant 0 \qquad\qquad y_i({w^\star}^T x_i + b^\star) \geqslant 1 - \xi_i^\star, \quad i = 1, \dots, m.$$

- Complementary slackness, i.e for $i = 1, \dots, m$

$$\lambda_i^\star(y_i({w^\star}^T x_i + b^\star) - 1 + \xi_i^\star) = 0 \qquad \Rightarrow \qquad \lambda_i^\star = 0 \vee y_i({w^\star}^T x_i + b^\star) = 1 - \xi_i^\star, \tag{2.8}$$

$$\lambda'^\star_i \xi_i^\star = 0 \qquad \Rightarrow \qquad \lambda'^\star_i = 0 \vee \xi_i^\star = 0. \tag{2.9}$$

Again the solution takes the form of linear combination of training points (2.5), and all x_i with $\lambda^\star \neq 0$ (2.8) are support vectors. However, now two types of support vectors are presented, see second complementary slackness condition (2.9). If $\xi_i^\star = 0$, then x_i lies on $H_\pm$, thus $y_i({w^\star}^T x_i + b^\star) = 1$, otherwise x_i is considered to be an *outlier* with non-zero slack. ○

This brings us to a dual problem of soft support vector machines. Later we use the dual formulation to obtain kernel SVM, that consumes highly non-linear cases.

Remark 2.12 (Soft SVM dual formulation)*.* The standard form of the dual function for the soft SVM is the following

$$g(\lambda, \lambda') = \inf_{w,b,\xi} \left(\frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \lambda_i (y_i(w^T x_i + b) - 1 + \xi_i) - \lambda'^T \xi \right).$$

Notice that the Lagrangian is a convex function of (w, b, ξ), since the soft SVM is a convex problem and convexity is preserved under non-negative weighted sums. So L is minimized, when its gradient with respect to primal variables is equal to zero. Now applying (2.5), (2.6), (2.7) we obtain

$$C\sum_{i=1}^{m} \xi_i = \lambda^T \xi + \lambda'^T \xi, \tag{2.10}$$

$$\frac{1}{2}\|w\|_2^2 = \frac{1}{2}\sum_{i,j=1}^{m} \lambda_i \lambda_j y_i y_j x_i^T x_j,$$

$$\sum_{i=1}^{m} \lambda_i (y_i(w^T x_i + b) - 1 + \xi_i) = \sum_{i,j=1}^{m} \lambda_i \lambda_j y_i y_j x_i^T x_j + \underbrace{(\lambda^T y)}_{=0} b + 1^T \lambda - \lambda^T \xi.$$

Notice that λ' vanishes after we plug in (2.10), however we still must handle $\lambda' \geqslant 0$. To avoid this, by (2.7) it is valid to write $0 \leqslant \lambda_i \leqslant C$. Finally, a dual problem for SVM with linearly non-separable data is given by

$$\begin{aligned} \underset{\lambda}{\text{maximize}}\ & 1^T \lambda - \frac{1}{2}\sum_{i,j=1}^{m} \lambda_i \lambda_j y_i y_j x_i^T x_j \\ \text{subject to}\ & 0 \leqslant \lambda_i \leqslant C, \qquad i = 1, \ldots, m, \\ & \lambda^T y = 0. \end{aligned} \tag{2.11}$$

Surely we have a convex problem, namely QP. As was mentioned in (2.11) by Slater condition strong duality is attained. The resulting classifier h is given in the following form

$$h(z) = \operatorname{sign}\left(\sum_{i=1}^{m} \lambda_i y_i x_i^T z + b \right).$$

Here we get b from any support vector x_i, i.e. from every x_i with $0 < \lambda_i < C$, by $w^T x_i + b = y_i$. Consider the following

$$b = y_i - \left(\sum_{j=1}^{m} \lambda_j y_j x_i \right)^T x_i. \qquad \circ$$

2.1.3 Kernel SVM

Notice that in (2.11) data points x_i, x_j are used only in terms of dot product, and we do not rely on the vectors' representation directly. That actually motivates us to use some sophisticated vector space, where we can compute inner products and also exploit structure of the data to make more accurate predictions. Since soft margin SVM is a regularized problem, term C preserves the model from overfitting.

This brings us to kernel functions. As was mentioned earlier, kernel functions actually represent an inner product in a Hilbert space denoted by $\mathbb{H}$. To get data to a such vector space we need a feature mapping $\phi : \mathcal{X} \to \mathbb{H}$. Now we can evaluate the kernel function on every two images of data points under the feature map.

Thus, instead of dot product we rewrite soft SVM in terms of kernels as the following

$$\begin{aligned}
&\underset{\lambda}{\text{maximize}}\ 1^T\lambda - \frac{1}{2}\sum_{i,j=1}^{m} \lambda_i\lambda_j y_i y_j k(x_i, x_j)\\
&\text{subject to}\ \ 0 \leqslant \lambda_i \leqslant C, \qquad i = 1, \ldots, m,\\
&\qquad\qquad \lambda^T y = 0.
\end{aligned}$$

The resulting classifier is now given by

$$h(z) = \operatorname{sign}\left(\sum_{i=1}^{m} \lambda_i y_i k(x_i, z) + b\right).$$

Our final version in the matrix form is the following.

Definition 2.4 (Kernel SVM learning rule)**.** Suppose $S \in (\mathcal{X} \times \mathcal{Y})^m$ is a linearly non-separable training set. The optimization problem of the following form

$$\begin{aligned}
&\underset{\lambda}{\text{maximize}}\ 1^T\lambda - \frac{1}{2}(\lambda \circ y)^T\mathbf{K}(\lambda \circ y)\\
&\text{subject to}\ \ 0 \leqslant \lambda_i \leqslant C, \qquad i = 1, \ldots, m,\\
&\qquad\qquad \lambda^T y = 0,
\end{aligned}$$

is called the *kernel support vector machine* learning rule. Here $(\lambda \circ y)$ is a Hadamard product of λ, y, defined as $(\lambda \circ y)_i = \lambda_i y_i$ for $i = 1, \ldots, m$. ◎

2.2 Regression

Now we move on to regression problem. The main difference between regression and classification is given by the output space $\mathcal{Y}$, since in terms of regression $\mathcal{Y} = \mathbb{R}$. Thus, we do not attempt to have precise predictions, we rather want our predictions to be as close to the real ones as possible. The regression setting is the following

- Training set is given by $S \in (\mathcal{X} \times \mathcal{Y})^m$, where $\mathcal{X} \subset \mathbb{R}^n$ is an input space and $\mathcal{Y} \subset \mathbb{R}^n$ is an output space.

- The learning problem consists of establishing hypothesis $h \in \mathcal{H}$ form the hypothesis space with the smallest training error.

In this section we introduce two main learning rules, namely leastsquares and neural networks. As in the SVM case we start with linear hypothesis, then derive a regularized version and finally apply kernel methods on regularized least squares. In case of neural nets we describe the basic model called feed–forward neural network. This section is based on the [7, Chapter 11]

2.2.1 Ordinary least squares

We start with a strong assumption that our training data are linearly predictable.

Definition 2.5 (Linear predictable case)**.** Suppose $S \in (\mathcal{X} \times \mathcal{Y})^m$ is a training set. We say that S is linearly predictable for $i = 1, \dots, m$ labels y_i are linearly dependent on x_i, i.e.

$$\exists\, w \in \mathbb{R}^n, b \in \mathbb{R} \ : \ y_i \approx w^T x_i + b, \qquad \forall (x_i, y_i) \in S. \qquad \circledcirc$$

Remark 2.13 (Hypothesis space)*.* In linear predictable case hypothesis set can be formulated as the following

$$\mathcal{H} = \{h(x) = w^T x + b \mid (x, y) \in S\}, \qquad w \in \mathbb{R}^n,\ b \in \mathbb{R}.$$

Here vector $w \in \mathbb{R}^n$ is usually called a *slope* and scalar b a *bias* term. $\circ$

Remark 2.14 (Residual sum of squares)*.* We base the learning strategy on commonly used squared loss also called residual sum of squares or just RSS. It is given by

$$L_2(h(x_i), y_i) = (h(x_i) - y_i)^2.$$

This type of loss function defines a learning rule called ordinary least squares. In standard form the learning rule is given by the optimization problem

$$\underset{w,b}{\text{minimize}} \ \frac{1}{m} \sum_{i=1}^{m} ((w^T x_i + b) - y_i)^2.$$

However, we better introduce a matrix formulation. $\circ$

Definition 2.6 (Ordinary least squares learning rule)**.** Let $S \in (\mathcal{X} \times \mathcal{Y})^m$ be a training set, $\mathbf{X} \in \mathbb{R}^{(n+1)\times m}$ be a matrix given by placing m training points x_i into columns and adding $(n+1)$th row with all ones, $w \in \mathbb{R}^{n+1}$ be a vector of weights and a bias term, i.e. $w = (w_1, \dots, w_n, b)$ and $y = (y_1, \dots, y_m) \in \mathbb{R}^m$ be a vector of labels.

$$\mathbf{X} = \begin{bmatrix} | & | & | \\ x_1 & \cdots & x_m \\ | & | & | \\ 1 & \cdots & 1 \end{bmatrix} \qquad w = \begin{bmatrix} w_1 \\ \vdots \\ w_n \\ b \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

The optimization problem given by

$$\underset{w}{\text{minimize}} \ f_0(w) = \frac{1}{m} \|\mathbf{X}^T w - y\|_2^2$$

is called the *ordinary least squares* learning rule or simply least squares. $\circledcirc$

To see convexity of least squares problem recall example (1.25).

Remark 2.15 (Least squares solution)*.* By (1.25) we can derive analytical solution, setting the gradient to zero

$$\nabla f_0 = 0 \Leftrightarrow \frac{2}{m} \mathbf{X}(\mathbf{X}^T w - y) \Leftrightarrow \mathbf{X}\mathbf{X}^T w = \mathbf{X} y.$$

Depending whether $\mathbf{X}\mathbf{X}^T$ is invertible or not, the solution takes the following form

$$w^\star = \begin{cases} (\mathbf{X}\mathbf{X}^T)^{-1} \mathbf{X} y & \mathbf{X}\mathbf{X}^T \text{is invertible,} \\ (\mathbf{X}\mathbf{X}^T)^{\dagger} \mathbf{X} y & \text{otherwise.} \end{cases}$$

Here $(\mathbf{X}\mathbf{X}^T)^\dagger$ is a pseudo inverse, that can be obtained via singular value decomposition or QR decomposition. $\circ$

Remark 2.16 (Least squares pitfalls). Two main issues with the least squares solution is this: a problem of multicolliniarity, when matrix $\mathbf{X}^T$ has almost linear dependent columns, i.e. some features of x_i are highly correlated. In this case computation of matrix inverse of $\mathbf{X}\mathbf{X}^T$ is not numerically stable. Moreover, from statistical point of view resulting $w^\star$ will have high variance, so since data points are highly correlated even small change in the training set will cause large changes in $w^\star$. In this case our regression model will overfit the training dataset and loose generalization property. ○

2.2.2 Ridge regression

Since we have observed the bottlenecks of ordinary least squares let us introduce a more robust approach, namely ridge regression. Basically, ridge regression is a special example of least squares regularization methods.

As in the soft margin SVM we introduce a regularization term $\|w\|_2^2$.

Remark 2.17 (Ridge regression in standard form). Suppose $S \in (\mathcal{X} \times \mathcal{Y})^m$ is a training set, $w_1, \dots, w_m \in \mathbb{R}$, $b \in \mathbb{R}$ is a bias term, and $\beta > 0$. Ridge regression learning rule in the standard form is given by the following optimization problem

$$\underset{w,b}{\text{minimize}} \ \frac{1}{m}\sum_{i=1}^{m}(w^T x_i + b - y_i)^2 + \beta \sum_{i=1}^{m} w_i^2.$$

Here $\|w\|_2^2$ is a regularization term. ○

Remark 2.18 (Regularization intuition). One may ask how regularization term prevents linear model from overfitting? The intuition is the following, obviously penalty term might slightly influence the optimal value $p^\star$ on the training data, however it will significantly reduce the solution variance by filtering hypothesis with high slope. Thus, making β to big we obtain an underfit model, that barely captures the data structure. As the result, we wiggle a bit with β to find a suitable β. ○

Remark 2.19 (Bias term and data centering). Notice that by the Definition 2.8 we do not penalize the bias term, since there is no reason to prefer solutions that are close to the origin. So we rather reformulate the standard form of the ridge regression to get rid of the bias. Obviously, objective function of ridge regression is convex in b, namely quadratic, hence the global minimum attained, when first derivative with respect to b [3] is equal to zero, i.e.

$$\frac{1}{m}\sum_{i=1}^{m} 2w^T x_i + 2b - 2y_i = 0 \quad \Leftrightarrow \quad b = \underbrace{\frac{1}{m}\sum_{i=1}^{m} y_i}_{\bar{y}} - w^T \left(\underbrace{\frac{1}{m}\sum_{i=1}^{m} x_i}_{\bar{x}} \right).$$

Here $\bar{y}$ and $\bar{x}$ correspond to the mean value of y_i and x_i respectively. Now we rewrite the objective as the following

$$\frac{1}{m}\sum_{i=1}^{m}(w^T x_i + (\bar{y} - w^T\bar{x}) - y_i)^2 + \beta\sum_{i=1}^{m} w_i = \frac{1}{m}\sum_{i=1}^{m}(w^T(x_i - \bar{x}) - (y_i - \bar{y}))^2 + \beta\sum_{i=1}^{m} w_i.$$

Procedure of subtracting means is called *data centering* and can be done as a preprocessing step. ○

[3]Since b is a scalar.

We drop the constant $1/m$ and reformulate the problem (2.8) in terms of matrices.

Definition 2.7 (Ridge regression learning rule)**.** Suppose $S \in (\mathcal{X} \times \mathcal{Y})^m$ is a training set, $\mathbf{X}' \in \mathbb{R}^{n\times m}$ is a matrix with vectors $x_i - \bar{x}$ in its columns, $w \in \mathbb{R}^n$ is a vector of weights, $y' = (y_1 - \bar{y}, \dots, y_m - \bar{y}) \in \mathbb{R}^m$ is a vector of labels, and $\beta > 0$. The optimization problem

$$\underset{w}{\text{minimize}}\;\; f_0(w) = \|\mathbf{X}'^T w - y'\|_2^2 + \beta\|w\|_2^2$$

is called the *ridge regression* learning rule with regularization term $\|w\|_2^2$. ◎

Remark 2.20 (Notation). For the rest of this section we rather write $\mathbf{X}$ instead of $\mathbf{X}'$ and similarly y instead of y'. However, $\mathbf{X}$ and y will refer to centered data. ○

Remark 2.21 (Solution). We can see that (2.7) is a convex problem, since objective is a non-negative weighted sum of convex functions. Thus, the minimum is attained at the point, where the gradient vanishes. Now consider the following

$$\nabla f_0(w^\star) = 0 \Leftrightarrow (\underbrace{\mathbf{X}\mathbf{X}^T}_{\in\mathbb{S}^n_+} + \underbrace{\beta\mathbf{I}}_{\in\mathbb{S}^n_{++}})w^\star = \mathbf{X}y \Leftrightarrow w^\star = (\mathbf{X}\mathbf{X}^T + \beta\mathbf{I})^{-1}\mathbf{X}y.$$

The inverse always exists, since the matrix is defined by the sum of positive semidefine and positive definite matrices. ○

Remark 2.22 (Prediction). Once we have computed $w^\star$ we can make our prediction in the following form

$$h(z) = w^{\star T} z = z^T(\mathbf{X}\mathbf{X}^T + \beta\mathbf{I})^{-1}\mathbf{X}y.$$

Notice by our notation (2.20) the matrix $\mathbf{X}$ contains data with subtracted mean, so does the vector y, hence the bias term is implicitly included. ○

Before computing a dual we formulate an equivalent problem by introducing slack variables

$$\begin{aligned} &\underset{w,\xi}{\text{minimize}}\;\; \frac{1}{2}\beta\|w\|_2^2 + \frac{1}{2}\|\xi\|_2^2 \\ &\text{subject to}\;\; \xi = \mathbf{X}^T w - y. \end{aligned} \tag{2.12}$$

Remark 2.23 (Dual formulation). The Lagrangian corresponding to the (2.12) is given by

$$L(w,\xi,\nu) = \frac{1}{2}\beta\|w\|_2^2 + \frac{1}{2}\|\xi\|_2^2 + \nu^T(\xi - \mathbf{X}^T w + y).$$

Generally, for the dual problem we have

$$\underset{\nu}{\text{maximize}}\;\; \inf_{w,\xi}\left(\frac{1}{2}\beta\|w\|_2^2 + \frac{1}{2}\|\xi\|_2^2 + \nu^T(\mathbf{X}^T w - y)\right).$$

Notice if we reformulate constrains of (2.12) as inequalities, they will be affine in ξ and w, hence the Slater's condition holds (1.47) and the strong duality is attained. Now by (1.55) $w^\star, \xi^\star$ are optimal if and only if exists $\nu^\star$ such that the KKT conditions hold

- The gradient of the Lagrangian with respect to primal variables evaluated at $w^\star, \xi^\star$ vanishes, i.e. $w^\star, \xi^\star$ minimize the Lagrangian with respect to w, ξ

$$\nabla_w L(w^\star, \xi^\star, \nu^\star) = \mathbf{X}\nu^\star + \beta w^\star \qquad \Rightarrow \quad w^\star = \frac{1}{\beta}\mathbf{X}\nu^\star, \tag{2.13}$$

$$\nabla_\xi L(w^\star, \xi^\star, \nu^\star) = \xi^\star + \nu^\star = 0 \qquad \Rightarrow \quad \xi^\star = -\nu^\star. \tag{2.14}$$

- Feasibility
$$\xi^\star = w^{\star T}\mathbf{X} - y.$$

- Complementary slackness
$$\nu^{\star T}(\xi^\star - w^{\star T}\mathbf{X} + y) = 0 \quad \Rightarrow \quad \nu^\star = 0 \vee \xi^\star = w^{\star T}\mathbf{X} - y.$$

Since $w^\star, \xi^\star$ minimize the Lagrangian, we obtain the following

$$\begin{aligned}
&\inf_{w,\xi}\left(\frac{1}{2}\beta\|w\|_2^2 + \frac{1}{2}\|\xi\|_2^2 + \nu^T(\xi - \mathbf{X}^T w + y)\right)\\
&= \frac{1}{2}\beta\|w^\star\|_2^2 + \frac{1}{2}\|\xi^\star\|_2^2 + \nu^T(\xi^\star - \mathbf{X}^T w^\star + y)\\
&= \frac{1}{2}\beta\left\|\frac{1}{2\beta}\mathbf{X}\nu\right\|_2^2 + \frac{1}{2}\|-\nu\|_2^2 + \nu^T(-\nu - \mathbf{X}^T\mathbf{X}\nu + y)\\
&= \frac{1}{2\beta}\left(\mathbf{X}\nu\right)^T\left(\mathbf{X}\nu\right) + \frac{1}{2}\nu^T\nu - \nu^T\nu - (\mathbf{X}\nu)^T\mathbf{X}\nu + \nu^T y\\
&= -\|\mathbf{X}\nu\|_2^2 - \beta\|\nu\|_2^2 + 2\nu^T y\\
&= -\nu^T(\mathbf{X}^T\mathbf{X} + \beta\mathbf{I})\nu + 2\nu^T y
\end{aligned}$$

Now we can formulate a dual problem as the following

$$\underset{\nu}{\text{maximize}}\ \ -\nu^T(\mathbf{X}^T\mathbf{X} + \beta\mathbf{I})\nu + 2\nu^T y. \tag{2.15}$$

Since the dual variable ν corresponds to equality constrains we do not need constrain $\nu \geqslant 0$. ○

Remark 2.24 (Dual solution)*.* Obviously, objective in (2.15) is differentiable and strictly concave by the same argument as in (2.21). We get the solution setting the gradient to zero, i.e.

$$2(\mathbf{X}^T\mathbf{X} + \beta\mathbf{I})\nu = 2y \quad \Leftrightarrow \quad \nu = (\mathbf{X}^T\mathbf{X} + \beta\mathbf{I})^{-1}y.$$

Applying (2.13) we obtain final predictor in the following form

$$h(z) = w^{\star T}z = \frac{1}{\beta}(\mathbf{X}\nu^\star)^T z = \frac{1}{\beta}z^T\mathbf{X}(\mathbf{X}^T\mathbf{X} + \beta\mathbf{I})^{-1}y.$$

Again by notation (2.20) bias term is implicitly included. ○

2.2.3 Kernel ridge regression

Now we formulate desired version of regularized linear regression in terms of kernels, so our final model will be able to deal with highly non-trivial data. To formulate a kernel version we use the results of (2.15), that does not depend on representation of training data and requires only inner products.

Definition 2.8 (Kernel ridge regression learning rule)**.** Suppose

- $S \in (\mathcal{X} \times \mathcal{Y})^m$ is a training set,
- $\mathbf{X} \in \mathbb{R}^{n\times m}$ is a matrix with centered vectors $x_i - \bar{x}$ in its columns,
- $y \in \mathbb{R}^m$ is a centered vector of labels,
- $\nu \in \mathbb{R}^m$ is the Lagrangian dual variable,

- $\beta > 0$ is a free positive parameter
- $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel function with matrix $\mathbf{K} \in \mathbb{S}^n_+$ such that $\mathbf{K}_{ij} = k(\mathbf{X}_{\bullet i}, \mathbf{X}_{\bullet j})$.

The optimization problem defined as

$$\underset{\nu}{\text{maximize}} \ -\nu^T(\mathbf{K} + \beta\mathbf{I})\nu + 2\nu^T y$$

is called the *kernel ridge regression* learning rule or just KRR. ⊚

Remark 2.25 (Solution). Solution is given by the dual solution, however we use kernel matrix $\mathbf{K}$ instead of $\mathbf{X}^T\mathbf{X}$, i.e.

$$\nu = (\mathbf{K} + \beta\mathbf{I})^{-1}y.$$

Recall that the solution is obtained for the centered data 2.19. Let $\phi : \mathcal{X} \to \mathbb{H}$ be a feature map to a feature space $\mathbb{H}$, then our prediction is given by a hypothesis h

$$\begin{aligned} h(z) = w^{\star T}\phi(z) &= \frac{1}{\beta}\left(\sum_{i=1}^{m}\langle\phi(z), \phi(x_i)\rangle_{\mathbb{H}}\right)^T (\mathbf{K} + \beta\mathbf{I})^{-1}y \\ &= \frac{1}{\beta}\left(\sum_{i=1}^{m} k(z, x_i)\right)^T (\mathbf{K} + \beta\mathbf{I})^{-1}y \end{aligned}$$

○

2.2.4 Neural Networks

We end our short tour in supervised learning models with a powerful tool called neural network[4]. Neural network, or artificial neural network, or neural net or just NN is a mathematical model inspired by activity of human brain. Even though neural networks seems to be mysterious they are just non-linear statistical models. We describe basic technique called a *feed–forward* NN. Neural nets can handle both discrete and continuous labels, we look at both cases and in the end we formalize the learning rule for regression. This chapter summarizes [9, Chapter 11] and [10, Chapter 20].

Remark 2.26 (Feed–forward neural network basic setting).

- A neural network consists of $d + 1$ disjoint sets $V_0, \dots, V_d$ of *neurons* called *layers*. Size of the layer V_k is denoted by $|V_k|$. Layer V_0 is called the *input layer* and V_d the *output layer*. Layers $V_1, \dots, V_{d-1}$ are called *hidden layers*, for the reasons that values of hidden neurons are not directly observed.
- A neuron is an atomic element of a neural net. In feed-forward nets neurons are only connected with neurons of previous and succeeding layers. We denote the ith neuron in the kth layer by $v_{k,i}$.
- Each connection (edge) between $v_{k,i}$ and $v_{k+1,j}$ has a weight $w^{(i,j)}_{k+1}$.
- A non-linear activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is a function that fires the neurons in response to the incoming stimuli. Usually we have one fixed activation function for whole NN. Almost every time we choose σ to be differentiable. Non-differentiability of σ was the main reason caused the interest decay in neural networks after their invention.

[4]Neural nets can be applied to unsupervised learning problems as well.

- An output function $g : \mathbb{R} \to \mathbb{R}$, that is only used in the output layer to produce the final result. We call g a *final transformation.* Obviously, we want g to be differentiable as well. $\circ$

Now we work out the matrix notation to represent the whole neural net as a function.

Remark 2.27 (Feed-forward neural network matrix setting).

- For every $v_{k+1,j}$ we define a vector of weights of incoming edges as
$$w_{k+1,j} = \left(w_{k+1}^{(1,j)}, w_{k+1}^{(2,j)}, \dots, w_{k+1}^{(|V_k|,j)}\right) \in \mathbb{R}^{|V_k|}. \tag{2.16}$$
- Since every $v_{k+1,j}$ has its own $w_{k+1,j}$ we can define a matrix $\mathbf{W}_{k+1}$ of weights for each $k+1$ layer as the following
$$\mathbf{W}_{k+1} = \begin{bmatrix} | & | & | & | \\ w_{k+1,1} & w_{k+1,2} & \dots & w_{k+1,|V_{k+1}|} \\ | & | & | & | \end{bmatrix} \in \mathbb{R}^{|V_k| \times |V_{k+1}|}.$$
- Now for every $v_{k+1,j}$ we define a function of the output $f_{k+1,j} : \mathbb{R}^{|V_k|} \to \mathbb{R}$ as
$$f_{k+1,j}(x) = \sigma\left(\sum_{l=1}^{|V_k|} w_{k+1}^{(l,j)} f_{k,l}(x)\right).$$
Notice that for every neuron in the first hidden layer V_1 we have
$$f_{1,j}(x) = \sigma\left(\sum_{l=1}^{|V_0|} w_1^{(l,j)} x_l\right).$$
Here V_0 simply returns $x \in \mathbb{R}^n$, moreover the size of V_0 often corresponds to the dimension of x, i.e. $n = |V_0|$.
- The output function of one neuron $v_{k+1,j}$ can be generalized to the output function of the whole $k+1$ layer, so we have $f_{k+1} : \mathbb{R}^{|V_k|} \to \mathbb{R}^{|V_{k+1}|}$ defined as
$$f_{k+1}(x) = \tilde{\sigma}\left(\mathbf{W}_{k+1}^T f_k(x)\right).$$
Here we extend the activation function to be vector valued, so it handles whole layers, i.e. $\tilde{\sigma} : \mathbb{R}^{|V_{k+1}|} \to \mathbb{R}^{|V_{k+1}|}$ is given by
$$\tilde{\sigma}\left(\mathbf{W}_{k+1}^T f_k(x)\right) = \begin{bmatrix} \sigma(w_{k+1,1}^T f_k(x)) \\ \sigma(w_{k+1,2}^T f_k(x)) \\ \vdots \\ \sigma(w_{k+1,|V_{k+1}|}^T f_k(x)) \end{bmatrix} \in R^{|V_{k+1}|}.$$
- Now for fixed $V_1, \dots, V_d$ and σ, g the neural network can be represented by a function
$$f(x) = \tilde{g}(x) = \tilde{g}\left(\mathbf{W}_d^T \tilde{\sigma}(\dots \tilde{\sigma}(\mathbf{W}_1^T x)\dots)\right),$$
where $\tilde{g} : \mathbb{R}^{|V_{d-1}|} \to \mathbb{R}^{|V_d|}$ is a vector valued extension of a scalar function g.

- We rather implicitly denote that the output of a neural net corresponds to the specific $\mathbf{W}_1, ...\mathbf{W}_d$, by the index notation
$$f_\Theta(x), \qquad \Theta = (\mathbf{W}_1, ...\mathbf{W}_d). \qquad \circ$$

Remark 2.28 (Bias term). Basically, we do not require the neural net f_Θ to pass through the origin, so we need to propagate some bias terms. This issue is solved by adding the extra neuron to each layer, except the output one. This neuron does not take any input and always returns 1, but with the weight assigned it takes the form of a some constant. For simplicity suppose that our neural network implicitly comes with biases. ○

Definition 2.9 (Architecture of a neural network)**.** Let (V, σ, g) be a triplet such that, $V = (V_1, \ldots, V_d)$ is a d-tuple of layers, $\sigma : \mathbb{R} \to \mathbb{R}$ an activation function, $g : \mathbb{R} \to \mathbb{R}$ is an output function. We call (V, σ, g) a *neural network architecture.* ◎

Definition 2.10 (Neural network hypothesis space)**.** Let (V, σ, g) be a neural network architecture. We define the *neural network hypothesis space* as the following
$$\mathcal{H}_{(V,\sigma,g)} = \left\{ h_\Theta : \mathbb{R}^{|V_0|} \to \mathbb{R}^{|V_d|} \;\middle|\; \Theta = (\mathbf{W}_1, \ldots, \mathbf{W}_d) \right\},$$
where Θ is a d-tuple of weights. ◎

Example 2.29 (NN with two hidden layers)**.** According to our notation neural net for fixed Θ and architecture (V, σ, g) with two hidden layers takes the form
$$f_\Theta(x) = \tilde{g}\left(\mathbf{W}_3^T \tilde{\sigma}\left(\mathbf{W}_2^T \tilde{\sigma}\left(\mathbf{W}_1^T x\right)\right)\right). \qquad \bigcirc$$

Remark 2.30 (Choice of activation function). The performance of a network highly depends on the activation function. If we choose a linear σ the net collapses to a linear model, hence generally NN can be viewed as non-linear generalization of linear models. Common choice for activation function are:

- sigmoid function $k : \mathbb{R} \to \mathbb{R}$
$$k(x) = \frac{1}{1 + e^{-x}},$$
- `softPlus` or `smoothReLU` $k : \mathbb{R} \to \mathbb{R}$
$$k(x) = \log(1 + e^x). \qquad \circ$$

Remark 2.31 (Choice of output function). As we stated earlier, neural nets can handle both classification problems and regression ones. Roughly, the difference lies in the output function.

- Common choice for the regression is the *identity function* $x \mapsto x$. Moreover, dealing with regression the output layer usually contains only one neuron, i.e.
$$f_\Theta(x) = g\left(w_{d,1}^T f_{d-1}(x)\right) = w_{d,1}^T f_{d-1}(x)$$
- With l-class classification the number of neurons in the output layer is l and common choice of the output function that is so-called *softmax* function $h : \mathbb{R}^m \to \mathbb{R}^m$ defined as
$$h(x) = \frac{e^x}{1^T e^x}, \qquad e^x = (e^{x_1}, \ldots, e^{x_m}).$$
The whole neural net takes the form
$$f_\Theta(x) = h(\mathbf{W}_d^T f_{d-1}(x)). \qquad \circ$$

So far we introduced all terminology related to evaluation a neural network at a point x. However nothing was said about the neural net learning rule. For these reasons we define a loss function that tells the difference between our predictions $f_\Theta(x_i)$ and real labels y_i.

Remark 2.32 (Loss function). In general, regardless the problem type we denote the loss as function of weights as $L(\Theta)$. Suppose $|V_d| = 1$, a common choice for regression is familiar residual sum of squares loss

$$L(\Theta) = \sum_{(x,y)\in S} (y - f_\Theta(x))^2 \,.$$

For classification cross-entropy is usually used

$$L(\Theta) = -\sum_{(x,y)\in S} y^T \log(f_\Theta(x)).$$

Here we use the vector valued extension for the log function

$$\log(f_\Theta(x)) = \left(\log(w_{d,1}^T f_{d-1}(x)), \dots, \log(w_{d,|V_d|}^T f_{d-1}(x))\right). \qquad \circ$$

Since we have defined loss functions for regression and classification we are ready to introduce the neural net learning rule. Here we demonstrate the learning rule with the regression problem.

Definition 2.11 (Neural network regression learning rule)**.** Suppose $\mathcal{X} \subset \mathbb{R}^n$ and $\mathcal{Y} \subset \mathbb{R}^m$, a set $S \in (\mathcal{X} \times \mathcal{Y})^m$ is the training set and

$$\mathcal{H}_{(V,\sigma,g)} = \left\{ h_\Theta : \mathbb{R}^{|V_0|} \to \mathbb{R}^{|V_d|} \,\middle|\, \Theta = (\mathbf{W}_1, \dots, \mathbf{W}_d) \right\}$$

is the hypothesis space corresponding to the neural network architecture (V, σ, g) with $|V_d| = 1$. The optimization problem given by

$$\underset{\Theta}{\text{minimize}} \sum_{(x,y)\in S} (y - h_\Theta(x))^2,$$

is called the *neural network regression* learning rule . ◎

Since we formulated the neural network learning rule as an optimization problem we introduce the famous learning algorithm, that will be described in great details in the next chapter.

Remark 2.33 (Back-propagation algorithm). As we already stated, evaluation of a network with fixed Θ and (V, σ, g) can be written as $f_\Theta(x)$. The process of output computation of a neural network for given x is called the *forward pass*, which is the first step of the back-propagation algorithm.

The second step is to compute the error for all pairs $(x, y) \in S$ and propagate this error to the preceding layers. We propagate the error and update the weights with the help of the gradient and the chain rule. This step is called the *backward pass*.

These two passes are repeated until convergence. In machine learning literature this two-pass algorithm is usually called *back-propagation algorithm*, in context of the mathematical optimization we usually call it the *gradient descent*, moreover we will discus it in the next chapter. ∘

Remark 2.34 (Methods of learning). Notice that in the backward pass we compute the error for the whole training set, and only then we update the weights. If the training set is not very small, such learning consumes a lot of time, so we better slightly modificate the objective for a faster performance.

- *Mini-batch learning* corresponds to random selection of a subset $C \subset S$ and compute an error within C, i.e. $\sum_{(x,y) \in C} (y - h_\Theta(x))^2$.
- In *online learning* case we use only one training point at a time, so the objective now is given by $(y - h_\Theta(x))^2$. ○

Although neural network learning rule seems to be a powerful learning strategy it has some drawbacks.

Remark 2.35 (issues with neural networks).

- We can see that learning a neural network requires establishing a huge amount of weights Θ, hence the first pitfall is *overparametrization.* Usually finding optimal $\Theta^\star$ causes overfitting, i.e. the final predictor excessively fits the training data, and at the same time has pour generalization properties and fails to predict unseen data. To avoid this situation according to the definition of the vector of weights (2.16) we can introduce a regularization term

$$J(\Theta) = \sum_{i=1}^{d} \sum_{j=1}^{|V_i|} \|w_{i,j}\|_2^2.$$

  Now the objective function can be formulated as

$$L(\Theta) + \lambda J(\Theta), \qquad \lambda \geqslant 0,$$

  with trade-off parameter λ.

- However, a more important drawback is that learning a neural network is generally a non-convex optimization problem, hence we can stuck in some local minimum. To have an intuition of non-convexity, consider that the whole network f_Θ for some architecture (V, σ, g) is not guaranteed to be a convex function of Θ. It can be shown that even for only one neuron, sigmoid activation function and residual sum of squares loss the objective has exponential number of local minima [11]. ○

2.3 Clustering

So far we have discussed supervised problems, where we used labels to estimate the performance of our model. Now we consider a widely known unsupervised task of determination some hidden patterns in the data, namely, we need to find some classes and then associate each data point with some class. In case of supervised learning these classes were given beforehand, here they are unknown and we need to work them out. In terms of machine learning this problem is called *clustering.* Usually establishing such clusters provides us with non-trivial insights of the dataset, unfortunately in most cases we have no idea about data relations.

Intuitively, we can formulate clustering as a dividing the original set of objects into well organized groups. By well organized cluster we can imagine a set, where its elements are very similar at the same time are very different comparing to elements from other clusters. In general, we are looking for some function that assigns each data point to some cluster.

There exists many approaches to determine a clustering of the data, but we focus on a famous one called k-means learning rule. Then we describe a widely used heuristic algorithm called k-means algorithm, that will converge to some local minima, since the learning rule is not convex.

Clustering setting is the following

- Unlabelled training set $S \subset \mathbb{R}^n,\ S = \{x_1, \dots, x_m\}$.
- Number of clusters k.
- Hypothesis space of functions $\mathcal{H} = \{h : S \to \{1, \dots, k\}\}$ mapping every training point to some cluster. Notice that our hypothesises are functions with discrete domain. A better way to understand function h is to take it as an element of $\{1, \dots, k\}^m$, here we use $x \in S$ as index in h, i.e. $h_x \in \{1, \dots, k\}$.
- Target function $h^\star$ to be found is a solution to the optimization problem.

2.3.1 K-means learning rule

We base our learning rule on minimization of within the cluster dissimilarity.

Definition 2.12 (Dissimilarity learning rule)**.** Suppose $S = \{x_1, \dots, x_m\} \subset \mathbb{R}^n$ is the training set and $\mathcal{H} = \{h : S \to \{1, \dots, k\}\}$ is the hypothesis space. The optimization problem defined as

$$\underset{h \in \mathcal{H}}{\text{minimize}} \sum_{l=1}^{m} \frac{1}{2|h^{-1}(l)|} \sum_{\substack{x,y:\\ h_x = h_y = l}} \|x - y\|_2^2, \tag{2.17}$$

is called the *within the cluster dissimilarity* learning rule. We denote by $|h^{-1}(l)|$ size of the lth cluster. ◎

One may notice that (2.21) does not implicitly take into account the distance between clusters so subtracting corresponding term we obtain

$$\underset{h \in \mathcal{H}}{\text{minimize}} \left(\sum_{l=1}^{m} \frac{1}{2|h^{-1}(l)|} \sum_{\substack{x,y:\\ h_x = h_y = l}} \|x - y\|_2^2 \right) - \frac{1}{2} \sum_{\substack{x,y:\\ h_x \neq h_y}} \|x - y\|_2^2. \tag{2.18}$$

Lemma 2.13 (Target function and cluster distance)**.** The solution $h^\star$ of (2.21) also solves (2.18).

Proof. Consider the second sum in optimization problem (2.18), we express it in the complement form, as the distance between all points minus the distance between points in the same cluster.

$$\sum_{\substack{x,y:\\ h_x \neq h_y}} \|x - y\|_2^2 = \sum_{x,y \in S} \|x - y\|^2 - \sum_{\substack{x,y:\\ h_x = h_y}} \|x - y\|_2^2 = A - \sum_{l=1}^{m} \sum_{\substack{x,y:\\ h_x = h_y = l}} \|x - y\|_2^2.$$

Here A is a non-negative constant for a given dataset. Now we express (2.18) as the following

$$\underset{h}{\text{minimize}} \left(\sum_{l=1}^{m} \frac{1}{|h^{-1}(l)|} \sum_{\substack{x,y:\\ h_x = h_y = l}} \|x - y\|_2^2 \right) - \frac{1}{2} A.$$

The above problem differs from (2.21) by positive scaling 2 and subtracting a non-negative constant, hence the solution remains the same. □

Now we look at the useful concept in clustering called center of mass.

Definition 2.14 (Center of mass)**.** Let $S = \{x_1, \dots, x_m\} \subset \mathbb{R}^n$ be a set. A *center of mass* of S is defined as the expected mean of its elements, i.e.

$$\mu_S = \frac{1}{m} \sum_{i=1}^{m} x_i.$$ ◎

Remark 2.36 (Expected distance from a center of mass). The expected squared distance between points and the center of mass can be expressed as the following

$$\begin{aligned}\frac{1}{m}\sum_{l=1}^{m}\|x_l-\mu_S\|_2^2 &= \frac{1}{m}\sum_{l=1}^{m}\|x_l\|_2^2-\frac{2}{m^2}\sum_{l,j=1}^{m}x_l^Tx_j+\frac{1}{m}\sum_{l=1}^{m}\frac{1}{m^2}\sum_{i,j=1}^{m}x_i^Tx_j\\ &= \frac{1}{m}\sum_{l=1}^{m}\|x_l\|_2^2-\frac{1}{m^2}\sum_{i,j=1}^{m}x_i^Tx_j \qquad (2.19)\end{aligned}$$

○

Lemma 2.15 (Center of mass solution)**.** Suppose $S=\{x_1,\dots,x_m\}$. The optimization problem

$$\underset{\beta}{\text{minimize}}\ \frac{1}{m}\sum_{i=1}^{m}\|x_i-\beta\|_2^2$$

is solved by the center of mass μ_S

Proof. Notice that the objective as a function of β is convex, since it is a non-negative weighted sum of squared norms precomposed with affine functions $g_i(\beta)=-\mathbf{I}\beta+x_i$. Thus we set its gradient equal to zero to get the minimum

$$\frac{1}{m}\sum_{i=1}2(-x_i+\beta)=0 \quad\Rightarrow\quad \beta=\frac{1}{m}\sum_{i=1}x_i=\mu_S.$$ □

Remark 2.37 (Equivalent problem). Let us reformulate the optimization problem (2.21) as the following

$$\begin{aligned}\sum_{l=1}^{k}\frac{1}{2|h^{-1}(l)|}\sum_{\substack{x,y:\\h_x=h_y=l}}\|x-y\|_2^2 &= \sum_{l=1}^{k}\frac{1}{2|h^{-1}(l)|}\sum_{\substack{x,y:\\h_x=h_y=l}}x^Tx-2x^Ty+y^Ty\\ &= \sum_{l=1}^{k}\frac{1}{2|h^{-1}(l)|}\left(\sum_{\substack{x,y:\\h_x=h_y=l}}x^Tx-2\sum_{\substack{x,y:\\h_x=h_y=l}}x^Ty+\sum_{\substack{x,y:\\h_x=h_y=l}}y^Ty\right)\\ &= \sum_{l=1}^{k}\frac{1}{|h^{-1}(l)|}\left(|h^{-1}(l)|\sum_{\substack{x:\\h_x=l}}\|x\|_2^2-\sum_{\substack{x,y:\\h_x=h_y=l}}x^Ty\right)\\ &= \sum_{l=1}^{k}\sum_{\substack{x:\\h_x=l}}\|x-\mu_l\|_2^2.\end{aligned}$$

Now optimization problem (2.21) takes the form

$$\underset{h}{\text{minimize}}\ \sum_{x\in S}\|x-\mu_{h_x}\|_2^2. \qquad (2.20)$$

Actually, the problem takes its name after formulation above. ○

Remark 2.38 (Centroid). In terms of clustering problem we call the center of mass of a lth cluster just *centroid*, and denote as

$$\mu_k=\frac{1}{|h^{-1}(l)|}\sum_{x:h_x=l}x$$ ○

Now we are ready to present the standard k-means learning rule.

Definition 2.16 (K-means learning rule)**.** Suppose $S = \{x_1, \dots, x_m\} \subset \mathbb{R}^n$ is the training set and $\mathcal{H} = \{h : S \to \{1, \dots, k\}\}$ is the hypothesis space. The optimization problem defined as

$$\underset{h}{\text{minimize}} \sum_{x \in S} \|x - \mu_{h_x}\|_2^2. \tag{2.21}$$

is called the *k-means* learning rule. Here μ_{h_x} corresponds to the centroid of the cluster given by h, where x is assigned. ◎

Remark 2.39 (Analysis of k-means problem). Even though the problem (2.20) does not seem to be difficult, the domain of the objective is discrete since the number of possible clusters is finite. Thus the objective is not continuous and is non-differentiable, so we can not apply classical derivative based approaches. Actually, this problem is NP-hard [12]. To solve this problem we can use popular heuristic algorithm that rapidly converges to some local minima. Unfortunately, no guarantees are given on the quality of the result, so common practice is to run that algorithm more times with different initial states. ○

2.3.2 K-means algorithm

In this subsection we briefly describe the k-means algorithm that is often used for minimization the k-means learning rule.

Algorithm 1 K-means

```
1:  procedure K-MEANS($S, k$)
2:      $i = 1$
3:      $\mu_1^{(i)}, \dots, \mu_k^{(i)}$ =RANDOM_CENTROIDS()
4:      $\forall x \in S:\ h^{(i)}(x) = \underset{1 \leqslant l \leqslant k}{\text{argmin}}\ \|x - \mu_l^{(i)}\|$
5:      $v^{(i)}$ = COMPUTE_OBJECTIVE($h^{(i)}$)
6:      $v^0 = \infty$
7:      while (IS_CONVEGING($v^{(i-1)}, v^{(i)}$)) do
8:          $i = i + 1$
9:          $\mu_1^{(i)}, \dots, \mu_k^{(i)}$ = RECOMPUTE_CENTROIDS($h^{(i-1)}$)
10:         $\forall x \in S:\ h^{(i)}(x) = \underset{1 \leqslant l \leqslant k}{\text{argmin}}\ \|x - \mu_l^{(i)}\|$
11:         $v^{(i)}$ = COMPUTE_OBJECTIVE($h^{(i)}$)
12:     return $h^{(i)}(\cdot)$
```

Lemma 2.17 (K-means algorithm correctness)**.** The k-means Algorithm 1 will stop and return some locally optimal clustering assignment.

Proof. Notice that the algorithm can not return a clustering until it go through the loop. New centroids replacing $\mu_1^{(i-1)}, \dots, \mu_k^{(i-1)}$ with $\mu_1^{(i)}, \dots, \mu_k^{(i)}$ for clusters given by $h^{(i-1)}$ will not increase the objective by (2.15). Assigning all $x \in S$ to the nearest cluster will not increase the objective as well. Thus every iteration does not increase the value of the objective, hence the algorithm converges. However we are not guaranteed to have a global optimal value, so the minimum attained is a local one. □

Chapter 3

Algorithms

So far we have discussed theoretical background and real problems in knowledge engineering, although nothing was said how to solve these problems. The goal of the chapter is to show basic methods for theoretical problems described in the first chapter and practical cases in chapter two as well. This chapter summarizes [1, chapters 9-11].

3.1 Unconstrained optimization methods

We start our tour in optimization techniques with the simplest case when there are no constraints presented.

Definition 3.1 (Unconstrained minimization problem)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex twice continuously differentiable function, hence domain of f is open. An optimization problem given by

$$\text{minimize} \;\; f(x)$$

with $p^\star = \inf f(x)$ finite and attained, is called the convex *unconstrained minimization problem.* ◎

Since the problem is convex, by Corollary 1.39 we are looking for such an $z \in \mathbf{dom}\, f$ that $\nabla f(z) = 0$.

Remark 3.1 (Quadratic objective). Notice that if the objective f is a quadratic function in x, then $\nabla f(x) = 0$ is a system of linear equations, hence can be solved analytically. Thus, we would consider non-quadratic problems where, iterative methods can be applied. ○

Remark 3.2 (Iterative methods). An iterative algorithm is a technique of finding sequence of points $\{x^{(k)}\}_{k=0}^{\infty} \in \mathbf{dom}\, f$ such that $f(x^{(k)}) \xrightarrow{k\to\infty} p^\star$ with initial point $x^{(0)}$. This sequence is called a *minimizing sequence*. We usually use some $\epsilon > 0$ representing a tolerated error, so the stopping criterion is given by $f(x^{(k)}) - p^\star \leqslant \epsilon$. Since we have assumed that $p^\star \in \mathbb{R}^n$ and attained this sequence converges. ○

Assumptions We assume that the set

$$S = \left\{ x \in \mathbb{R}^n \;\middle|\; f(x) \leqslant f(x^{(0)}) \right\}$$

is closed. The set S is usually called *α-sublevel set* with $\alpha = f(y)$, in our case $\alpha = f(x^{(0)})$. The intuition is simple, if S is not closed the algorithm might converge to the point that is not in the domain. Unfortunately, this condition is not easy to verify, but if all sublevel sets are closed or, equivalently, $\mathbf{epi}\, f$ is closed the condition trivially holds. Another way to say that all sublevel sets are closed is that function values goes to infinitely when x reaches the boundary of the domain of f.

We also suppose that problem given by the Definition 3.1 is strongly convex on the sublevel set S, i.e.

$$\nabla^2 f(x) \succcurlyeq m\mathbf{I}, \qquad x \in S \tag{3.1}$$

for some $m > 0$. Since f is twice differentiable, for $x, y \in S$ using Taylor theorem we express $f(y)$ as

$$f(y) = f(x) + \nabla f(x)^T(x-y) + \underbrace{\frac{1}{2}(y-x)^T\nabla^2 f(z)(y-x)}_{\substack{\text{remainder in the Lagrange form,}\\ z=\theta x+(1-\theta)y,\ 0\leqslant\theta\leqslant 1}}.$$

By strong convexity the remainder can be rewritten as

$$\frac{1}{2}(y-x)^T\nabla^2 f(z)(y-x) \geqslant \frac{m}{2}(y-x)^T\mathbf{I}(y-x).$$

Thus, we obtain better lower bound for $f(y)$ than first order condition rom the Theorem 1.24

$$f(y) \geqslant f(x) + \nabla f(x)^T(x-y) + \frac{m}{2}\|y-x\|_2^2. \tag{3.2}$$

We now show how the above inequality can be used to measure suboptimality of the point $x \in S$. Notice that the right hand side of the inequality is a convex quadratic function of y, so setting its gradient to zero we obtain $\tilde{y} = x - (1/m)\nabla f(x)$ that minimizes the right hand side. We now rewrite the inequality above with $\tilde{y}$ instead of y

$$f(y) \geqslant f(x) - \frac{1}{2m}\|\nabla f(x)\|_2^2.$$

Notice that the result above holds for every $y \in S$, so taking $y^\star \in S$ such that $p^\star = f(y^\star)$, we have

$$f(x) - p^\star \leqslant \frac{1}{2m}\|\nabla f(x)\|_2^2. \tag{3.3}$$

Therefore, inequality (3.3) can be viewed as a suboptimality criterion, i.e. if the gradient is small then $f(x)$ is near the optimum, i.e.

$$\|\nabla f(x)\|_2 \leqslant \sqrt{2m\epsilon} \implies f(x) - p^\star \leqslant \epsilon.$$

Here we are talking about *conceptual stopping criterion*, since m is actually unknown, however we can use it to derive a practical one. We pick $\eta > 0$, such that $\|\nabla f(x)\|_2 \leqslant \eta$. When η is small enough, it will to be very likely smaller than $\sqrt{2m\epsilon}$, hence $f(x) - p^\star \leqslant \epsilon$.

By the inequality (3.2) we also have that all off sublevel sets in S are bounded, so S is bounded as well. Since $\nabla^2 f(x)$ is a continuous function of x by the Definition 3.1, we have that the maximum eigenvalue of $\nabla^2 f(x)$ is bounded above on S, in other words there exists $M > 0$ such that for all $x \in S$ we get

$$\nabla^2 f(x) \preccurlyeq M\mathbf{I}.$$

In similar way as in (3.2) we obtain the following inequality

$$f(y) \leqslant f(x) + \nabla f(x)^T(x-y) + \frac{M}{2}\|y-x\|_2^2. \tag{3.4}$$

Now we introduce general descent method and discuss basic ideas and related terminology.

Algorithm 2 General descent method

```
1: procedure General descent($x \in \mathbf{dom}\, f$)
2:     while not (stoping criterion) do
3:         Determine a descent direction $\Delta x$
4:         Choose $t > 0$ via line search
5:         $x = x + t\Delta x$
6:     return $x$
```

Remark 3.3 (Terminology). The Algorithm 2 produces a minimizing sequence of points $x^{(k+1)} = x^{(k)} + t^{(k)}\Delta x^{(k)}$ such that $f(x^{(k+1)}) < f(x^{(k)})$. Here $t^{(k)} > 0$ is called *step* and $\Delta x^{(k)} \in \mathbb{R}^n$ is called *descent direction*. ○

Remark 3.4 (Descent direction). Using the first order condition 1.24 get the following

$$\nabla f(x^{(k)})^T(y - x^{(k)}) \geqslant 0 \implies f(y) \geqslant f(x^{(k)}),$$

hence to get $f(x^{(k+1)}) < f(x^{(k)})$ we want Δx make a negative inner product with the gradient, i.e.

$$\nabla f(x^{(k)})^T \Delta x < 0.$$ ○

Line search The line search in the fourth step corresponds to finding such t along the ray $\{x + t\Delta x \mid t > 0\}$ the value of the objective decreases, i.e. $f(x) > f(x + t\Delta x)$.

One way to determine t is the *exact line search*. We define a function of one variable $\tilde{f}(t) = f(x + t\Delta x)$ and now t is given by

$$t_{\text{exact}} = \underset{s \geqslant 0}{\operatorname{argmin}}\ \tilde{f}(s).$$

Here we minimize a convex function of one variable, that can be done by setting the first derivative of $\tilde{f}(t)$ to zero and finding the root by bisection.

However, in practice we are also satisfied with t that approximately minimize $\tilde{f}$. A popular inexact method of line search is called *backtracking line search*. Here we find t using the following algorithm.

Algorithm 3 Backtracking line search

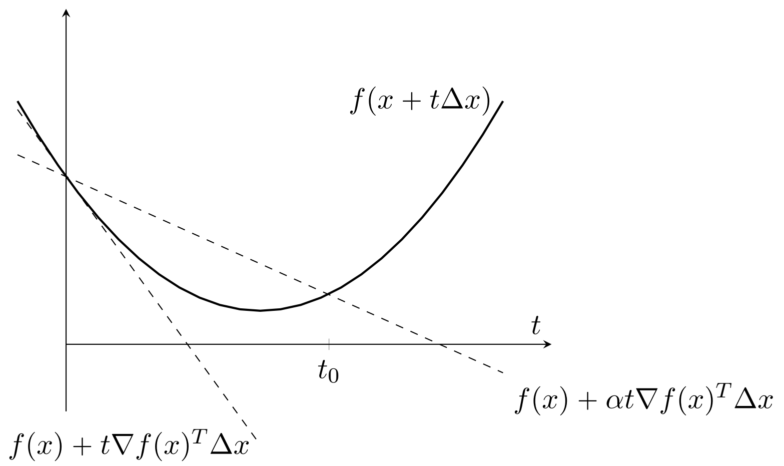
```
1: procedure Backtrack($\Delta x$ for $f(x)$, $0 < \alpha < 0.5$, $0 < \beta < 1$)
2:     $t = 1$
3:     while $\tilde{f}(t) > f(x) + \alpha t \nabla f(x)^T \Delta x$ do
4:         $t = \beta t$
5:     return $t$
```

Do not be confused by $f(x) + \alpha t \nabla f(x)^T \Delta x$, it is a linear function of t, since $f(x)$ and $\alpha \nabla f(x)^T \Delta x$ are constants. The intuition behind is this, we pick some $0 < \alpha < 0.5$ and degrade the slope of the lower bound $f(x) + t\nabla f(x)^T \Delta x$ by a factor of α, and then search for such t where $\tilde{f}(t) \leqslant f(x) + \alpha t \nabla f(x)^T \Delta x$. This situation is illustrated in the Figure **??**.

To see why this algorithm terminates consider that for a small enough $t > 0$ the function value $f(x + t\Delta x)$ is almost $f(x) + t\nabla f(x)^T \Delta x$ and since $\nabla f(x)^T \Delta x < 0$ we have the following

$$f(x + t\Delta x) \approx f(x) + t\nabla f(x)^T \Delta x < f(x) + \alpha t \nabla f(x)^T \Delta x.$$

Notice that the algorithm will terminate with $t = 1$ or $t \in (\beta t_0, t_0]$, where t_0 is a point such that $\tilde{f}(t_0) = f(x) + \alpha t \nabla f(x)^T \Delta x$.



$f(x + t\Delta x)$
t
t_0
$f(x) + \alpha t \nabla f(x)^T \Delta x$
$f(x) + t \nabla f(x)^T \Delta x$

Figure 3.1: Backtracking illustration.

3.1.1 Gradient descent

We are ready to instantiate our first descent method, namely the *gradient descent algorithm*. Intuition is really straightforward, by using the gradient at some point we obtain local information about the direction where the objective is going up the fastest, so we take the opposite direction.

Algorithm 4 Gradient descent method

```
1: procedure GRADIENT DESCENT($x \in \textbf{dom} f$)
2:     while True do
3:         $\Delta x = -\nabla f(x)$.
4:         if $\|\Delta x\|_2 < \eta$ then
5:             break
6:         Choose $t > 0$ by line search.
7:         $x = x + t\Delta x$.
8:     return $x$
```

Convergence analysis

We now show converge analysis for the case of the exact line search. Let $\tilde{f} : \mathbb{R} \to \mathbb{R}$ be a function of a step length defined as $\tilde{f}(t) = f(x - t\nabla f(x))$, such that $x - t\nabla f(x) \in S$. Setting $y = x - t\nabla f(x)$ to the inequality (3.4) we get

$$\tilde{f}(t) \leqslant f(x) - t\|\nabla f(x)\|_2^2 + \frac{Mt^2}{2}\|\nabla f(x)\|_2^2.$$

Since exact line search is used, we minimize over t both sides of the inequality. On the left hand side we obtain t_{exact} that minimizes $\tilde{f}$, and since the right hand side is a simple quadratic it is minimized by $t = 1/M$ and the minimum is attained at $f(x) - \frac{1}{2M}\|\nabla f(x)\|_2^2$. Thus, we have

$$\tilde{f}(t_{exact}) = f(x^{(k+1)}) \leqslant f(x^{(k)}) - \frac{1}{2M}\|\nabla f(x^{(k)})\|_2^2.$$

Now we subtract $p^\star$ and by using (3.3) in the form of $\|\nabla f(x^{(k)})\|_2^2 \geqslant 2m(f(x^{(k)}) - p^\star)$ we get

$$f(x^{(k+1)}) - p^\star \leqslant (1 - m/M)(f(x^{(k)}) - p^\star).$$

If we apply this inequality recursively, we get

$$f(x^{(k)}) - p^\star \leqslant (1 - m/M)^k (f(x^{(0)}) - p^\star). \tag{3.5}$$

Thus, we conclude that $f(x^{(k)})$ converges to $p^\star$ as $k \to \infty$ at least as fast as geometric sequence.

Remark 3.5 (Gradient descent drawback). Even though by (3.5) the gradient descent algorithm converges exponentially with the number of iterations, it depends on unknown constants m, M. The most non-obvious fact that this algorithm can work really bad. Recall that m and M were a lower and upper bounds on the eigenvalues of $\nabla^2 f(x)$ for all $x \in S$, so depending on the initial step $x^{(0)}$ if $0 < m/M \ll 1$ we might get horrible performance. ○

3.1.2 Newton's method

We now look at some improvements of the gradient method that will bring us to the Newton's method.

Remark 3.6 (Directional derivative). Consider the first order Taylor approximation of $f(x+v)$ at $v = 0$

$$f(x+v) \approx f(x) + \nabla f(x)^T v.$$

Here the term $\nabla f(x)^T v$ corresponds to the *directional derivative* of $f(x)$ in the direction of v. The directional derivative tells us the following, if we are at x how does the function changes if we move in the direction of v. ○

Thus, now we want to choose v in such a way that the directional derivative is as negative as possible. Obviously, we normalize $\nabla f(x)^T v$ by $\|v\|$, since we are only interested in the direction and but in the magnitude.

Definition 3.2 (Normalized steepest ascent direction)**.** Suppose $\|\cdot\|$ is an arbitrary norm on $\mathbb{R}^n$. A *normalized steepest descent direction* is defined as

$$\Delta x_{nsd} = \operatorname{argmin} \{\nabla f(x)^T u \mid \|u\| = 1\}.$$ ◎

The most unintuitive result is this, if we want to determine the direction where the function decreases the fastest, our result will depend on the norm.

Definition 3.3 (Dual norm)**.** Let $\|\cdot\|$ be a norm on $\mathbb{R}^n$. The function $\|\cdot\|_* : \mathbb{R}^n \to \mathbb{R}$ defined as

$$\|a\|_* = \sup\{|a^T x| \mid \|x\| \leqslant 1\} = \sup_{x \neq 0} \frac{a^T x}{\|x\|}$$

is a called the *dual norm* of $\|\cdot\|$. ◎

We can also interpret the dual norm as an operator norm of a^T, recall 1.64. Intuition behind the dual norms is the following, if we interpret a^T as a function $a^T(\cdot)$, the dual norm of a^T gives the largest function value of $a^T(\cdot)$ divided by the norm of the maximizer[5].

Definition 3.4 (Unnormalized steepest descent direction)**.** Suppose $\|\cdot\|$ is an arbitrary norm on $\mathbb{R}^n$ and $\|\cdot\|_*$ is the corresponding dual norm. The descent direction

$$\Delta x_{sd} = \|\nabla f(x)\|_* \Delta x_{nsd}$$

is called an *unnormalized steepest descent direction.* ◎

[5]In fact, $a^T(\cdot)$ is a linear functional 1.54 given by the Euclidean inner product on $\mathbb{R}^n$, i.e. $\langle a, x\rangle_{\mathbb{R}^n}$

Remark 3.7 (Unnormalized steepest descent direction property). By the definition of the dual norm we obtain

$$\nabla f(x)^T \Delta x_{nsd} = -\nabla f(x)^T \left(\operatorname*{argmax}_{\|u\|=1} \nabla f(x)^T u \right) = -\|\nabla f(x)\|_*$$

Now we have the following useful property of Δx_{sd}

$$\nabla f(x)^T \Delta x_{sd} = \|\nabla f(x)\|_* \nabla f(x)^T \Delta x_{nsd} = -\|\nabla f(x)\|_*^2. \qquad \circ$$

We now look at some concrete examples of Δx_{nsd} and Δx_{sd} for given norms.

Example 3.8 (Steepest descent in the Euclidean norm)**.** Notice if we use the Euclidean norm, then the steepest descent direction Δx_{sd} is in fact the negative gradient. To see this we first show that the Euclidean norm is self-dual

$$\|\nabla f(y)\|_* = \sup_{x \neq 0} \frac{|\nabla f(y)^T x|}{\|x\|_2}.$$

By the Cauchy–Schwarz inequality we get

$$\frac{|\nabla f(y)^T x|}{\|x\|_2} \leqslant \frac{\|\nabla f(y)\|_2 \, \|x\|_2}{\|x\|_2}.$$

If we take $x = c\nabla f(y)$ with $c \neq 0$ we get the equality, so the maximum value is $\|\nabla f(y)\|_2$, hence the Euclidean norm is self dual. Now applying $\Delta x_{nsd} = -\nabla f(x)$ to the Property 3.7, we get $-\|\nabla f(x)\|_2^2$, hence we conclude that the negative gradient gives the direction of the steepest descent in the Euclidean norm. $\bigcirc$

Example 3.9 (Steepest descent for quadratic norm)**.** Supposed we are given a quadratic norm $\|\cdot\|_{\mathbf{P}} : \mathbb{R}^n \to \mathbb{R}$ such that

$$\|a\|_{\mathbf{P}} = (a^T \mathbf{P} a)^{1/2} = \|\mathbf{P}^{1/2} a\|_2, \qquad \mathbf{P} \in \mathbb{S}_{++}^n.$$

The corresponding dual norm is $\|a\|_* = \|\mathbf{P}^{-1/2} a\|_2$. The normalized and unnormalized steepest descent directions are given by

$$\Delta x_{sd} = -\mathbf{P}^{-1} \nabla f(x), \qquad \Delta x_{nsd} = -\left(\nabla f(x)^T \mathbf{P}^{-1} \nabla f(x)\right)^{-1/2} \mathbf{P}^{-1} \nabla f(x).$$

The steepest descent direction can be viewed as the gradient descent direction in new coordinates. We define $y = \mathbf{P}^{1/2} x$ so $\|y\|_2 = \|x\|_{\mathbf{P}}$ and $g : \mathbb{R}^n \to \mathbb{R}$ such that

$$g(y) = f(\mathbf{P}^{-1/2} y) = f(x).$$

Now we apply the gradient descent method on $g(y)$

$$y^{(k+1)} = y^{(k)} - \eta \nabla g(y^{(k)}) = y^{(k)} - \eta \mathbf{P}^{-1/2} \nabla f(\mathbf{P}^{-1/2} y^{(k)}).$$

In terms of the original coordinates we obtain

$$\begin{aligned} \mathbf{P}^{-1/2} y^{(k+1)} &= \mathbf{P}^{-1/2} y^{(k)} - \eta \mathbf{P}^{-1/2} \mathbf{P}^{-1/2} \nabla f(\mathbf{P}^{-1/2} y^{(k)}) \\ x^{(k+1)} &= x^{(k)} - \eta \mathbf{P}^{-1} \nabla f(x^{(k)}). \end{aligned}$$

We conclude that $-\mathbf{P}^{-1} \nabla f(x^{(k)})$ corresponds to the steepest descent direction Δx_{sd} in quadratic norm $\|\cdot\|_{\mathbf{P}}$ Furthermore, we derive Δx_{nsd} by the definition of Δx_{sd}. $\bigcirc$

Now the question is how we choose $\|\cdot\|$ for the faster convergence. The intuitive approach is to choose $\mathbf{P}$ that approximately matches the level curves of the given function f. Another way to think about $\mathbf{P}$ is that we want to transform the coordinates in such a way that the level curves are almost rounded, then we apply the gradient decent method.

These ideas bring us to the Newton's method. The intuition is quite straightforward, we choose the norm based on our best guess of the curvature of f. Moreover, we change the norm after every step made, since the curvature also changes. Surely, one of the best guesses is given by the Hessian $\nabla^2 f(x)$. Thus, using 3.9 we define the Newton step.

Definition 3.5 (Newton step)**.** Suppose we are given an unconstrained problem 3.1. Let $x \in \mathbf{dom}\, f$, the vector Δx_{nt} defined as

$$\Delta x_{nt} = -\nabla^2 f(x)^{-1} \nabla f(x)$$

is called the *Newton step* for $f(x)$. ◎

Remark 3.10 (Descent direction of the Newton step). Since we assumed that f is strictly convex, the Hessian for every $x \in \mathbf{dom}\, f$ is positive definite, hence invertible and $\nabla^2 f(x)^{-1} \in \mathbb{S}^n_{++}$. Thus, for all $\nabla f(x) \neq 0, x \in S$ we have

$$\nabla f(x) \Delta x_{nt} = -\nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x) < 0.$$

Thus the Newton step is a valid descent direction. ○

Remark 3.11 (Interpretations of the Newton step).

- Consider the second order Taylor approximation of f at the point x denoted by $T_{2,x}(x+v)$

$$T_{2,x}(x+v) = f(x) + \nabla f(x)^T v + \frac{1}{2} v^T \nabla^2 f(x) v.$$

  It is a convex quadratic function of v, hence to find the minimizer we need to solve the system of linear equations

$$\nabla_v T_{2,x}(x+v) = \nabla f(x)^T v + \nabla^2 f(x) v = 0.$$

  The solution is given by $v = -\nabla^2 f(x)^{-1} \nabla f(x)$ that is exactly the Newton step.

- Another interpretation is the following. Suppose we are at x and we are searching for v such that $\nabla f(x+v) = 0$. This actually corresponds to finding $x^\star = x+v$ since $\nabla f(x^\star) = 0$. However, if $f : \mathbb{R}^n \to \mathbb{R}$ is a non-quadratic function equality $\nabla f(x+v) = 0$ leads to solving system of non-linear equations. To avoid this, we simply linearize $\nabla f(x+v)$ by the first order Taylor approximation at x and set the approximation to zero

$$\nabla f(x+v) \approx T_{1,x}(x+v) = \nabla f(x) + \nabla^2 f(x) v = 0.$$

  Thus, we have a system of linear equations and the solution is given by $v = \Delta x_{nt}$.

- The last interpretation is exactly derived from our motivation to adapt $\mathbf{P} \in \mathbb{S}^n_{++}$ to the level curves for better convergence in $\|\cdot\|_P$. Thus by Example 3.9 Δx_{nt} is the steepest descent direction at x induced by the Hessian $\nabla^2 f(x)$, in other words

$$\|u\|_{\nabla^2 f(x)} = (u^T \nabla^2 f(x) u)^{1/2}.$$ ○

Lemma 3.6 (Affine invariance)**.** Newton step is independent of a linear change of coordinates.

Proof. Suppose $\mathbf{A} \in \mathbb{R}^{n\times n}$ is non-singular. We define a function $g(y) = f(\mathbf{A}y)$. The gradient and the Hessian of g are

$$\nabla g(y) = \mathbf{A}^T \nabla f(x) \qquad \nabla^2 g(y) = \mathbf{A}^T \nabla^2 f(x)\mathbf{A}, \qquad x = \mathbf{A}y. \tag{3.6}$$

Thus, the Newton step for g at y is given by

$$\Delta y_{nt} = -\nabla^2 g(y)^{-1}\nabla g(y) = -(\mathbf{A}^T\nabla^2 f(\mathbf{A}y)\mathbf{A})^{-1}(\mathbf{A}^T\nabla f(\mathbf{A}y)) = \mathbf{A}^{-1}\Delta x_{nt}. \tag{3.7}$$

Here Δx_{nt} is the Newton step of f at $x = \mathbf{A}y$. Hence we see if $x = \mathbf{A}y$ then the Newton step for f at x is $\Delta x_{nt} = \mathbf{A}\Delta y_{nt}$, i.e. the same transformation holds. □

According to the lemma above, if we change coordinates the descent direction remains unchanged and we will make the same Newton steps in new coordinate system. Thus if $x^{(0)} = \mathbf{A}y^{(0)}$ then $x^{(k)} = \mathbf{A}y^{(k)}$.

Remark 3.12 (Gradient descendent and affine invariance). However, the gradient descent direction is not affine invariant. To see this, suppose $g(y) = f(\mathbf{A}y)$ with $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\det \mathbf{A} \neq 0$. The gradient of g is given by $\nabla g(y) = \mathbf{A}^T\nabla f(x)$ with $x = \mathbf{A}y$, so the gradient descent direction for $g(y)$ is given by $\Delta y = \mathbf{A}^T\Delta x$. Finally, if we choose $x = \mathbf{A}y$ then the gradient descendent direction for $f(x)$ is $\Delta x = \mathbf{A}^{T^{-1}}\Delta y$, hence linear transformation differs. ○

Definition 3.7 (Newton decrement)**.** Suppose we have an unconstrained optimization problem 3.1. The following quantity

$$\lambda(x) = (\nabla f(x)^T\nabla^2 f(x)^{-1}\nabla f(x))^{1/2}$$

is called the *Newton decrement* of f at a point x. ◎

Remark 3.13 (Newton decrement properties).

- Newton decrement gives an estimate of suboptimality of $f(x)$, for some $x \in S$. based on the second order Taylor approximation of f at x

$$\begin{aligned} f(x) - \inf_v T_{2,x}(v) =& f(x) - T_{2,x}(x + \Delta x_{nt}) \\ =& f(x) - f(x) - \nabla f(x)^T\Delta x_{nt} - \frac{1}{2}\Delta x_{nt}^T\nabla^2 f(x)\Delta x_{nt} \\ =& \frac{1}{2}\lambda(x)^2. \end{aligned}$$

- Notice that the Newton decrement can be viewed as the quadratic norm of Newton step defined by the Hessian

$$\lambda(x) = \left(\Delta x_{nt}^T\nabla^2 f(x)\Delta x_{nt}\right)^{1/2} = \|\Delta x_{nt}\|_{\nabla^2 f(x)}. \tag{3.8}$$

- We can also use $\lambda(x)$ in backtracking line search since

$$\nabla f(x)^T\Delta x_{nt} = -\lambda(x)^2.$$

- By (3.6) and 3.7 we have that for $g(y) = f(\mathbf{A}y)$, with $\det \mathbf{A} \neq 0$, the Newton decrement $\lambda_g(y)$ is equal to $\lambda_f(x)$ for f at $x = \mathbf{A}y$. Thus, Newton decrement is affine invariant since the stopping criterion for $g(y)$ and $f(x)$ is the same. ○

We now are ready to present *Newton's method*, namely *damped Newton's method*, since we can use step length $t < 1$.

Algorithm 5 Newton's method

```
1: procedure NEWTON UNCONSTRAINED($x \in \mathbf{dom}\, f,\ \epsilon > 0$)
2:     while True do
3:         $\Delta x_{nt} = -\nabla^2 f(x)^{-1} \nabla f(x)$
4:         $\lambda(x)^2 = \nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x)$
5:         if $\lambda(x)^2/2 \leqslant \epsilon$ then
6:             return $x$
7:         Choose $t > 0$ via backtracking
8:         $x = x + t\Delta x_{nt}$
9:     return $x$
```

Convergence analysis

We now present outline of convergence analysis for Newton's method and its conclusions, complete proof can be found in the book [1, p. 489-491].

Assume that f is twice continuously differentiable and strong convexity holds, hence by implications of strong convexity we have

$$\exists\, m, M > 0 : m\mathbf{I} \preccurlyeq \nabla^2 f(x) \preccurlyeq M\mathbf{I}, \qquad \forall x \in S.$$

Also suppose that $\nabla^2 f(\cdot)$ is Lipschitz continuous 1.63 on S with constant L, i.e.

$$\|\nabla^2 f(x) - \nabla^2 f(y)\|_2 \leqslant L\|x - y\|_2, \qquad x, y \in S.$$

Remark 3.14 (Operator norm). Here $\|\cdot\|_2 : \mathbb{R}^{n\times m} \to \mathbb{R}$ corresponds to the operator norm 1.64 of $X : \mathbb{R}^n \to \mathbb{R}^n$, with matrix $\mathbf{X} \in \mathbb{R}^{n\times n}$ induced by the Euclidean norm, i.e.

$$\|X\|_2 = \sup_{0 \neq u \in \mathbb{R}^n} \frac{\|\mathbf{X}u\|_2}{\|u\|_2} = \sigma_{\max}(\mathbf{X}).$$

As we see this this norm is given by the largest singular value of $\mathbf{X}$. Moreover, since $\nabla^2 f(x) \in \mathbb{S}^n_+$ this norm gives the largest eigenvalue. ○

Remark 3.15 (Comments on Lipschitz constant). Actually Lipschitz constant L says how fast the second derivative changes, so we can interpret L as a bound on the third derivative, which in case of multivariable functions is a third order tensor or a 3-linear form. Notice that if f is quadratic, the second order approximation is the function itself, so the Hessian is constant, hence we can take $L = 0$. Now intuition is this, if quadratic model was good enough the third derivative is small, so the Newtons method will perform really well. Thus, there are two ways to say that Newton's method works well

- The third derivative is small,
- The second derivative has small Lipschitz constant.

We rather interpret L in the second way, to avoid dealing with obscure tensors. ○

The result of convergence proof is that there exist η, γ such that $0 < \eta \leqslant m^2/L$ and $\gamma > 0$ and the following holds:

- If $\|\nabla f(x^{(k)})\|_2 \geqslant \eta$ then

$$f(x^{(k+1)}) - f(x^{(k)}) \leqslant -\gamma. \tag{3.9}$$

- If $\|\nabla f(x^{(k)})\|_2 < \eta$ then backtracking line search selects $t^{(k)} = 1$ and we have

$$\frac{L}{2m^2}\|\nabla f(x^{(k+1)})\|_2 \leqslant \left(\frac{L}{2m^2}\|\nabla f(x^{(k)})\|_2\right)^2. \tag{3.10}$$

We now analyse the second result. Assume $\|\nabla f(x^{(k)})\|_2 < \eta$ holds, then for every next iteration the condition remains satisfied. To see this, recall that $0 < \eta \leqslant m^2/L$ and consider the following

$$\frac{L}{2m^2}\|\nabla f(x^{(k+1)})\|_2 \leqslant \left(\frac{L}{2m^2}\|\nabla f(x^{(k)}\|_2\right)^2 < \left(\frac{L}{2m^2}\right)^2\eta^2 \leqslant \frac{L}{4m^2}\eta.$$

Applying the inequality $\ell > k$ times we get

$$\frac{L}{2m^2}\|\nabla f(x^{(\ell)})\|_2 \leqslant \left(\frac{L}{2m^2}\|\nabla f(x^{(k)})\|_2\right)^{2^{\ell-k}} \leqslant \left(\frac{1}{2}\right)^{2^{\ell-k}}.$$

Thus, using implications of strong convexity, namely (3.3) we obtain

$$f(x^{(l)}) - p^\star \leqslant \frac{L}{2m^2}\|\nabla f(x^{(l)})\|_2^2 \leqslant \frac{2m^3}{L^2}\left(\frac{1}{2}\right)^{2^{l-k+1}}. \tag{3.11}$$

As we see in the last inequality if $\|\nabla f(x^{(k)})\|_2 < \eta$ algorithm converges very fast. Informally, the number of correct digits doubles after each iteration. This type of convergence is called *quadratic convergence*.

Depending on the condition $\|\nabla f(x^{(k)})\|_2 < \eta$, the algorithm can be divided into two phases. First one is called *damped Newton phase* and the second is *quadratically convergent stage*.

We now count the number of iterations in each stage separately. Since in first phase value of the objective decreases at least by γ the total number of iterations is less than

$$\frac{f(x^{(0)}) - p^\star}{\gamma}.$$

To derive the bound in quadratic convergent phase we use (3.11) and take $\epsilon_0 = \frac{2m^3}{L^2}$

$$f(x^{(l)}) - p^\star \leqslant \epsilon_0 \left(\frac{1}{2}\right)^{2^{l-k+1}} \leqslant \epsilon.$$

Now consider the following

$$\begin{aligned} \epsilon_0 2^{-2^{l-k+1}} &\leqslant \epsilon \\ \log_2 \epsilon_0 - 2^{l-k+1} &\leqslant \log_2 \epsilon \\ \log_2 \log_2(\epsilon_0/\epsilon) &\leqslant l - k + 1. \end{aligned}$$

Hence to achieve precision at least ϵ we perform no more than $\log_2 \log_2(\epsilon_0/\epsilon)$ iterations. Thus, total number of iterations is given by

$$\frac{f(x^{(0)}) - p^\star}{\gamma} + \log_2 \log_2(\epsilon_0/\epsilon).$$

Remarkably, number of iterations for good solution and outstandingly precise one differers by the last term, that grows extremely slowly.

3.2 Equality constrained minimization

We are stepping towards general methods for solving convex problems, and now add simplest constrains, namely equality ones, these, according to the definition of the convex problem 1.3, are linear. In this section we modify the Newton's method to handle equality constrains.

Definition 3.8 (Equality constrained minimization)**.** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex twice continuously differentiable function, $\mathbf{A} \in \mathbb{R}^{p\times n}$ such that $\mathbf{rank}\,\mathbf{A} = p < n$ and $b \in \mathbb{R}^p$. An optimization problem given by

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } \mathbf{A}x = b \end{aligned}$$

with $p^\star = \inf_x\{f(x) \mid \mathbf{A}x = b\}$ finite and attained, is called the convex *equality constrained minimization problem.* ⊚

Using Lagrange duality and the KKT conditions 1.54 we formulate the optimality conditions.

Corollary 3.9 (Optimality conditions)**.** Assume we have an equality constrained minimization problem and $x^\star \in \mathbf{dom}\, f$. Point $x^\star$ is *optimal point* if and only if there exists $\nu^\star \in \mathbb{R}^p$ such that the following hold

$$\mathbf{A}x^\star = b, \qquad \nabla f(x^\star) + \mathbf{A}^T\nu^\star = 0.$$

These equations are called the *KKT equations.*

As we discussed earlier, assuming that the problem is not quadratic, $\nabla f(x) + \mathbf{A}^T\nu = 0$ is a system of nonlinear equations in x. As we will see, solving convex quadratic problems with equality constants is pure linear algebra.

Example 3.16 (Equality constrained convex quadratic optimization)**.** Suppose we are given a problem

$$\begin{aligned} &\text{minimize } f(x) = (1/2)x^T\mathbf{P}x + q^Tx + r, \\ &\text{subject to } \mathbf{A}x = b, \end{aligned} \tag{3.12}$$

with $\mathbf{P} \in \mathbb{S}^n_+,\ \mathbf{A} \in \mathbb{R}^{p\times n},\ b \in \mathbb{R}^p,\ q \in \mathbb{R}^n$, such that $\mathbf{rank}\,\mathbf{P} = p < n$. The optimality condition is given by

$$\mathbf{A}x^\star = b, \qquad \mathbf{P}x^\star + \mathbf{A}^T\nu^\star + q = 0.$$

We can rewrite the optimality condition using block matrix notation as the following

$$\begin{bmatrix} \mathbf{P} & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix} \begin{bmatrix} x^\star \\ \nu^\star \end{bmatrix} = \begin{bmatrix} -q \\ b \end{bmatrix}.$$

As we see the optimality condition is just a system of $n + p$ linear equations. This system is called the *KKT system* for equality constrained QP (3.12) with the *KKT matrix* of coefficients.

Solution of the KKT system depends on the non-singularity of KKT matrix. If the matrix is non-singular there exists exactly one pair $(x^\star, \nu^\star)$ solving the KKT system. If The KKT matrix is singular but the system is solvable then there exists infinitely many solutions $(x^\star, \nu^\star)$, otherwise the KKT system is unsolvable, meaning that the optimization problem is unbounded below or infeasible. ○

Lemma 3.10 (Non-singularity of the KKT system)**.** Suppose we are given the problem (3.12). The following condition for $0 \neq x \in \mathbb{R}^n$

$$\mathbf{A}x = b \implies x^T\mathbf{P}x > 0,$$

is equivalent to non-singularity of the KKT matrix.

Proof. We prove both implications by contradiction.

- $\Longleftarrow$ Suppose $\mathbf{A}x = 0,\ \mathbf{P}x = 0,\ x \neq 0$, then we obtain

$$\begin{bmatrix} \mathbf{P} & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} = 0,$$

  hence the KKT matrix is singular.

- $\Longrightarrow$ Suppose the KKT matrix is singular, i.e. for x, ν not both zero we obtain

$$\begin{bmatrix} \mathbf{P} & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix} \begin{bmatrix} x \\ \nu \end{bmatrix} = 0,$$

  Thus, $\mathbf{P}x + \mathbf{A}^T\nu = 0$ and $\mathbf{A}x = 0$. We multiply the first equality by x^T on the left and get $x^T\mathbf{P}x + x^T\mathbf{A}^T\nu = 0$. Applying $\mathbf{A}x = 0$ first equality reduces to $x^T\mathbf{P}x = 0$, if $x \neq 0$ we immediately get contradiction, if $x = 0$ then $\nu \neq 0$ and $\mathbf{A}\nu = 0$, i.e. $\mathbf{rank}\,\mathbf{A} < p$. □

3.2.1 Newton's method with equality constrains

This brings us to the extension of the Newton step and the Newton decrement for equality constrained minimization. In case of constrained minimization we need to preserve feasibility at each iteration, moreover, the initial point $x^{(0)}$ should also be feasible. This actually means that every Newton step must satisfy $\mathbf{A}\Delta x_{nt} = 0$, i.e. Δx_{nt} is an element of the null space of $\mathbf{A}$.

Definition 3.11 (Newton step for equality constrained optimization)**.** Suppose we are given an equality constrained problem 3.8. The Newton step Δx_{nt} for feasible x is characterized by solution in v of

$$\begin{bmatrix} \nabla^2 f(x) & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} -\nabla f(x) \\ 0 \end{bmatrix},$$

with dual variable w. ◎

Notice that in unconstrained case the system of linear equations reduces to $\nabla^2 f(x)v = -\nabla f(x)$. Thus, the Newton step for equality constrained optimization is an extension of the original Newton step. Moreover, the same interpretations as in unconstrained case hold.

Remark 3.17 (Interpretations of the Newton step)*.*

- As the example 3.12 suggests the Newton step Δx_{nt} solves an optimization problem with objective given by the second order Taylor approximation near x with variable v

$$\begin{aligned} &\text{minimize } T_{2,x}(x+v) = f(x) + \nabla f(x)^T v + (1/2)v^T\nabla^2 f(x)v \\ &\text{subject to } \mathbf{A}(x+v) = b. \end{aligned}$$

- The Newton step can be also viewed as the solution to linearized version of optimality conditions 3.9, taking $x^\star = x + \Delta x_{nt}$ and $\nu^\star = w$ we obtain

$$\nabla f(x) + \nabla^2 f(x)\Delta x_{nt} + \mathbf{A}^T w = 0, \qquad \mathbf{A}(x + \Delta x_{nt}) = b.$$

  Since $\mathbf{A}x = b$ we obtain exactly the same system of linear equations as in the definition 3.11

$$\nabla^2 f(x)\Delta x_{nt} + \mathbf{A}^T w = -\nabla f(x), \qquad \mathbf{A}\Delta x_{nt} = 0.$$

○

Definition 3.12 (Newton decrement for equality constrained optimization)**.** Assume that we are given an equality constrained problem 3.8. The Newton decrement is defined as the following

$$\lambda(x) = (\Delta x_{nt}^T \nabla^2 f(x) \Delta x_{nt})^{1/2}. \qquad ⊚$$

Notice that by the definition the Newton decrement in equality constrained case is equal to (3.8). However it *can not* be expressed as 3.7, since we are using extended version of the Newton step. The Newton decrement for constrained problems derives its properties from an unconstrained case.

Remark 3.18 (Properties of the Newton decrement)*.*

- The Newton decrement gives an estimation of suboptimality for $f(x)$ based on the second order approximation

  $$\begin{aligned} f(x) - \inf_v \{T_{2,x}(x+v) \mid \mathbf{A}(x+v) = b\} \\ &= f(x) - f(x) - \nabla f(x)^T v + \frac{1}{2} v^T \nabla^2 f(x) v \\ &= -\lambda(x)^2/2. \end{aligned}$$

  Since by the definition 3.11 we have $\Delta x_{nt}^T \nabla^2 f(x) \Delta x_{nt} = -\Delta x_{nt}^T \nabla f(x)$.

- We can also use the Newton decrement in the backtracking line search

  $$\nabla f(x)^T \Delta x_{nt} = -\lambda(x)^2. \qquad \circ$$

Lemma 3.13 (Affine invariance)**.** The Newton step and the Newton decrement for an equality constrained optimization problem 3.8 are affine invariant.

Proof. Suppose $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is invertible. We define function $g(y) = f(\mathbf{Q}y) = f(x),\ x = \mathbf{Q}y$. The gradient and the Hessian for g are

$$\nabla g(y) = \mathbf{Q}^T \nabla f(\mathbf{Q}y), \qquad \nabla^2 g(y) = \mathbf{Q}^T \nabla^2 f(\mathbf{Q}y)\mathbf{Q}.$$

We now formulate the following optimization problem

$$\begin{aligned} &\text{minimize } g(y) \\ &\text{subject to } \mathbf{AQ}y = b. \end{aligned} \tag{3.13}$$

To start with, we determine the Newton step Δx_{nt} corresponding to the original problem 3.8 by solving system of linear equations

$$\begin{bmatrix} \nabla^2 f(x) & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix} \begin{bmatrix} \Delta x_{nt} \\ w \end{bmatrix} = \begin{bmatrix} -\nabla f(x) \\ 0 \end{bmatrix}.$$

We reformulate this system as

$$\Delta x_{nt} = -\nabla^2 f(x)^{-1}(\mathbf{A}^T w + \nabla f(x)), \qquad \mathbf{A}\Delta x_{nt} = 0$$

Now we work out the Newton step Δy_{nt} for the optimization problem given by g (3.13)

$$\begin{bmatrix} \mathbf{Q}^T \nabla^2 f(\mathbf{Q}y)\mathbf{Q} & \mathbf{Q}^T\mathbf{A}^T \\ \mathbf{AQ} & 0 \end{bmatrix} \begin{bmatrix} \Delta y_{nt} \\ \bar{w} \end{bmatrix} = \begin{bmatrix} -\mathbf{Q}^T \nabla f(\mathbf{Q}y) \\ 0 \end{bmatrix}.$$

This system can be rewritten as

$$\Delta y_{nt} = -\mathbf{Q}^{-1}\nabla^2 f(\mathbf{Q}y)^{-1}(\mathbf{A}^T\bar{w} + \nabla f(\mathbf{Q}y)), \qquad \mathbf{AQ}\Delta y_{nt} = 0$$

Notice if $x = \mathbf{Q}y$ the system above takes the following form

$$\Delta y_{nt} = \mathbf{Q}^{-1}\Delta x_{nt}, \qquad \mathbf{A}\Delta x_{nt} = 0.$$

Thus, the the Newton steps for x and y are related by the same transformation and $\bar{w} = w$. Finally, using $\Delta y_{nt} = \mathbf{Q}^{-1}\Delta x_{nt}$ we see that the Newton decrement $\lambda_g(y)$ matches $\lambda_f(x)$ for $x = \mathbf{Q}y$. □

Now we present the Newton's method for equality constrained minimization, which is a modified version of the algorithm for unconstrained problems 5. However, here we use the extended Newton step and decrement. This method is usually called *feasible descent method*, since every $x^{(k+1)} = x^{(k)} + t\Delta x_{nt}$ is a feasible point and $f(x^{(k+1)}) < f(x^{(k)})$.

Algorithm 6 Newton's method for equality constrained problems

```
1: procedure NEWTON EQUALITY($x \in \mathbf{dom}\, f,\ \epsilon > 0$)
2:     while True do
3:         Compute $\Delta x_{nt}$ according to 3.11.
4:         $\lambda(x)^2 = \Delta x_{nt}\nabla^2 f(x)\Delta x_{nt}$
5:         if $\lambda(x)^2/2 \leqslant \epsilon$ then
6:             return $x$
7:         Choose $t > 0$ via backtracking
8:         $x = x + t\Delta x_{nt}$
9:     return $x$
```

Convergence analysis

To show that Newton's method for equality constrained optimization converges, we eliminate the equality constraints and propagate them to the objective

$$\underset{z}{\text{minimize}} \;\; f(\mathbf{F}z + \tilde{x}).$$

Here columns of $\mathbf{F} \in \mathbb{R}^{n\times(n-p)}$ span the null space of $\mathbf{A}$, i.e. $\mathbf{AF} = 0$, and $\mathbf{rank}\,\mathbf{F} = n - p$. The point $\tilde{x}$ is a particular solution of $\mathbf{A}x = b$.

In general, the proof remains the same. The relation between Newton's method applied on the eliminated problem and one that directly handles the equality constraints along with slightly changed convergence analysis can be found in the book [1, p. 528 - 531].

Here we only outline modified assumptions of the Newton's method for inequality constraints.

- The sublevel sets of f are closed, hence the initial sublevel set $S = \{x \in \mathbb{R}^n \mid x \in \mathbf{dom}\, f, f(x) \leqslant f(x^{(0)}), \mathbf{A}x = b\}$ with $x^{(0)} \in \mathbf{dom}\, f, \mathbf{A}x^{(0)} = b$ is also closed.
- For any $x, y \in S$ the Hessian is Lipschitz continuous on S, i.e. there exists a positive Lipschitz constant such that $\|\nabla^2 f(x) - \nabla^2 f(y)\|_2 \leqslant L\|x - y\|_2$.
- The inverse of the KKT matrix exists and is bounded on S, i.e.

$$\left\| \begin{bmatrix} \nabla^2 f(x) & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix}^{-1} \right\|_2 \leqslant K.$$

We also assume that the Hessian as a function of $x \in S$ is bounded, i.e. $\nabla^2 f(x) \preccurlyeq M\mathbf{I},\ M > 0$.

To understand the last assumption on KKT matrix recall operator norm 3.14, since the KKT matrix is symmetric it has real eigenvalues. Notice if there were no equality constraints this condition reduces to $\|\nabla^2 f(x)^{-1}\|_2 \leqslant K$ on S, so if $m = 1/K$ we have $\nabla^2 f(x) \succcurlyeq m\mathbf{I}$ from strong convexity assumptions (3.1). Here the norm of the inverse of the KKT matrix returns $1/|\lambda'|$, where $|\lambda'| \geqslant 1/K$ is the smallest eigenvalue of the KKT matrix in the absolute value, hence we want positive and negative eigenvalues be away from zero.

3.3 Inequality constrained minimization

We now are ready to look at the algorithm that handles general convex problems with inequalities. As previously we reformulate the optimization problem in a proper way to apply already shown algorithms for equality constrained and unconstrained minimization.

Definition 3.14 (Inequality constrained problem)**.** Let $f_0, \dots, f_m : \mathbb{R}^n \to \mathbb{R}$ be a convex twice continuously differentiable functions, $\mathbf{A} \in \mathbb{R}^{p\times n}$ such that $\mathbf{rank}\,\mathbf{A} = p < n$. An optimization problem given by

$$\begin{aligned} \text{minimize}\ & f_0(x) \\ \text{subject to}\ & f_i(x) \leqslant 0, \qquad i = 1, \dots, m \\ & \mathbf{A}x = b \end{aligned}$$

where $p^\star$ is a finite and attained optimal value and solution $x^\star$ exists. This problem is called a convex *inequality constrained minimization problem.* ◎

As we see QCQP, QP, LP are just special cases of inequality constrained problems.

Remark 3.19 (The KKT conditions). We also assume that the problem is strictly feasible, i.e. $\exists\, x \in \mathcal{D}$ such that $\mathbf{A}x = b$ and $f_i(x) < 0,\ i = 1, \dots, m$ so the Slater's qualification holds 1.52, hence the strong duality attained. By the characterization of optimal point 1.55 vector $x^\star$ is optimal if and only if there exist dual optimal $\lambda^\star \in \mathbb{R}^m,\ \nu^\star \in \mathbb{R}^p$ and the KKT conditions hold

$$\begin{aligned} \mathbf{A}x^\star &= b, \\ f_i(x^\star) &\leqslant 0, && i = 1, \dots, m. \\ \lambda^\star &\succcurlyeq 0, \\ \lambda^{\star T} f_i(x^\star) &= 0, && i = 1, \dots, m. \\ \nabla f_0(x^\star) + \textstyle\sum_{i=1}^m \lambda^\star f_i(x^\star) + \mathbf{A}^T \nu^\star &= 0. \end{aligned} \tag{3.14}$$

○

3.3.1 Barrier method

Now we reformulate the inequality constrained problem to one with equality constrains, although with non-differentiable objective. Further, we look how to properly approximate it. The original problem 3.14 can be reformulated as

$$\begin{aligned} \text{minimize}\ & f_0(x) + \textstyle\sum_{i=1}^m I_-(f_i(x)), \\ \text{subject to}\ & \mathbf{A}x = b. \end{aligned} \tag{3.15}$$

Here $I_- : \mathbb{R} \to \mathbb{R}$ is an irritation function for negative reals given by

$$I_-(x) = \begin{cases} 0, & x \leqslant 0, \\ \infty, & x > 0. \end{cases}$$

Since I_- is not differentiable we approximate it as the following

$$I_- \approx \widehat{I}_- = -(1/t)\log(-x), \qquad \mathbf{dom}\,\widehat{I}_- = \mathbb{R}_{++}.$$

Here parameter $t > 0$ determines the accuracy of the approximation, and as we will see later when $t \to \infty$ the approximation gets more accurate. The given approximation is valid by assumption of strict feasibility. Notice that the approximation of the indicator function is differentiable and closed, since every its sublevel set is closed. We also extend $\widehat{I}_-$ to take value of infinity when $x > 0$, so $\widehat{I}_-$ is convex non-decreasing function by the composition rule 1.33. We now reformulate 3.15 as the following

$$\begin{aligned} &\text{minimize } f_0(x) + \textstyle\sum_{i=1}^m -(1/t)\log(f_i(x)) \\ &\text{subject to } \mathbf{A}x = b. \end{aligned} \tag{3.16}$$

Definition 3.15 (Logarithmic barrier)**.** Suppose we have an inequality optimization problem 3.14. Let $\varphi : \mathbb{R}^n \to \mathbb{R}$ be a function with domain $\mathbf{dom}\,\varphi = \{x \in \mathbb{R}^n \mid f_i(x) < 0,\, i = 1, \dots, m\}$ defined as

$$\varphi(x) = -\sum_{i=1}^m \log(-f_i(x)).$$

Function φ is called the *logarithmic barrier* or simply the *log barrier.* ◎

For further purposes we work out the gradient and the hessian of the log barrier

$$\begin{aligned} \nabla\varphi(x) &= \sum_{i=1}^m \frac{1}{-f_i(x)}\nabla f_i(x) \\ \nabla^2\varphi(x) &= \sum_{i=1}^m \frac{1}{f_i(x)^2}\nabla f_i(x)\nabla f_i(x)^T + \sum_{i=1}^m \frac{1}{-f_i(x)}\nabla^2 f_i(x). \end{aligned}$$

Now we closely look at the approximated optimization problem (3.16)

$$\begin{aligned} &\text{minimize } tf_0(x) + \varphi(x), \\ &\text{subject to } \mathbf{A}x = b. \end{aligned} \tag{3.17}$$

Suppose the Newton's method is used for solving this problem, and for $\forall t > 0$ the solution $x^\star(t)$ is unique, later we will discuss these assumptions.

Definition 3.16 (Central path)**.** A set $\{x^\star(t) \in \mathbb{R}^n \mid t > 0\}$ is called the *central path* of the problem 3.14. Every $x^\star(t)$ is called the *central point.* ◎

Remark 3.20 (Characterization of central points). Every central point $x^\star(t)$ on the central path can be characterized as the following: $x^\star(t)$ is a central point if and only if it is strictly feasible and there exists $\bar{\nu} \in \mathbb{R}^p$ such that

$$\begin{aligned} 0 =& t\nabla f(x^\star(t)) + \nabla\varphi(x^\star(t)) + \mathbf{A}^T\bar{\nu} \\ =& \nabla f(x^\star(t)) + \sum_{i=1}^m \frac{1}{-tf_i(x^\star(t))}\nabla f_i(x^\star(t)) + (1/t)\mathbf{A}^T\bar{\nu}. \end{aligned}$$ ○

Remark 3.21 (Duality). Notice that by the characterization of central points we can work out the dual variables $\lambda^\star(t), \nu^\star(t)$ defined as

$$\nu^\star(t) = \bar{\nu}/t, \qquad \lambda_i^\star(t) = \frac{1}{-tf_i(x^\star(t))},\ i = 1, \dots, m.$$

Since $\lambda_i^\star(t) > 0$ and by 3.20 $x^\star(t)$ minimizes the Lagrangian at $\lambda^\star(t), \nu^\star(t)$

$$L(x, \lambda^\star(t), \nu^\star(t)) = f_0(x) + \sum_{i=1}^{m} \lambda_i^\star(t) f_i(x) + \nu^\star(t)^T(\mathbf{A}x - b)$$

points $\lambda^\star(t), \nu^\star(t)$ are dual feasible. Hence, the dual function at $\lambda^\star(t), \nu^\star(t)$ is finite and we have

$$\begin{aligned} g(\lambda^\star(t), \nu^\star(t)) &= f_0(x^\star(t)) + \sum_{i}^{m} \lambda_i^\star(t) f_i(x^\star(t)) + \nu^\star(t)^T \underbrace{(\mathbf{A}x^\star(t) - b)}_{=0} \\ &= f_0(x^\star(t)) - m/t. \end{aligned}$$

By definition $g(\lambda^\star(t), \nu^\star(t))$ determines a lower bound on $f_0(x^\star(t))$, so we can determine the suboptimality of $f_0(x^\star(t))$ as

$$f_0(x^\star(t)) - p^\star \leqslant m/t.$$

The inequality above actually proves the intuition, that when $t \to \infty$ central point $x^\star(t)$ converges to $x^\star$ of 3.14. ○

Remark 3.22 (Interpretation). The characterization 3.20 can be interpret in terms of the modified KKT conditions. Point $x^\star(t)$ is optimal if and only if there exist λ, ν such that

$$\begin{aligned} &\mathbf{A}x^\star(t) = b,\ f_i(x^\star(t)) \leqslant 0, \qquad i = 1, \dots, m. \\ &\lambda \succcurlyeq 0 \\ &\lambda^T f_i(x^\star(t)) = 1/t, \qquad i = 1, \dots, m. \\ &\nabla f_0(x^\star(t)) + \textstyle\sum_{i=1}^{m} \lambda f_i(x^\star) + \mathbf{A}^T\nu = 0 \end{aligned}$$

As we see these conditions slightly differers from conditions given in 3.14, in particular we replaced complementary condition $\lambda_i f_i(x) = 0$ by $\lambda^\star f_i(x^\star(t)) = 1/t$. Thus as t increases we approximate the original KKT conditions. ○

Now we are ready to look at the algorithm handling inequality constrained problems, that is actually a simple extension of the Newton's method. The method is called *barrier method* or SUMT, sequential unconstrained minimization technique, since Newton's method for equality constrains corresponds to applying original Newton's method on eliminated problem.

Algorithm 7 Barrier method

```
1: procedure BARRIER(strictly feasible $x$, $t^{(0)} > 0$, $\mu > 1$, $\epsilon > 0$)
2:     $t = t^{(0)}$
3:     while True do
4:         Centring step: $x^\star(t) = \text{argmin } \{tf_0(x) + \varphi(x) \mid \mathbf{A}x = b\}$
5:         $x = x^\star(t)$
6:         if $m/t < \epsilon$ then
7:             return $x$
8:         $t = \mu t$
```

As we see the algorithm produces sequence of $x^\star(t)$ until the duality gap is not small enough, i.e. $m/t < \epsilon$. The algorithm can also return $(\lambda^\star(t), \nu^\star(t))$ as a certificate proving the optimality of x. As was assumed earlier, in the centring step we use Newton's method, however any other method handling equality constrains can be used. We usually call centring step *outer iteration* and iterations of the Newton's method *inner iterations*.

Remark 3.23 (Choice of μ and t_0). As we wil see in convergence analysis choice of μ determines the trade-off between the number of inner and outer iterations, choosing μ large there will be fewer outer, however number of inner steps increases. The intuition behind this is the following, when μ is small enough previous x already approximates well enough, so we produce a small number of iterations in probably in quadratic convergent stage. The usual choice of μ is between 10 and 50. In addition, choice of t_0 determines the same trade-of since it gives how "far" the next centring step $x^{(1)}$ will be. ○

Remark 3.24 (Feasibility). Notice that the barrier method requires strictly feasible starting point x. To find one or determine that such a point does not exist we formulate the following optimization problem

$$\begin{aligned} \underset{x,s}{\text{minimize}}\ & s \\ \text{subject to}\ & f_i(x) \leqslant s, \qquad i = 1, \dots, m \\ & \mathbf{A}x = b. \end{aligned}$$

Here decision variables are $x \in \mathbb{R}^n$, $s \in \mathbb{R}$. Fortunately, this problem is always feasible, we just pick any $x^{(0)} \in \mathbf{dom}\, f_1 \cap \dots \cap \mathbf{dom}\, f_m$ as a starting point and determine $s^{(0)} = \max_{i=1,\dots,m} f_i(x^{(0)}) + \xi$ where $\xi > 0$. Now barrier method can be applied on feasibility problem.

We terminate the barrier method when $s^{(k)} < 0$ and use $x^{(k)}$ as a starting point for the original problem. If we finished with $s^{(k)} > 0$ this means that there is no such x that for all f_i satisfies $f_i(x) \leqslant 0$ and the problem is infeasible. However if $s^{(k)} = 0$ returned the original problem is feasible but not strictly feasible, hence we can not apply the barrier method. ○

Convergence analysis

Since whole barrier method is based on the Newton's method for equality constrains, we do need a separate proof. Suppose that $tf_0 + \varphi$ can be minimized by Newton's method fot $t = t^{(0)}, \mu t^{(0)}, \mu^2 t^{(0)}, \dots$. We know that the duality gap will be exactly $m/\mu^{(k)} t^{(0)}$ after k iterations so we can determine the precise number of steps

$$\begin{aligned} m &\leqslant \epsilon \mu^k t^{(0)} \\ \log_\mu m &\leqslant k + \log_\mu \epsilon t^{(0)} \\ k &\geqslant \frac{\log(m/(\epsilon t^{(0)}))}{\log \mu} \\ k &= \left\lceil \frac{\log(m/(\epsilon t^{(0)}))}{\log \mu} \right\rceil \end{aligned}$$

plus initial centering step in the feasibility phase. Here we exactly see the trade-of determined by μ.

Thus, we conclude that the problem is solvable if $tf_0 + \varphi$ satisfies the conditions discussed in the convergence analysis of Newton's method for equality constrained minimization. To ensure strong convexity we can add new $(m+1)$th constraint $\|x\|_2^2 \leqslant R^2$, this constraint adds a new term $-\log(R^2 - x^T x) = f_{m+1}(x)$ to $\varphi(x)$. Now we define $\tilde{\varphi}(x) = -\sum_{i=1}^{m+1} \log(-f_i(x))$ and show that $\nabla^2(f_0(x) + \tilde{\varphi}(x))$ is positive definite. The gradient and the Hessian of $f_{m+1}(x)$ are

$$\begin{aligned} \nabla f_{m+1}(x) &= \frac{2}{R^2 - x^T x} x \\ \nabla^2 f_{m+1}(x) &= \frac{2}{(R^2 - x^T x)^2}((R^2 - x^T x)\mathbf{I} + 2xx^T). \end{aligned}$$

Now consider the following

$$\begin{aligned}
\nabla^2(f_0(x) + \tilde{\varphi}(x)) &= \nabla^2(f_0(x) + \varphi(x)) + \frac{2}{(R^2 - x^T x)}\mathbf{I} + \frac{2}{(R^4 - x^T x)^2} x x^T \\
&\succcurlyeq \nabla^2(f_0(x) + \varphi(x)) + \frac{2}{R^2}\mathbf{I} \\
&\succcurlyeq \frac{2}{R^2}\mathbf{I}.
\end{aligned}$$

Thus, $f_0 + \tilde{\varphi}$ is strongly convex.

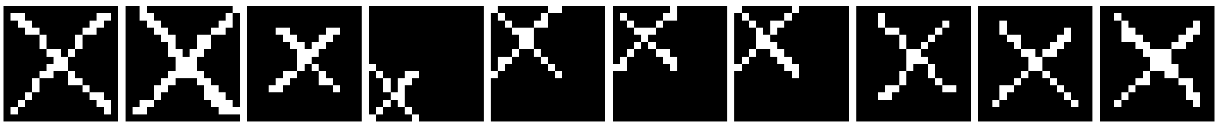
Chapter 4

Applications

In this chapter we focus on programming part of mathematical optimization. Firstly, we recall the Kernel SVM learning rule and describe a problem of image recognition. Then we explore existing and benchmark solvers in Julia and Python that can handle optimization problems. Finally, we show how kernel SVM learning rule can be optimized with a selected solver.

4.1 Problem specification

A nice illustrative example of kernel SVM usage is image recognition, in particular the task is to determine wether an image contains letter "X" or not. Some sampling from the dataset is shown in the Figure 4.1.


(a) Images with letters "X".

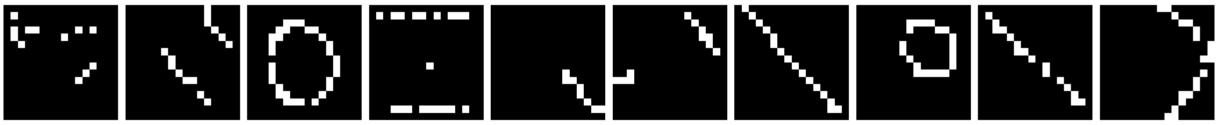

(b) Other images.

Figure 4.1: Dataset illustration.

Every image is represented by 16×16 binary matrix. Since the dataset contains only 168 images, we enlarge it adding rotated versions of original images, i.e we rotate images by 90, 180, 270 degrees. Then for every image we add its transpose as well.Hence in total we obtain about 1500 images.

Since determining wether image contains "X" is a binary classification problem, we use kernel SVM learning rule 2.4. For this example $\mathcal{X} \subseteq \mathbb{R}^{16\times 16}$ is an input space of images, $\mathcal{Y} = \pm 1$ is the output space of labels and $((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$ is a training set of m points. For the kernel function $k(\cdot, \cdot) : \mathbb{R}^{16\times 16} \times \mathbb{R}^{16\times 16} \to \mathbb{R}$ we use the Gaussian kernel (1.7) defined as $k(x, y) = \exp\left(-\frac{\|x-y\|_2^2}{2\sigma^2}\right)$. The task is to find the dual variable $\lambda^\star$ that solves the optimization

```julia
using JuMP
function solve_opt(K, y; optimizer, C=22)
    dim = length(y)
    m = Model(with_optimizer(optimizer))
    @variable(m, 0 <= L[1:dim] <= C)
    @variable(m, t)
    @objective(m, Min, t)
    @constraint(m, con1,
                sum(L[i]*L[j]*y[i]*y[j]*K[i][j] for j in 1:dim for i in 1:dim)
                    <= sum(var for var in L) + t)
    @constraint(m, con2, sum(y[i]*L[i] for i in 1:dim) == 0)
    optimize!(m)

    return value.(L)
end
```

Listing 1: Function for solving optimization problems for different optimizers.

problem

$$\begin{aligned}
\underset{\lambda \in \mathbb{R}^m}{\text{minimize}} \quad & \sum_{i,j=1}^{m} \lambda_i \lambda_j y_i y_j k(x_i, x_j) - 1^T \lambda \\
\text{subject to} \quad & 0 \leqslant \lambda_i \leqslant C, \qquad i = 1, \ldots, m, \\
& \lambda^T y = 0.
\end{aligned}$$

Recall that, the constant $C > 0$ gives the trade off between higher margin and number of missclassifications, since higher margin leads to more missclassifications. Thus, in our final model we have two free parameters C and σ, in the example section we will also establish a good enough choice for them by cross validation.

4.2 Julia

In Julia programming language we use a nice modeling tool, namely JuMP [13]m for mathematical optimization that provides a simple API for solvers usage. In the following code Listing 1 is shown how to use JuMP for solving optimization problems. This function takes as an input a kernel matrix $\mathbf{K} \in \mathbb{R}^{m \times m}$, labels $y \in \mathbb{R}^m$, solver or optimizer and a hyperparameter $C > 0$, for now we stick with $C = 22$. Firstly, we call the constructor for our model `m` by `Model()` with the given optimizer. Here the macro `@variable` determines the decision variable λ, t[6]. Notice that bounds on the decision variables can be directly handled in macro `@variable`. We use macros `@objective` and `@constraint` to define objective and constraint functions. When the problem is formulated we simply call `optimize!(m)` and return the solution $\lambda^{\star}$[7].

[6]We introduced a slack variable t for technical reasons, since solvers do not directly support QP.
[7]If the problem is feasible.

```julia
using ECOS
using Gurobi
using Mosek
using MathOptInterfaceMosek

l = [ECOS.Optimizer, Gurobi.Optimizer, MosekOptimizer]
for solver in l
    solve_opt(K, Y, optimizer = solver)
end
```

Listing 2: Solvers usage.

In the Julia section we focus on the following solvers

- ECOS [14],
- Gurobi [15],
- Mosek [16].

4.2.1 Solvers

The following code Listing 2 illustrates how to use solvers with function `solve_opt`.

ECOS ECOS is an embedded conic solver for second order cone programs, these can be viewed as generalization of QCQP. Even though it is a distributed under GPL as we will see ECOS shows competitive results comparable to commercial solvers. ECOS implements a sophisticated version of the barrier method for second order cone problems.

Gurobi Gurobi is a commercial solver, however it is available for academic purposes. This optimizer handles plenty of problems, including non-convex ones like integer linear programming and integer second order cone problems. Remarkably, Gurobi also implements barrier method for second order cone programs.

Mosek Mosek is another commercial tool available under academic licence. Comparing with Gurobi, Mosek allows to solve more general convex problems, namely conic ones. In conic optimization we have a convex objective function and the constrains are given by some convex cone, like positive semidefinite cone 1.10. Additionally, Mosek can solve integer versions of linear and second order cone programs. As other solvers, Mosek implements barrier methods.

4.2.2 Benchmarks

For benchmarking we use Julia package called `BenchmarkTools.jl`, it provides intuitive interface for carrying out measurements. Usage of this package is illustrated in the code Listing 3. To estimate time and memory usage we simply use macro `@benchmarkable` for our function. Then we set benchmark variables and run our experiments. Setting `seconds` to 1800 means limits the benchmark time to 30 minutes. In this benchmark we evaluate every model 12 times 2 times per each sample to get good enough overview of the solvers performance.

```julia
using BenchmarkTools
b = @benchmarkable solve_opt(K, Y, optimizer = Gurobi.Optimizer)
run(b, samples = 6, evals = 2, seconds = 1800)
```

Listing 3: Benchmarking.

Here we compare the solvers' performance using the whole dataset[8]. Moreover, we slightly extend it by moving around with images, i.e. every image is replaced with its two horizontally shifted versions by one column. These manipulation allow us to get about 3000 of images, and that is fairly enough for our benchmarks.

m	1500 images			2000 images			2500 images		
solvers	ECOS	Gurobi	Mosek	ECOS	Gurobi	Mosek	ECOS	Gurobi	Mosek
memory	6.62	5.85	6.29	11.68	10.38	11.10	18.35	16.24	17.39
min. time	31.125	18.004	15.749	113.89	44.145	35.611	239.76	90.531	72.696
mean time	33.211	18.763	16.305	117.16	45.360	36.406	252.71	94.004	74.475
max. time	35.498	19.493	18.050	122.16	47.149	37.267	282.59	98.024	77.011

Table 4.1: Benchmarks of the solvers performance, memory is measured in gibibytes and time is measured in seconds.

The results of benchmarks are given in the Table 4.1.

Obviously, commercial solvers demonstrate better performance, but as we see open source ECOS solver shows satisfying computer time results for not large problems. Notice that computational time doubles every 500 images, so problem with about 4000 or 5000 images can be solved in less than an hour with commercial solvers. Moreover, even not so large problems with only 2000 images require more than ten gibibytes of memory.

This benchmark shows that the Mosek has the best time performance out of the selected solvers, however memory usage is almost the same for all optimizers.

4.3 Python

For Python we also use a modeling language, namely CVXPY [17], [18]. This modeling language has similar interface as JuMP, moreover it implicitly supports matrix operation for problem formulation. Another advantage of CVXPY is that it is implicitly distributed with open source solvers. As earlier we use one generic function `solve_opt` for testing solvers performance. The code for this function is given in the Listing 4. We import package `cvxopt` and use classes `Variable`, `Problem` and `Minimize` to formulate an optimization problem. Notice that constraints are simply stored in the list. As we mention CVXPY comes with implicit solvers, in particular

- OQCP [19],
- SCS [20] [21],
- ECOS,

thus we use them for our benchmarking.

[8]Since in this we are only carrying out the benchmarks and not searching for the best model.

```python
import cvxpy as cp
@benchmark(12)
def solve_opt(K, y, optimizer=None, C=22):
    dim = len(y)
    x = cp.Variable(dim)
    obj = cp.Minimize(cp.quad_form(cp.multiply(y,x),K) - sum(x))
    constraints = [x >= 0, x <= C, sum([x[i]*y[i] for i in range(len(y))]) == 0]
    probl = cp.Problem(obj, constraints)
    probl.solve(optimizer)
    return  x.value
```

Listing 4: Python `solve_opt` function.

```python
X,Y = perturb(*preprocessing())
K = compute_kernel_matrix(X)
solver_list = [cp.OSQP, cp.ECOS, cp.SCS]
for s in solver_list:
    solve_opt(K,Y, optimizer=s)
```

Listing 5: Passing solvers.

4.3.1 Solvers

All solvers that we use in Python are open source and as we will see some of them also show even better results than solvers we used in Julia. Although, these optimizers are not written in Python they provide Python interface, hence ECOS is exactly the same solver described earlier, but now we communicate with it through Python API. Since OQCP, SCS, ECOS are directly installed in CVXPY, there is no need to import other libraries and we only pass different parameter to `solve_opt`, it is illustrated in the Listing 5.

OQCP OQCP is a solver for QP problems, hence the inequality constraints must be affine[9]. This solver implements alternating direction method of multipliers (ADMM), that belongs to the family of proximal algorithms that can directly handle non-smooth functions.

SCS Split conic solver or SCS is another optimizer that implements ADMM algorithm for conic problems. As Mosek it can be applied on positive semidefinite and second order cones.

4.3.2 Benchmarks

For benchmarking we implement a simple decorator function, the implementation can be viewed in the Listing 6. To display statistics we use package `pandas` and for time measurements `time` is used. Thus to keep our Python benchmarks consistent with Julia ones we perform the same preprocessing routine. We now run every optimizer on 1500, 2000, 2500 images 12 times. Here we only measure time performance, but not the memory usage.

[9]Notice that it is exactly our case.

```python
import functools
import time
import pandas as pd

def benchmark(n=2):
    def timer(func):
        @functools.wraps(func)
        def wrapper_timer(*args, **kwargs):
            l = []
            value = None
            for x in range(n):
                start_time = time.perf_counter()
                value = func(*args, **kwargs)
                end_time = time.perf_counter()
                l.append (end_time - start_time)
            print()
            print('Stats:')
            s = pd.Series(l)
            print(s.describe())
            return value
        return wrapper_timer
    return timer
```

Listing 6: Decorator function for carrying out benchmarks.

m	1500 images			2000 images			2500 images		
solvers	OQCP	ECOS	SCS	OQCP	ECOS	SCS	OQCP	ECOS	SCS
min. time	6.789	137.96	15.749	6.789	137.96	134.29	24.694	639.17	347.19
mean time	7.249	142.94	16.305	7.249	142.94	145.33	24.88	671.67	360.39
max. time	8.313	150.45	18.050	8.313	150.45	178.02	25.308	712.38	381.20

Table 4.2: Python solvers benchmark, time is measured in seconds.

As we see in the Table 4.2, ADMM algorithm implemented in OQCP works really well even comparing to commercial solvers with Julia interface. One reason might be that OQCP is a specific solver for convex quadratic problems with affine constrains, whereas other solvers handle more general optimization tasks. However, in contrast to Julia version ECOS solver for Python shows way worse results.

	OSQP	
m	3000	3500
min. time	44.07	67.144
mean time	47.168	70.565
max. time	54.40	75.168

Table 4.3: OQCP benchmark, time is measured in seconds.

Let us now test OQCP on larger sets of images. Thus as is demonstrated in the Table 4.3

```julia
function train_test_split(X, Y, p = 1301)
    joined = shuffle([(X[i], Y[i]) for i in 1:length(X)])
    X_train = [joined[i][1] for i in 1:length(joined) if i < p]
    Y_train = [joined[i][2] for i in 1:length(joined) if i < p]
    X_test = [joined[i][1] for i in p:length(joined)]
    Y_test = [joined[i][2] for i in p:length(joined)]
    return X_train, Y_train, X_test, Y_test
end
```

Listing 7: Julia `train_test_split` function.

OQCP applied on image recognition works unbelievably well and definitely beats commercial solvers discussed in Julia section. Unfortunately, this solver can be applied only on specific optimization problems, namely on QP.

4.4 Example

In this section we show how to use optimization solvers in problems of knowledge engineering, in particular optimizing kernel SVM learning rule for image recognition. For this example we stay with Julia and use Mosek solver.

Firstly, we split the original dataset with 1477 unique images, into training and testing data. Then we use five fold cross validation to measure the cross validation error on the training set, this allows us to choose suitable hyperparameters σ and C for the final model. When the choice of hyperparameters is made we test the final model on the testing data.

To divide the data into training and testing set we use custom function `train_test_plit` that is presented in the Listing 7. Here parameter `p` specifies the number of training points, the remaining data points form the testing data.

Next from the training data we create five cross validation sets by calling custom function `five_fold_cross_validation`. We perform cross validation to find good enough hyperparameters σ, C. This is done by making five validation sets S_i', $i = 1, \ldots, 5$ from the training set, where for every S_i' four fifths of it are used to train the model with fixed σ, C and the remaining one fifth is used determine the error ε_i. Then we compute the cross validation error as $(1/5)\sum_i^5 \varepsilon_i$. Obviously, we select σ, C with the lowest cross validation error. Code for `five_fold_cross_validation` is shown in the Listing 8. To estimate the error we simply count the number of missclassifications for training or validation set, and divide it by the number of training points.

We now show the code Listing 9 for optimization kernel SVM learning rule. Here the function `preprocessing` corresponds to reading the images from the text file. After we established σ, C with the lowest cross validation error, we use these hyperparameters to train the model on the whole training set, and afterwards display the results.

For this example the minimum cross validation error 0.0646 corresponds to $\sigma = 8.5$, $C = 27$. As the result, we have 0.0508 for the testing error, this leads to 9 out of 177 missclassifications. The missclassified images are illustrated in the Figure 4.2.

As we see kernel SVM learning rule does really good job in image classification, moreover it can be easily implemented using JuMP package.

```julia
function five_fold_cross_validation(X,Y)
    n=260
    xt = reshape(X, (n, div(length(X), n)))
    xl = [xt[:,i] for i in 1:div(length(X), n) ]
    xr = [(x,collect(Iterators.flatten([u for u in xl if x != u]))) for x in xl]

    yt = reshape(Y, (n, div(length(Y), n)))
    yl = [yt[:,i] for i in 1:div(length(Y), n) ]
    yr = [(x,collect(Iterators.flatten([u for u in yl if x != u]))) for x in yl]
    res = [(xr[i][2], yr[i][2],xr[i][1], yr[i][1]) for i in 1:length(yr)]
    return res
end
```

Listing 8: Julia `five_fold_cross_validation` function.

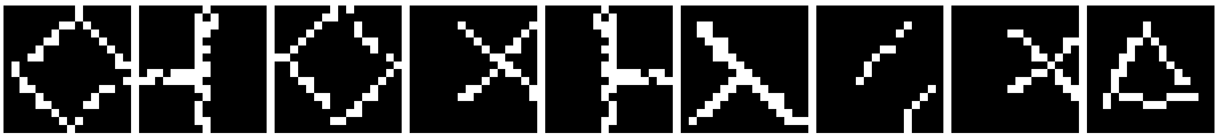


Figure 4.2: Missclassified images.

```
X, Y = preprocessing();
X_train, Y_train, X_test, Y_test = train_test_split(X, Y)
validation = ten_fold_cross_validation(X_train, Y_train)
sigmas = [
    6, 6, 6,
    7.5, 7.5, 7.5, 7.5,  7.5,
    8.5, 8.5, 8.5, 8.5, 8.5,
    10, 10, 10, 10, 10
]
Cs = [
    17, 27, 34,
    17, 27, 34, 40, 48,
    17, 27, 34, 40, 48,
    25, 34, 40, 44, 55,
]
hyppar = [(sigmas[i], Cs[i]) for i in 1:length(sigmas)]


result = []
for (sigma, c) in hyppar
    validation_errors = []
    for (X_t, Y_t, X_val, Y_val) in validation
        K = compute_kernel_matrix(X_t, sigma)
        lambda = solve_opt(K, Y_t, optimizer=MosekOptimizer, C=c)
        h = compute_classifier(X_t, Y_t, c, sigma, lambda)
        push!(validation_errors, error_estimate(h, X_val, Y_val)[1])
    end
    push!(result, (sigma, c, sum(err for err in validation_errors)/5))
end

min_err_sigma = 0
min_err_C = 0
min_err = result[1][end]
for (sigma, c, err) in result
    if min_err > err
        min_err_sigma = sigma
        min_err_C = c
        min_err = err
    end
end

K = compute_kernel_matrix(X_train, min_err_sigma)
lambda = solve_opt(K, Y_train, optimizer=MosekOptimizer, C=min_err_C)
h = compute_classifier(X_train, Y_train, min_err_C, min_err_sigma, lambda)
missclassified_images_display(error_estimate(h, X_test, Y_test)...,
                                X_test, min_err_C, min_err_sigma)
```

Listing 9: Kernel SVM learning rule optimized.

Conclusion

The main goal was to make a review of theory and methods of mathematical optimization and apply them on problems belonging to the realm of knowledge engineering.

1. There was introduced theoretical background of mathematical optimization, based on its convex part and we showed how to model plenty of problems using convex techniques. We studied the importance of convex optimization and its guarantees for the optimal results.
2. Then we looked at various problems in the field of knowledge engineering. Using already developed optimization framework we expressed these problems in terms of mathematical optimization, so they can be efficiently solved.
3. Next we studied modern methods for solving optimization problems. As we saw, these methods not only solve the optimization problems but can also return the certificate of the optimality.
4. Finally, we explored various software tools that can handle optimization problems using Python and Julia programming languages. We tested not only open source solvers, but also commercial ones. As it was shown, for specific problems open source tools can give competitive results comparing to commercial solvers.

Bibliography

[1] Boyd, S.; Vandenberghe, L. *Convex Optimization.* New York, NY, USA: Cambridge University Press, 2004, ISBN 0521833787.

[2] Dino Sejdinovic, A. G. What is an RKHS? 2014. Available from: `http://www.stats.ox.ac.uk/~sejdinov/teaching/atml14/Theory_2014.pdf`

[3] Académie des sciences (France). *Mémoires de l'Académie des sciences de l'Institut de France.* Gauthier-Villars, Firmin-Didot, 1827. Available from: `https://catalogue.bnf.fr/ark:/12148/cb343783130`

[4] Kreinovich, V.; Kearfott, R. B. Beyond Convex? Global Optimization is Feasible Only for Convex Objective Functions: A Theorem. *J. of Global Optimization*, volume 33, no. 4, Dec. 2005: pp. 617–624, ISSN 0925-5001, doi:10.1007/s10898-004-2120-1. Available from: `http://dx.doi.org/10.1007/s10898-004-2120-1`

[5] Li, L. *Selected Applications of Convex Optimization.* Springer-Verlag Berlin Heidelberg, 2015, ISBN 978-3-662-46355-0, doi:10.1007/978-3-662-46356-7.

[6] Rudin, W. *Real and Complex Analysis, 3rd Ed.* New York, NY, USA: McGraw-Hill, Inc., 1987, ISBN 0070542341.

[7] Mohri, M.; Rostamizadeh, A.; et al. *Foundations of Machine Learning.* The MIT Press, 2012, ISBN 026201825X, 9780262018258.

[8] Kalvoda, T. Nonlinear classification: Support Vector Machines. 2013. Available from: `https://github.com/kalvotom/cow13`

[9] Hastie, T.; Tibshirani, R.; et al. *The Elements of Statistical Learning.* Springer Series in Statistics, New York, NY, USA: Springer New York Inc., 2001.

[10] Shalev-Shwartz, S.; Ben-David, S. *Understanding Machine Learning: From Theory to Algorithms.* New York, NY, USA: Cambridge University Press, 2014, ISBN 1107057132, 9781107057135.

[11] Auer, P.; Herbster, M.; et al. Exponentially many local minima for single neurons. In *Advances in Neural Information Processing Systems 8*, edited by D. S. Touretzky; M. C. Mozer; M. E. Hasselmo, MIT Press, 1996, pp. 316–322. Available from: `http://papers.nips.cc/paper/1028-exponentially-many-local-minima-for-single-neurons.pdf`

[12] Aloise, D.; Deshpande, A.; et al. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, volume 75, no. 2, May 2009: pp. 245–248, ISSN 1573-0565, doi:10.1007/s10994-009-5103-0. Available from: `https://doi.org/10.1007/s10994-009-5103-0`

[13] Dunning, I.; Huchette, J.; et al. JuMP: A Modeling Language for Mathematical Optimization. *SIAM Review*, volume 59, no. 2, 2017: pp. 295–320, doi:10.1137/15M1020575.

[14] Domahidi, A.; Chu, E.; et al. ECOS: An SOCP solver for embedded systems. In *European Control Conference (ECC)*, 2013, pp. 3071–3076.

[15] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual. 2018. Available from: `http://www.gurobi.com`

[16] MOSEK ApS. *MOSEK Optimizer API for C 9.0.87*. 2017. Available from: `https://docs.mosek.com/9.0/capi/index.html`

[17] Agrawal, A.; Verschueren, R.; et al. A Rewriting System for Convex Optimization Problems. *Journal of Control and Decision*, volume 5, no. 1, 2018: pp. 42–60.

[18] Diamond, S.; Boyd, S. CVXPY: A Python-Embedded Modeling Language for Convex Optimization. *Journal of Machine Learning Research*, volume 17, no. 83, 2016: pp. 1–5.

[19] Stellato, B.; Banjac, G.; et al. OSQP: An Operator Splitting Solver for Quadratic Programs. *ArXiv e-prints*, Nov. 2017, `1711.08013`.

[20] O'Donoghue, B.; Chu, E.; et al. SCS: Splitting Conic Solver, version 2.1.0. `https://github.com/cvxgrp/scs`, Nov. 2017.

[21] O'Donoghue, B.; Chu, E.; et al. Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding. *Journal of Optimization Theory and Applications*, volume 169, no. 3, June 2016: pp. 1042–1068. Available from: `http://stanford.edu/~boyd/papers/scs.html`

Appendix **A**

Contents of enclosed CD

```
readme.txt ........................ the file with CD contents description
src ............................... the directory of source codes
  Julia ........................... Julia nootebooks
  Python .......................... Python nootebooks
  thesis .......................... the directory of LaTeX source codes of the thesis
text .............................. the thesis text directory
  thesis.pdf ...................... the thesis text in PDF format
```