NUMERICAL OPTIMIZATION IN THE CONTEXT OF CLASSIFICATION PROBLEMS

MARIA CAMERON

Contents

1. Motivation: classification problems

Classification problem[1] is one of major problems considered in data science and machine learning. Instances of this problem include text categorization, image recognition, separating results of screening medical tests to positive and negative, etc. We consider the classification problem with only two classes. This is not a big loss of generality, as in practice, if classification into $K > 2$ categories is required, for each $1 \le k \le K$, one solves the problem of attributing or not attributing a given object to category k. Mathematically, the input data for a classification problem are of the form of a collection of pairs vector-label:

$$(\mathbf{z}_i, y_i) \quad \mathbf{z}_i \in \mathbb{R}^D, \quad y_i \in \{1, -1\}, \quad i = 1, \dots, n, \tag{1}$$

[1]The first classification algorithm was proposed by R. A. Fisher in 1936 where he derived an optimal decision function to separate two Gaussian distributions (see [1]).

The mainstream approach to solving classification problems is to map the input vectors $\mathbf{z}_i$ to a feature space $X \subset \mathbb{R}^d$, i.e., $\mathbf{z}_i \mapsto \phi(\mathbf{z}_i) =: \mathbf{x}_i \in X$, in which the images of points with labels 1 and -1 are expected to be separable using a hyperplane. The space feature space X is often high-dimensional. The map ϕ is often nonlinear. I will give two examples borrowed from the review paper by L. Bottou, F. Curtis, and J. Nocedal "Optimization Methods for Large-Scale Machine Learning" [2] (see Section 2).

1.1. **Text categorization.** Reuters Corpus Volume I (RCV1) [3] is a manually categorized archive of over 800,000 news stories. Most stories consist of less than 1000 words (less than two A4 pages). Each story is mapped to the 47,152-dimensional feature space corresponding to the vocabulary of 47,152 words. In this space, points corresponding to each category are separable by a hyperplane from the others. Hence, the task is to find a "good" hyperplane $\mathbf{w}^\top \mathbf{x} + b = 0$ such that

$$\begin{cases} \mathbf{w}^\top \mathbf{x}_i + b > 0, & y_i = 1, \\ \mathbf{w}^\top \mathbf{x}_i + b < 0, & y_i = -1. \end{cases} \tag{2}$$

Once such a hyperplane is found, the *prediction function* h can be chosen to be

$$h(\mathbf{x}, \mathbf{w}, b) := \mathbf{w}^\top \mathbf{x} + b, \tag{3}$$

Then one evaluates

$$\mathsf{sign}\,(h(\mathbf{x}_i, \mathbf{w}, b)) \tag{4}$$

in order to attribute $\mathbf{x}_i$ to one class or the other.

In 1995, Cortes and Vapnik [1] proposed a successful approach based on duality called Support-Vector Machine or Support-Vector Network for finding the optimal hyperplane for which the right-hand sides of the inequalities in (2) are replaced with 1 and -1 respectively. In the same paper, extended this approach for nonlinear dividing hypersurfaces and applied it to a number of benchmark datasets including NIST handwritten digit dataset. We will discuss this approach as it is quite interesting.

However, as we will see, it involves some numerical difficulties that make it less appealing than an approach sharpened for robustness rather than for optimality. This approach is based on the use of a smooth *loss function* and regularization, *Tikhonov* or *lasso*. For example, a *log-loss* function is of the form

$$l(h, y) := \log(1 + \exp\left[-yh(\mathbf{x}_i, \mathbf{w}, b)\right]). \tag{5}$$

Moreover, the optimization problem is often regularized by adding an extra Tikhonov's regularizing term $\frac{\lambda}{2}\|\mathbf{w}\|^2$ with a positive parameter λ:

$$\min_{(\mathbf{w},b)\in\mathbb{R}^d\times\mathbb{R}} \frac{1}{n}\sum_{i=1}^{n} l\left[h(\mathbf{x}_i, \mathbf{w}, b), y_i)\right] + \frac{\lambda}{2}\|\mathbf{w}\|^2. \tag{6}$$

Adding such a term renders the optimization problem *convex*.

1.2. **Image recognition and deep neural networks.** The problem of image recognition is much harder than the problem of text categorization. Even the task of digit recognition is difficult as there is no rule describing properties of pixel combinations that define each digit (see, e.g. Fig. 2.1 in [2] or Wiki: MNIST database). As for today, the most successful approaches for image recognition are based on *deep neural networks* (DNNs). DNNs are often described with jargon borrowed from neuroscience as they were originally inspired by models of biological neurons.

The input vectors $\mathbf{z}_i \equiv \mathbf{x}_i^{(0)}$ are mapped to the feature space by a composition of functions of the form

$$\mathbf{x}_i^{(j)} = s\left(W_j \mathbf{x}_i^{(j-1)} + \mathbf{b}_j\right) \in \mathbb{R}^{d_j}, \quad j = 1, \dots, J, \tag{7}$$

where J is the number of *layers*, i.e., the number of functions in the composition, the $d_j \times d_{j-1}$-matrix W_j is vector $\mathbf{b}_j \in \mathbb{R}^{d_j}$ determine parameters of jth layer, and s is a chosen nonlinear *activation* function acting component-wise. Popular choices are ReLU (rectified linear unit) $s(x) = \max\{0, x\}$ and sigmoid $s(x) = (1 + \exp(-x))^{-1}$ shown in Fig. 1. To simplify notation, we introduce a parameter vector w that collects all parameters:

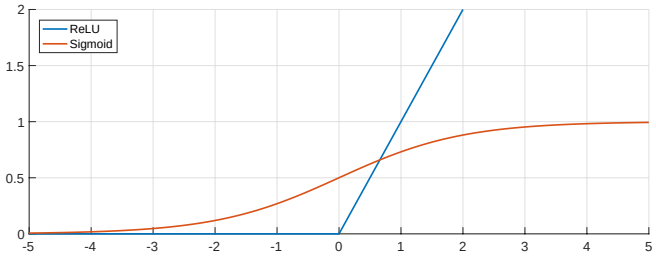

FIGURE 1. Popular choices of the activation function in DNNs: rectified linear unit (ReLU) $s(x) = \max\{0, x\}$ and sigmoid $s(x) = (1 + \exp(-x))^{-1}$.

$$\mathsf{w} := \{(W_1, \mathbf{b}_1), (W_2, \mathbf{b}_2), \dots, (W_J, \mathbf{b}_J)\}. \tag{8}$$

The optimization problem becomes

$$\min_{\mathsf{w} \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} l\left[h(\mathbf{x}_i, \mathsf{w}), y_i\right], \tag{9}$$

where l is the chosen loss function. In contrast to (9) this optimization problem is highly nonlinear and nonconvex, i.e., intractable to solve to global optimality [2].

Here are some examples of DNNs for image recognition[2]:

- AlexNet (2012), 8 layers. The original paper by A. Krizhevsky, I. Sutsever, and G. E. Hinton [4] is available here.
- ResNet (2016), a resudial neural network, proposed in [5] may have up to 152 layers. Apparently, ResNet-18 is implemented in Matlab.

2. A warm-up classification problem

Let us start with a simple classification problem. We are given a set of data $\{\mathbf{x}_i, y_i\}$, $i = 1, \dots, n$, where $\mathbf{x}_i \in \mathbb{R}^d$ are vectors and $y_i \in \{1, -1\}$ are labels. We assume that the sets of points with labels 1 and -1 are separable by a hyperplane. The problem posed in [1] is the following (see Fig. 2):

find a hyperplane $\mathbf{w}^\top \mathbf{x} + b = 0$ *such that*

$$f(\mathbf{w}, b) := \frac{1}{2}\|\mathbf{w}\|^2 \to \min \quad \text{subject to} \tag{10}$$

$$y_i \left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1, \quad i = 1, \dots, n. \tag{11}$$

Let us explain (10). The distance between the two parallel hyperplanes in (11), $\mathbf{w}^\top \mathbf{x}_i + b - 1 = 0$ and $\mathbf{w}^\top \mathbf{x}_i + b + 1 = 0$, is equal to the distance between the points of intersection of these hyperplanes with the normal line $\{\alpha \mathbf{w} \mid \alpha \in \mathbb{R}\}$. These two points are

$$\frac{1-b}{\|\mathbf{w}\|}\frac{\mathbf{w}}{\|\mathbf{w}\|} \quad \text{and} \quad \frac{-1-b}{\|\mathbf{w}\|}\frac{\mathbf{w}}{\|\mathbf{w}\|}. \tag{12}$$

Hence, the distance between these hyperplanes is

$$\rho := \left|\frac{1-b}{\|\mathbf{w}\|} - \frac{-1-b}{\|\mathbf{w}\|}\right| = \frac{2}{\|\mathbf{w}\|}. \tag{13}$$

Therefore, (10) maximizes the distance between the these two planes. The constraint (11) is equivalent to

$$\begin{cases} \mathbf{w}^\top \mathbf{x}_i + b \geq 1, & y_i = 1, \\ \mathbf{w}^\top \mathbf{x}_i + b \leq -1, & y_i = -1. \end{cases} \quad i = 1, \dots, n. \tag{14}$$

Compare it with (2).

The optimization problem (10)–(11) is a *quadratic program*, i.e., the objective function is quadratic, while the constraints defining the feasible set are linear. Note that the objective function is convex but not strictly convex. In order to solve it, we will need to learn some theory and methodology of constrained optimization.

3. Basics of constrained optimization

As the methods that we will use for solving the optimization problem (10)–(11) are suitable for a much larger class of problems than quadratic programs, we will generalize the problem in-hand to the following:

$$\begin{aligned} & f(\mathbf{x}) \to \min \quad \text{subject to} \\ & c_i(\mathbf{x}) = 0, \quad i \in \mathcal{E}, \quad \text{(equality constraints)} \\ & c_i(\mathbf{x}) \geq 0, \quad i \in \mathcal{I}, \quad \text{(inequality constraints)}. \end{aligned} \tag{15}$$

[2]Thanks to Avi Schwarzschild, AMSC PhD student.

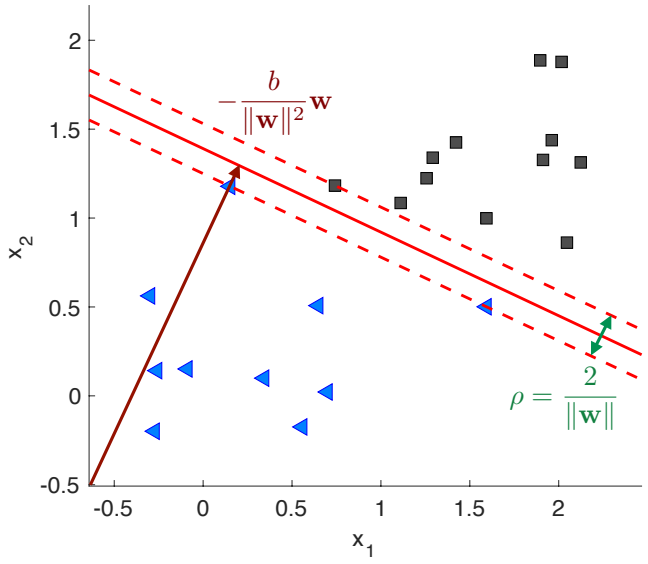

FIGURE 2. An illustration to optimization problem (10)–(11).

We assume that f and c_i, $i \in \mathcal{E} \cup \mathcal{I}$, are continuously differentiable.

In order to understand how to solve (15), first recall the method of Lagrange multipliers from your calculus course. A helpful quick reminder is given in the Wiki article "Lagrange Multiplier"–look at Figure 1 there.

Imagine that we have a single equality constraint $c(\mathbf{x}) = 0$. Then the minimizer of $f(\mathbf{x})$ subject to $c(\mathbf{x}) = 0$ lies at a point $\mathbf{x}^*$ such that the level set $\{\mathbf{x} \in \mathbb{R}^d \mid f(\mathbf{x}) = f(\mathbf{x}^*)\}$ is tangent to the hypersurface $c(\mathbf{x}) = 0$. Indeed, moving along the hypersurface $c(\mathbf{x}) = 0$ and keeping track of the values of $f(\mathbf{x})$, the extreme values of f will be achieved at the points where the level set of f is tangent to $c(\mathbf{x}) = 0$ (see Fig. 3(a)). At such points, the gradients of f and c will be parallel to each other, i.e., they will relate via

$$\nabla f(\mathbf{x}) = \lambda \nabla c(\mathbf{x}). \tag{16}$$

The factor λ is called *the Lagrange multiplier*. The condition (16) motivates the definition of *the Lagrangian function*

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda c(\mathbf{x}). \tag{17}$$

Stationary points of $L(\mathbf{x}, \lambda)$ are those where its gradient is zero, i.e.,

$$\nabla_{\mathbf{x}} L = \nabla f(\mathbf{x}) - \lambda \nabla c(\mathbf{x}) = 0, \tag{18}$$

$$\nabla L_\lambda = c(\mathbf{x}) = 0, \tag{19}$$

Hence every stationary point of $f(\mathbf{x})$ where $\mathbf{x}$ is restricted to $c(\mathbf{x}) = 0$ is a stationary point of $L(\mathbf{x}, \lambda)$.

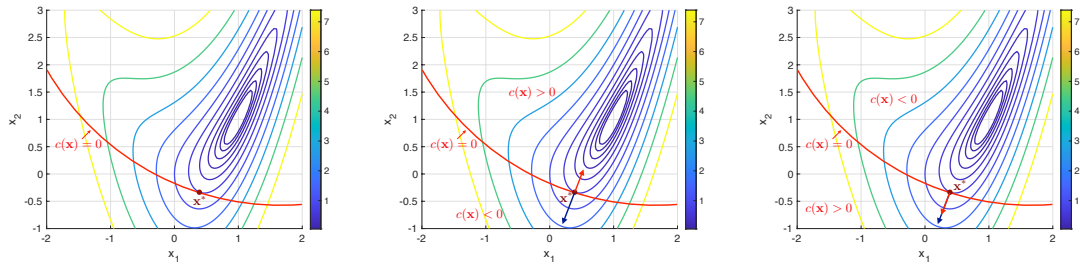

FIGURE 3. The geometric sense of Lagrange multiplier. (a): The solution to the equality-constrained minimization problem is reached at the point $\mathbf{x}^*$ where the level set of f is tangent to $c(\mathbf{x}) = 0$. (b): The unconstrained minimum of f is the solution to the inequality-constrained optimization problem $c(\mathbf{x}) \geq 0$. (c): At the constrained minimum of f, $\nabla f(\mathbf{x}^*) = \lambda \nabla c(\mathbf{x}^*)$, where $\lambda > 0$. *The level sets are those of the Rosenbrock function* $f(x_x, x_2) = (1 - x_1)^2 + (x_2 - x_1^2)^2$.

Now we replace the equality constraint $c(\mathbf{x}) = 0$ with the inequality constraint $c(\mathbf{x}) \geq 0$. For visuality, we assume that level sets of $f(\mathbf{x})$ are simple closed surfaces. This implies that f has a unique local minimizer, and it is its global minimizer.

- If the minimum of f lies in the region $c(\mathbf{x}) \geq 0$, the constrained minimum coincides with the unconstrained minimum. Apart from this global minimum, consider the point $\mathbf{x}^*$ where a level set of f touches the level set $c(\mathbf{c}) = 0$. The gradients $\nabla f(\mathbf{x}^*)$ and $\lambda \nabla c(\mathbf{x}^*)$ are antiparallel, i.e., $\nabla f(\mathbf{x}^*) = \lambda \nabla c(\mathbf{x}^*)$ for some $\lambda < 0$ (see Fig. 3(b)).
- If the global minimizer of f does not belong to the feasible set where $c(\mathbf{x}) \geq 0$ as in Fig. 3(c), the constrained minimum will be achieved at the point $\mathbf{x}^*$ lying at $c(\mathbf{x}) = 0$ where $\nabla f(\mathbf{x}^*) = \lambda \nabla c(\mathbf{x}^*)$ for some $\lambda > 0$. In this case, we say that the constraint $c(\mathbf{x}) \geq 0$ is *active* at the solution $\mathbf{x}^*$.

This illustrative example shows that the sign of the Lagrange multiplier is important and gives an intuition for the *Karush-Kuhn-Tucker* first-order optimality conditions.

3.1. **Karush-Kuhn-Tucker optimality conditions.** This section is based on J. Nocedal and S. Wright "Numerical Optimization" [6], Chapter 12.

Definition 1. *The active set* $\mathcal{A}(\mathbf{x})$ *at any feasible* $\mathbf{x}$ *consists of the equality constraint indices from* $\mathcal{E}$ *together with the indices of the inequality constraints* i *for which* $c_i(\mathbf{x}) = 0$*; that is,*

$$\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} \mid c_i(\mathbf{x}) = 0\}. \tag{20}$$

At any feasible point $\mathbf{x}$, the inequality constraint $c_i(\mathbf{x}) \geq 0$ is *active* if $c_i(\mathbf{x}) = 0$ and inactive if $c_i(\mathbf{x}) > 0$.

The *Lagrangian function* is defined by

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(\mathbf{x}). \tag{21}$$

Definition 2. *A feasible point* $\mathbf{x}^*$ *is a local solution of* (15) *if all feasible sequences* $\mathbf{z}_k \to \mathbf{x}^*$ *as* $k \to \infty$ *have the property that* $f(\mathbf{z}_k) \geq f(\mathbf{x}^*)$ *for all* k *sufficiently large.*

Definition 3. *A direction* $\mathbf{d}$ *at a feasible point* $\mathbf{x}$ *is feasible if* $\nabla c_i(\mathbf{x})^\top d = 0 \ \forall i \in \mathcal{E}$ *and* $\nabla c_i(\mathbf{x})^\top d \geq 0 \ \forall i \in \mathcal{A}(\mathbf{x}) \cap \mathcal{I}$.

Definition 4. *The vector* $\mathbf{d}$ *is a tangent vector to the feasible set* Ω *at a point* $\mathbf{x}$ *if there are a feasible sequence* $\mathbf{z}_k \to \mathbf{x}$ *and a sequence of positive scalars* $t_k \to 0$ *as* $k \to \infty$ *such that*

$$\lim_{k \to \infty} \frac{\mathbf{z}_k - \mathbf{x}}{t_k} = \mathbf{d}. \tag{22}$$

The set of all tangent vectors at $\mathbf{x}$ *is called the tangent cone and is denoted by* $T_\Omega(\mathbf{x})$.

The first-order optimality conditions known as the *Karush-Kuhn-Tucker (KKT) conditions* are stated in the following theorem.

Theorem 1. *Suppose* $\mathbf{x}^*$ *is a local solution of* (15) *where* f *and* c_i*'s are continuously differentiable, and the set of active constraints gradients*

$$\{\nabla c_i(\mathbf{x}^*), \ i \in \mathcal{A}(\mathbf{x}^*)\} \tag{23}$$

is linearly independent[3]*. Then there is a Lagrange multiplier vector* $\boldsymbol{\lambda}^* = \{\lambda_i\}_{i \in (\mathcal{E} \cup \mathcal{I})}$ *such that the following conditions are satisfied at* $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$

$$\nabla_{\mathbf{x}} L(\mathbf{x}^*, \boldsymbol{\lambda}^*) = 0, \tag{24}$$

$$c_i(\mathbf{x}^*) = 0 \quad \forall i \in \mathcal{E}, \tag{25}$$

$$c_i(\mathbf{x}^*) \geq 0 \quad \forall i \in \mathcal{I}, \tag{26}$$

$$\lambda_i^* \geq 0 \quad \forall i \in \mathcal{I}, \tag{27}$$

$$\lambda_i^* c_i(\mathbf{x}^*) = 0 \quad \forall i \in \mathcal{E} \cup \mathcal{I}. \tag{28}$$

We will prove Theorem 1 for the case where the set of constrains consists only of linear inequality constraints, i.e., for the problem

$$f(\mathbf{x}) \to \min \quad \text{subject to} \tag{29}$$
$$A\mathbf{x}^* - \mathbf{b} \geq 0, \quad A \text{ is } n \times d.$$

The proof for (29) follows the same steps as the proof for (15) given in [6] but is shorter as it requires fewer technicalities. One important point is that the LICQ condition is not required if all constraints are linear.

[3]This condition is called the *linear independence constraint qualification (LICQ).*

Proof. Without the loss of generality we assume that the active set consists of the first m inequalities: $\mathcal{A}(\mathbf{x}^*) = \{1, \dots, m\}$. The matrix consisting of the first m rows of A and the vector consisting of the first m components of $\mathbf{b}$ will be denoted by A, and b respectively.

Step 1. Show that the set of feasible directions is

$$\mathcal{F}(\mathbf{x}^*) = \{\mathbf{d} \in \mathbb{R}^d \mid \mathsf{A}\mathbf{d} \geq 0\} \tag{30}$$

and coincides with the tangent cone $T_\Omega(\mathbf{x}^*)$.

Indeed, since $\nabla(A\mathbf{x} - \mathbf{b})_i$ is the ith row of A, Definition 3 implies that a direction $\mathbf{d}$ is feasible if and only if $\mathsf{A}\mathbf{d} \geq 0$.

Furthermore, for any tangent vector $\mathbf{d}$ we have a feasible sequence $\mathbf{z}_k$ and a sequence of scalars $t_k \to 0$ such that

$$\mathbf{z}_k = \mathbf{x}^* + t_k\mathbf{d} + o(t_k).$$

Multiplying this equation by A and subtracting b we get:

$$0 \leq \mathsf{A}\mathbf{z}_k - \mathsf{b} = \underbrace{\mathsf{A}\mathbf{x}^* - \mathsf{b}}_{=0} + t_k\mathsf{A}\mathbf{d} + o(t_k) = t_k\mathsf{A}\mathbf{d} + o(t_k).$$

Dividing the last equality by t_k and letting $k \to 0$, we get $\mathsf{A}\mathbf{d} \geq 0$ which means that any tangent vector is a feasible direction, i.e., $T_\Omega(\mathbf{x}^*) \subseteq \mathcal{F}(\mathbf{x}^*)$.

On the other hand, if $\mathbf{d}$ is a feasible direction then the sequence $\mathbf{z}_k = \mathbf{x}^* + t_k\mathbf{d}$ is feasible for sufficiently small t_k. Indeed, multiplying this equality by A and subtracting b we get

$$\mathsf{A}\mathbf{z}_k - \mathsf{b} = \underbrace{\mathsf{A}\mathbf{x}^* - \mathsf{b}}_{=0} + t_k\mathsf{A}\mathbf{d} = t_k\mathsf{A}\mathbf{d} \geq 0$$

which implies that $\mathbf{d}$ is a tangent vector. Hence $\mathcal{F}(\mathbf{x}^*) \subseteq T_\Omega(\mathbf{x}^*)$

Step 2. Show that if $\mathbf{x}^*$ is a local solution of (29) then

$$\nabla f(\mathbf{x}^*)^\top \mathbf{d} \geq 0 \quad \forall \mathbf{d} \in T_\Omega(\mathbf{x}^*). \tag{31}$$

Indeed, assume the opposite: there is a tangent vector $\mathbf{d}$ such that $\nabla f(\mathbf{x}^*)^\top \mathbf{d} < 0$. Then there exist sequences $\mathbf{z}_k \to \mathbf{x}^*$, feasible, and $t_k \to 0$, $t_k > 0$, such that

$$f(\mathbf{z}_k) = f(\mathbf{x}^*) + t_k\nabla f(\mathbf{x}^*)^\top \mathbf{d} + o(t_k).$$

Hence, subtracting $f(\mathbf{x}^*)$ from both sides and dividing by t_k, we get:

$$0 \leq \frac{f(\mathbf{z}_k) - f(\mathbf{x}^*)}{t_k} = \underbrace{\nabla f(\mathbf{x}^*)^\top \mathbf{d}}_{<0} + o(1) < 0$$

for sufficiently small t_k – a contradiction. Therefore, (31) holds.

Step 3 (Farkas' lemma). Consider the cone

$$K := \left\{\mathsf{A}^\top\lambda \mid \lambda \in \mathbb{R}^m,\ \lambda_i \geq 0\ \forall 1 \leq i \leq m\right\}. \tag{32}$$

Note that K is convex and closed. Prove that for any vector $\mathbf{g} \in \mathbb{R}^d$ the following alternative takes place:

- **either** $\mathbf{g} \in K$, i.e., there is a vector $\lambda \in \mathbb{R}^m$ with nonnegative components such that $\mathbf{g} = \mathsf{A}^\top\lambda$,

- **or** there exists a vector $\mathbf{d} \in \mathbb{R}^d$ such that the hyperplane normal to $\mathbf{d}$ separates $\mathbf{g}$ and K, i.e.,

$$\mathbf{g}^\top \mathbf{d} < 0, \quad \text{while} \quad \mathsf{A}\mathbf{d} \geq 0. \tag{33}$$

First we show that the two alternatives cannot take place simultaneously. From converse, assume that $\mathbf{g} = \mathsf{A}^\top \lambda$ for $\lambda \geq 0$ while $\mathbf{g}^\top \mathbf{d} < 0$ and $\mathsf{A}\mathbf{d} \geq 0$. Then

$$0 > \mathbf{g}^\top \mathbf{d} = \mathbf{d}^\top \mathsf{A}^\top \lambda = (\underbrace{\mathsf{A}\mathbf{d}}_{\geq 0})^\top \underbrace{\lambda}_{\geq 0} \geq 0,$$

a contradiction.

Now we show that if $\mathbf{g} \notin K$ then (33) holds. Let $\mathbf{y}$ be the point of the cone K closest to $\mathbf{g}$ in terms of the Euclidean distance. Such a point exists as K is closed. Note that $\mathbf{y}$ may be the origin or a boundary point of K different from the origin – these two cases are depicted in Fig. 4. In both cases, the argument below is valid.

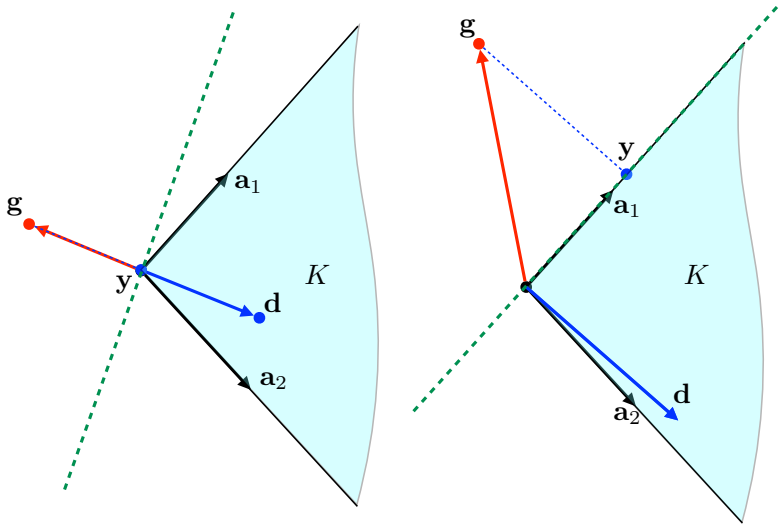

FIGURE 4. Illustration to step 3 of the proof of Theorem 1 in the case where $d = 2$ and $m = 2$. The vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ are the rows of the matrix A.

By the definition of cone, the whole ray emanating from the origin and passing through $\mathbf{y}$ belongs to K:

$$\{\alpha \mathbf{y} \in K \mid \alpha \geq 0\}.$$

Since $\mathbf{y}$ is the closest point to $\mathbf{g}$ in this ray as well, the minimum of the function

$$\phi(\alpha) := \frac{1}{2}(\alpha \mathbf{y} - \mathbf{g})^\top (\alpha \mathbf{y} - \mathbf{g}) = \frac{1}{2}\alpha^2 \|\mathbf{y}\|^2 - \alpha \mathbf{y}^\top \mathbf{g} + \frac{1}{2}\|\mathbf{g}\|^2$$

is achieved at $\alpha = 1$. Therefore,

$$0 = \frac{d\phi}{d\alpha}(1) = \|\mathbf{y}\|^2 - \mathbf{y}^\top \mathbf{g} = \mathbf{y}^\top(\mathbf{y} - \mathbf{g}).$$

Let $\mathbf{z} \in K$, $\mathbf{z} \neq \mathbf{y}$. Since K is convex, we have

$$\mathbf{y} + \theta(\mathbf{z} - \mathbf{y}) \in K \quad \forall \theta \in [0, 1].$$

By the minimizing property of $\mathbf{y}$ we have

$$\|\mathbf{y} + \theta(\mathbf{z} - \mathbf{y}) - \mathbf{g}\|^2 \geq \|\mathbf{y} - \mathbf{g}\|^2 \quad \forall \theta \in [0, 1].$$

Hence

$$2\theta(\mathbf{z} - \mathbf{y})^\top(\mathbf{y} - \mathbf{g}) + \theta^2\|\mathbf{z} - \mathbf{y}\|^2 \geq 0.$$

Dividing this equation by θ and taking limit $\theta \downarrow 0$, we get

$$0 \leq (\mathbf{z} - \mathbf{y})^\top(\mathbf{y} - \mathbf{g}) = \mathbf{z}^\top(\mathbf{y} - \mathbf{g}) - \underbrace{\mathbf{y}^\top(\mathbf{y} - \mathbf{g})}_{=0} = \mathbf{z}^\top(\mathbf{y} - \mathbf{g}), \quad \text{i.e.}$$

$$\mathbf{z}^\top(\mathbf{y} - \mathbf{g}) \geq 0 \quad \forall \mathbf{z} \in K. \tag{34}$$

Now we set

$$\mathbf{d} := \mathbf{y} - \mathbf{g}$$

and show that $\mathbf{d}$ satisfies (33). Note that $\mathbf{d} \neq 0$ as $\mathbf{g} \notin K$. Indeed, (34) means that

$$\mathbf{d}^\top \mathsf{A}^\top \lambda \geq 0 \quad \forall \lambda \geq 0,$$

which is true only if $\mathbf{d}^\top \mathsf{A}^\top \geq 0$, i.e, $\mathsf{A}\mathbf{d} \geq 0$. On the other hand,

$$\mathbf{d}^\top \mathbf{g} = \mathbf{d}^\top(\mathbf{y} - \mathbf{d}) = (\mathbf{y} - \mathbf{g})^\top(\mathbf{y} - \mathbf{d}) = \underbrace{(\mathbf{y} - \mathbf{g})^\top \mathbf{y}}_{=0} - \|\mathbf{d}\|^2 < 0.$$

Step 4. Now we set $\mathbf{g} = \nabla f(\mathbf{x}^*)$. By Farkas' lemma proven in Step 3, either $\nabla f(\mathbf{x}^*) \in K$, i.e., there is a vector λ with nonnegative components such that

$$\nabla f(\mathbf{x}^*) = \mathsf{A}^\top \lambda, \tag{35}$$

or there is a feasible direction $\mathbf{d}$ such that $\mathsf{A}\mathbf{d} \geq 0$ and $\nabla f(\mathbf{x}^*)^\top \mathbf{d} < 0$ which means that $\mathbf{x}^*$ is not a local solution. Therefore, since $\mathbf{x}^*$ is a local solution, (35) takes place.

Finally, we set the Lagrange multipliers with indices in the active set ($1 \leq i \leq m$) to be equal to λ and we set to zero the rest of them, i.e.,

$$\boldsymbol{\lambda} = \begin{bmatrix} \lambda \\ 0 \end{bmatrix}.$$

The conditions (24)–(28) are readily verified. □

3.2. **Active-set method.** This section is based on J. Nocedal and S. Wright "Numerical Optimization" [6], Chapter 16. We consider a convex quadratic program (QP)

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top G\mathbf{x} + \mathbf{c}^\top \mathbf{x} \;\to\; \min \quad \text{subject to} \tag{36}$$

$$\mathbf{a}_i^\top \mathbf{x} = \mathbf{b}_i, \quad i \in \mathcal{E} \tag{37}$$

$$\mathbf{a}_i^\top \mathbf{x} \geq b_i, \quad i \in \mathcal{I}. \tag{38}$$

Convexity of the QP means that the matrix G is positive definite.

A quite natural method to solve (36)–(38) is *the active-set method*. At every step of this algorithm, we solve a QP

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top G\mathbf{x} + \mathbf{c}^\top \mathbf{x} \;\to\; \min \quad \text{subject to} \tag{39}$$

$$\mathbf{a}_i^\top \mathbf{x} = \mathbf{b}_i, \quad i \in \mathcal{W}, \tag{40}$$

(41)

where $\mathcal{W}$ is the current set of active constraints. We will denote by $A_\mathcal{W}$ the matrix whose rows at $\mathbf{a}_i$, $i \in \mathcal{W}$, and $\mathbf{b}_\mathcal{W}$ the vector of b_i's, $i \in \mathcal{W}$. Then (40) becomes $A_\mathcal{W}\mathbf{x} = \mathbf{b}_\mathcal{W}$.

We do so as follows. Let $\mathbf{x}_k$ be the current iterate. We want to find a step $\mathbf{p}_k$ to obtain the next iterate

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k.$$

In the case if $\mathbf{x}_{k+1}$ is not feasible, we shorten the step length just enough to make $\mathbf{x}_{k+1}$ remain feasible. Plugging $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$ into (39) we get

$$\frac{1}{2}(\mathbf{x}_k + \mathbf{p}_k)^\top G(\mathbf{x}_k + \mathbf{p}_k) + \mathbf{c}^\top(\mathbf{x}_k + \mathbf{p}_k) = \frac{1}{2}\mathbf{p}_k^\top G\mathbf{p}_k + (G\mathbf{x}_k + \mathbf{c})^\top \mathbf{p}_k + f(\mathbf{x}_k).$$

We observe that $G\mathbf{x}_k + \mathbf{c} \equiv \nabla f(\mathbf{x}_k)$ and $f(\mathbf{x}_k)$ is independent of $\mathbf{p}_k$ and hence does not affect the minimizer. Moreover, $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$ into (40) we get the following constraint for $\mathbf{p}_k$:

$$A_\mathcal{W}(\mathbf{x}_k + \mathbf{p}_k) - \mathbf{b}_\mathcal{W} = \underbrace{A_\mathcal{W}\mathbf{x} - \mathbf{b}_\mathcal{W}}_{=0} + A_\mathcal{W}\mathbf{p}_k = A_\mathcal{W}\mathbf{p}_k = 0.$$

Therefore, the minimization problem for $\mathbf{p}_k$ reduces to

$$\frac{1}{2}\mathbf{p}^\top G\mathbf{p} + \nabla f(\mathbf{x}_k)^\top \mathbf{p} \;\to\; \min \quad \text{subject to} \tag{42}$$

$$A_\mathcal{W}\mathbf{p}_k = 0. \tag{43}$$

(44)

The KKT system for (42)–(43) is

$$G\mathbf{p}_k + \nabla f(\mathbf{x}_k) - A_\mathcal{W}^\top \boldsymbol{\lambda} = 0, \quad A_\mathcal{W}\mathbf{p}_k = 0.$$

It can be rewritten in the matrix form:

$$\begin{bmatrix} G & A_\mathcal{W}^\top \\ A_\mathcal{W} & 0 \end{bmatrix} \begin{bmatrix} -\mathbf{p}_k \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \nabla f(\mathbf{x}_k) \\ 0 \end{bmatrix}. \tag{45}$$

Exercise Show that the matrix in (45) with G $d \times d$ being symmetric positive definite and A $m \times d$ having linearly independent rows, is of *saddle-point type*, i.e., it has d positive eigenvalues and m negative ones. *Hint: Omit the subscript $\mathcal{W}$ for brevity. Find matrices X and S (S is called the **Schur compliment**) such that*

$$\begin{bmatrix} G & A^\top \\ A & 0 \end{bmatrix} = \begin{bmatrix} I & 0 \\ X & I \end{bmatrix} \begin{bmatrix} G & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & X^\top \\ 0 & I \end{bmatrix}.$$

Then use Sylvester's Law of Inertia (look it up!) to finish the proof.

Exercise Consider an equality-constrained QP (G is symmetric)

$$\frac{1}{2}\mathbf{x}^\top G\mathbf{x} + \mathbf{c}^\top \mathbf{x} \;\to\; \min \quad \text{subject to} \tag{46}$$

$$A\mathbf{x} = \mathbf{b}. \tag{47}$$

(48)

Assume that A is full rank (i.e., its rows are linearly independent) and $Z^\top GZ$ is positive definite where Z is a basis for the null-space of A, i.e., $AZ = 0$.

(1) Write the KKT system for this case in the matrix form.
(2) Show that the matrix of this system K is invertible. *Hint: assume that there is a vector $\mathbf{z} := (\mathbf{x}, \mathbf{y})^\top$ such that $K\mathbf{z} = 0$. Consider the form $\mathbf{z}^\top K\mathbf{z}$, and so on ... . You should arrive at the conclusion that then $\mathbf{z} = 0$.*
(3) Conclude that there exists a unique vector $(\mathbf{x}^*, \boldsymbol{\lambda}^*)^\top$ that solves the KKT system. Note that since we have only equality constraints, positivity of $\boldsymbol{\lambda}$ is irrelevant.

According to the claims in the exercises, (45) has a unique solution $(\mathbf{p}, \boldsymbol{\lambda})$. We consider two cases.

- If $\mathbf{p} \neq 0$, we can make a step in the direction $\mathbf{p}$. Let us show that this is a descend direction for $f(\mathbf{x})$, i.e., $\nabla f(\mathbf{x}_k)^\top p < 0$. Indeed, we have

$$G\mathbf{p} + \nabla f(\mathbf{x}_k) = A^\top \boldsymbol{\lambda}$$
$$A\mathbf{p} = 0.$$

  Multiplying the first equation by $\mathbf{p}^\top$ we get

$$\mathbf{p}^\top (G\mathbf{p} + \nabla f(\mathbf{x}_k)) = \mathbf{p}^\top G\mathbf{p} + \nabla f(\mathbf{x}_k)^\top \mathbf{p} = \mathbf{p}^\top A^\top \boldsymbol{\lambda} = (\underbrace{A\mathbf{p}}_{=0})^\top \boldsymbol{\lambda} = 0.$$

  Since G is positive definite and $\mathbf{p} \neq 0$ by assumption, we conclude that $\nabla f(\mathbf{x}_k)^\top \mathbf{p} < 0$ which means that $\mathbf{p}$ is a descend direction for f.

  Next, we start moving from $\mathbf{x}_k$ along the direction $\mathbf{p}$ until we either travel the full distance $\|\mathbf{p}\|$ or activate another constraint:

$$\mathbf{a}_i \mathbf{x} = b_i \quad \text{for some} \quad i \notin \mathcal{W}.$$

  Hence, in order to find step length α, we consider

$$\mathbf{a}_i^\top (\mathbf{x}_k + \alpha \mathbf{p}_k) = \mathbf{a}_i \mathbf{x}_k + \alpha \mathbf{a}_i^\top \mathbf{p}_k = b_i \quad \text{for all} \quad i \notin \mathcal{W}.$$

Since $\mathbf{a}_i^\top \mathbf{x}_k > b_i \; \forall i \notin \mathcal{W}$, if $\mathbf{a}_i^\top \mathbf{p} \geq 0$, we can only reinforce this constraint by moving along $\mathbf{p}$. In contrast, if $\mathbf{a}_i^\top \mathbf{p} < 0$, we may hit the constraint prior to traveling the full distance. Hence, the step length α is given by

$$\alpha = \min \left\{ 1, \min_{i \notin \mathcal{W}, \mathbf{a}_i^\top \mathbf{p} < 0} \frac{b_i - \mathbf{a}_i^\top \mathbf{p}}{\mathbf{a}_i^\top \mathbf{p}} \right\}. \tag{49}$$

Finally, we set $\alpha_k = \alpha$ and $\mathbf{p}_k = \mathbf{p}$.

- If $\mathbf{p} = 0$, we cannot move anywhere from $\mathbf{x}_k$ given the set of constraints $\mathcal{W}$. Hence, there are two possibilities.
  - We have found the solution to (36)–(38). To check if this is the case, we look at the vector $\boldsymbol{\lambda}$ of Lagrange's multipliers, and check their signs. If all Lagrange multipliers corresponding to inequality constrains are nonnegative, a solution is found, and we terminate the iterations.
  - If there is a negative Lagrange multiplier corresponding to an inequality constraint, we remove it from $\mathcal{W}$ and proceed with iterations.

A Matlab function `ASM` implements the active-set method for the case where f is allowed to be nonconvex and nonquadratic, and the set of constraints consists of inequality constraints only. A driver for it with the Rosenbrock function and the feasible region being a hexagon is the function `ASMdriver`.

```
 function ASMdriver()
%%  the Rosenbrock function
a = 5;
func = @(x,y)(1-x).^2 + a*(y - x.^2).^2;  % Rosenbrock's function
gfun = @(x)[-2*(1-x(1))-4*a*(x(2)-x(1)^2)*x(1);2*a*(x(2)-x(1)^2)]; % gradient of f
Hfun = @(x)[2 + 12*a*x(1)^2 - 4*a*x(2), -4*a*x(1); -4*a*x(1), 2*a]; % Hessian of f
lsets = exp([-3:0.5:2]);
%% constraints
Nv = 6;
t = linspace(0,2*pi,Nv+1);
t(end) = [];
t0 = 0.1;
verts = [0.1+cos(t0+t);0.1+sin(t0+t)];
R = [0,-1;1,0];
A = (R*(circshift(verts,[0,-1])-verts))';
b = verts(1,:)'.*A(:,1) + verts(2,:)'.*A(:,2); % b_i = a_i*verts(:,i)
x = [-0.5;0.5];
W = [];
[xiter,lm] = ASM(x,gfun,Hfun,A,b,W);
%% graphics
close all
fsz = 16;
figure(1);
```

```
hold on;
n = 100;
txmin = min(verts(1,:))-0.2;
txmax = max(verts(1,:))+0.2;
tymin = min(verts(2,:))-0.2;
tymax = max(verts(2,:))+0.2;
tx = linspace(txmin,txmax,n);
ty = linspace(tymin,tymax,n);
[txx,tyy] = meshgrid(tx,ty);
ff = func(txx,tyy);
contour(tx,ty,ff,lsets,'Linewidth',1);
edges = [verts,verts(:,1)];
plot(edges(1,:),edges(2,:),'Linewidth',2,'color','k');
plot(xiter(1,:),xiter(2,:),'Marker','.','Markersize',20,'Linestyle','-',...
    'Linewidth',2,'color','r');
xlabel('x','Fontsize',fsz);
ylabel('y','Fontsize',fsz);
set(gca,'Fontsize',fsz);
colorbar;
grid;
daspect([1,1,1]);
end

function [xiter,lm] = ASM(x,gfun,Hfun,A,b,W)
%% minimization using the active set method (Nocedal & Wright, Section 16.5)
% Solves f(x) --> min subject to Ax >= b
% x = initial guess, a column vector
TOL = 1e-10;
dim = length(x);
g = gfun(x);
H = Hfun(x);
iter = 0;
itermax = 1000;
m = size(A,1); % the number of constraints
% W = working set, the set of active constrains
I = (1:m)';
Wc = I; % the compliment of W
xiter = x;
while iter < itermax
    % compute step p: solve 0.5*p'*H*p + g'*p --> min subject to A(W,:)*p = 0
    AW = A(W,:); % LHS of active constraints
    % fix H if it is not positive definite
    ee = sort(eig(H),'ascend');
```

```
if ee(1) < 1e-10
    lam = -ee(1) + 1;
else
    lam = 0;
end
H = H + lam*eye(dim);
if ~isempty(W)
    M = [H, -AW';AW,zeros(size(W,1))];
    RHS = [-g;zeros(size(W,1),1)];
else
    M = H;
    RHS = -g;
end
aux = M\RHS;
p = aux(1:dim);
lm = aux(dim+1:end);
if  norm(p) < TOL  % if step == 0
    if ~isempty(W)
        lm = AW'\g; % find Lagrange multipliers
        if min(lm) >= 0 % if Lagrange multipliers are positive, we are done
            % the minimizer is one of the corners
            fprintf('A local solution is found, iter = %d\n',iter);
            fprintf('x = [\n'); fprintf('%d\n',x);fprintf(']\n');
            break;
        else % remove the index of the most negative multiplier from W
            [lmin,imin] = min(lm);
            W = setdiff(W,W(imin));
            Wc = setdiff(I,W);
        end
    else
        fprintf('A local solution is found, iter = %d\n',iter);
        fprintf('x = [\n'); fprintf('%d\n',x);fprintf(']\n');
        break;
    end
else % if step is nonzero
    alp = 1;
    % check for blocking constraints
    Ap = A(Wc,:)*p;
    icand = find(Ap < -TOL);
    if ~isempty(icand)
        % find step lengths to all possible blocking constraints
        al = (b(Wc(icand)) - A(Wc(icand),:)*x)./Ap(icand);
        % find minimal step length that does not exceed 1
```

```
            [almin,kmin] = min(al);
            alp = min(1,almin);
        end
        x = x + alp*p;
        g = gfun(x);
        H = Hfun(x);
        if alp < 1
            W = [W;Wc(icand(kmin))];
            Wc = setdiff(I,W);
        end
    end
    iter = iter + 1;
    xiter = [xiter,x];
end
if iter == itermax
    fprintf('Stopped because the max number of iterations %d is performed\n',iter);
end
end
```

3.3. **Finding the dividing hyperplane for classification problems.** Now we return to the constrained minimization problem (10)–(11), a QP with a convex but not strictly convex objective function and linear inequality constraints. We rewrite it in a form convenient for feeding into the active-set solver:

$$\mathsf{w} := \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}, \quad \mathsf{A} = \begin{bmatrix} y_1 & \dots & y_1 \\ \vdots & & \vdots \\ y_n & \dots & y_n \end{bmatrix} \odot \begin{bmatrix} \mathbf{x}_1^\top & \to, & 1 \\ \vdots & & \vdots \\ \mathbf{x}_n^\top & \to, & 1 \end{bmatrix}, \quad \mathsf{b} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix},$$

where the symbol $\odot$ denotes the componentwise matrix multiplication. The vector w is $(d+1) \times 1$, the matrix $\mathsf{A} = [X, 1_{n\times 1}]$ is $n \times (d+1)$, where X is the matrix whose rows are $\mathbf{x}_i^\top$, $i = 1, \dots, d$, and the right-hand side vector b is $n \times 1$; n is the number of data points and d is the dimension of the data space. Hence, the optimization problem becomes:

$$f(\mathsf{w}) = \frac{1}{2}\mathsf{w}^\top \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & 0 \end{bmatrix} \mathsf{w} \to \min \quad \text{subject to} \tag{50}$$

$$\mathsf{A}\mathsf{w} = \mathsf{b}. \tag{51}$$

An example of the active-set algorithm applied to find the optimal line dividing samples from two 2D Gaussian distributions with means $(0.5, 0.5)^\top$ and $(1.5, 1.5)^\top$ and variances 1 is shown in Fig. 2. There are two complications with this approach.

- The active-set method needs a feasible point to start with. It is hard-to-find by inspection even in 2D. In order to find the initial guess for w, I wrote an extremely

simple routine shown below. I define a loss function

$$l(\mathsf{w},\mathsf{A},\mathsf{b}) := \sum_{i=1}^{n} \mathsf{ReLU}(\mathsf{b}_i - (\mathsf{Aw})_i), \quad \text{where} \quad \mathsf{ReLU}(x) = \max\{0, x\},$$

and organize a gradient descend with constant step length. Note that $l = 0$ and $\nabla l = 0$ if w is a feasible point.

```
function [w,l,lcomp] = FindInitGuess(w,A,b)
relu = @(w)max(w,0);
drelu = @(w)ones(size(w)).*sign(relu(w));
l = sum(relu(b - A*w));
iter = 0;
itermax = 10000;
step = 0.1;
while l > 0 && iter < itermax
    dl = sum(-drelu(b - A*w)'*A,1)';
    if norm(dl) > 1
        dl = dl/norm(dl);
    end
    w = w - step*dl;
    l = sum(relu(b - A*w));
    iter = iter + 1;
end
fprintf('%d iterations: x = [%d,%d],f = %d\n',iter, w(1),w(2),l);
lcomp = relu(b - A*w);
end
end
```

The output `x` of this function is the desired initial guess. If the output `l` is nonzero, this means that the algorithm failed to find a feasible point. Why I need the output `lcomp` you will see shortly – I need it to address the issue in the next item.

- As you can easily imagine, samples from the mentioned two Gaussian distributions may not be separable by a line. This is a very common situation in real-life problems. For example, screening medical tests give results that indicate a risk for some condition but do not diagnose it exactly for the reason that data from normal and affected patients are not separable, though do exhibit certain trends (see e.g. this example). A way to address this issue proposed in [1] is to introduce *soft margins*, i.e., a set of n penalty variables $\xi_i \geq 0$, $i = 1, \ldots, n$. The new problem becomes:

$$f(\mathbf{w}, \mathbf{b}, \boldsymbol{\xi}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i, \tag{52}$$

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = 1, \ldots, n, \tag{53}$$

$$\xi_i \geq 0, \quad i = 1 \ldots, n. \tag{54}$$

To write this problem in the matrix form we define $\mathsf{x} := [\mathbf{w}, b, \boldsymbol{\xi}]^\top \in \mathbb{R}^{d+1+n}$ and get:

$$f(\mathsf{x}) = \frac{1}{2}\mathsf{x}^\top \begin{bmatrix} I_{d\times d} & 0_{d\times(n+1)} \\ 0_{(n+1)\times d} & 0_{(n+1)\times(n+1)} \end{bmatrix} \mathsf{x} + C[0_{1\times(d+1)}, 1_{1\times(n+1)}]\mathsf{x} \to \min, \tag{55}$$

$$\begin{bmatrix} [\mathbf{y}\cdot 1_{1\times(d+1)}] \odot [X, 1_{n\times 1}] & I_{n\times n} \\ 0_{n\times d+1} & I_{n\times n} \end{bmatrix} \mathsf{x} \geq \begin{bmatrix} 1_{n\times 1} \\ 0_{n\times 1} \end{bmatrix}. \tag{56}$$

The positive constant C is chosen depending on what what goal is being pursued. I chose $C = 10^3$, a large number. To find an initial guess, I run the same routine for finding the initial guess. the initial guess for the vector of errors $\boldsymbol{\xi}$ is the output `lcomp`. An example of separating samples from the same two Gaussian distributions with soft marging is shown in Fig. 5

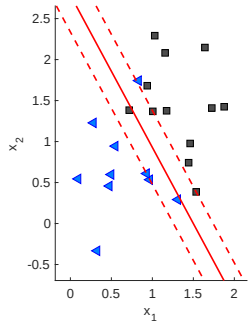

FIGURE 5. Separating samples from two Gaussian distributions with soft margins (55)–(56)

.

4. Duality

The solution proposed in [1] to address the issue of finding the initial guess for the QP arising in the classification problem is to consider the dual problem [6] (Section 12.9). We will first discuss what it is in general, and then set it up for the classification problem.

Recall the definition of the Lagrangian function:

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \boldsymbol{\lambda}^\top \mathbf{c}(\mathbf{x}), \tag{57}$$

where $\mathbf{c}(\mathbf{x}) = [c_1(\mathbf{x}), \ldots, c_n(\mathbf{x})]^\top$ is the vector of the left-hand sides of the constraints $c_i(\mathbf{x}) \geq 0$. The solution to the KKT system maximizes L with respect to $\boldsymbol{\lambda}$ and while minimizes it with respect to $\mathbf{x}$. This suggests to consider the *dual problem* defined by

$$q(\boldsymbol{\lambda}) = \inf_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}) \to \max \quad \text{subject to} \quad \boldsymbol{\lambda} \geq 0. \tag{58}$$

The domain of q is defines at the set of those $\boldsymbol{\lambda}$ for which q is finite, i.e.,

$$\mathcal{D}_q := \{\boldsymbol{\lambda} \mid q(\boldsymbol{\lambda}) > -\infty\}.$$

In general, the dual problem might be easier or harder than the original problem.

Now we will derive the dual problem for the classification problem (50)–(51) (see [1], Appendix A). The Lagrangian for (50)–(51) is

$$L(\mathbf{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}\mathbf{w}^\top \mathbf{w} - \sum_{i=1}^{n} \lambda_i \left[y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 \right]. \tag{59}$$

Let $\mathbf{w}^*\mathbf{x} + b^* = 0$ be the optimal hyperplane and $\boldsymbol{\lambda}^*$ be the corresponding vector of Lagrange multipliers. Taking the gradient of L with respect to $\mathbf{w}$ and b and evaluating it at $(\mathbf{w}^*, b^*, \boldsymbol{\lambda}^*)$ we obtain:

$$\frac{\partial L}{\partial \mathbf{w}}(\mathbf{w}^*, b^*, \boldsymbol{\lambda}^*) = \mathbf{w}^* - \sum_{i=1}^{n} \lambda_i y_i \mathbf{x}_i = 0, \tag{60}$$

$$\frac{\partial L}{\partial b}(\mathbf{w}^*, b^*, \boldsymbol{\lambda}^*) = -\sum_{i=1}^{n} \lambda_i y_i = 0. \tag{61}$$

Therefore, the optimal vector $\mathbf{w}^*$ relates to the optimal $\boldsymbol{\lambda}$ via

$$\mathbf{w}^* = \sum_{i=1}^{n} \lambda_i y_i \mathbf{x}_i, \tag{62}$$

and the optimal $\boldsymbol{\lambda}^*$ also satisfies the following equality condition

$$\mathbf{y}^\top \boldsymbol{\lambda} = 0. \tag{63}$$

Substituting (62) and (63) into (59) we obtain the dual function $q(\boldsymbol{\lambda})$:

$$\begin{aligned} q(\boldsymbol{\lambda}) &= \frac{1}{2}\left(\sum_{i=1}^{n} \lambda_i y_i \mathbf{x}_i^\top\right)\left(\sum_{j=1}^{n} \lambda_j y_j \mathbf{x}_j\right) - \sum_{i=1}^{n} \lambda_i \left[y_i \left\{ \left(\sum_{j=1}^{n} \lambda_j y_j \mathbf{x}_j^\top\right) \mathbf{x}_i + b \right\} - 1 \right] \\ &= \frac{1}{2}\boldsymbol{\lambda}^\top D \boldsymbol{\lambda} - \boldsymbol{\lambda}^\top D \boldsymbol{\lambda} - b\mathbf{y}^\top \boldsymbol{\lambda} + 1_{1\times n}\boldsymbol{\lambda} \\ &= -\frac{1}{2}\boldsymbol{\lambda}^\top D \boldsymbol{\lambda} + 1_{1\times n}\boldsymbol{\lambda}, \text{where} \end{aligned} \tag{64}$$

$$D = \left(\mathbf{y}\mathbf{y}^\top\right) \odot \left(\begin{bmatrix} \mathbf{x}_1^\top & \rightarrow \\ \vdots & \\ \mathbf{x}_n^\top & \rightarrow \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_n \\ \downarrow & & \downarrow \end{bmatrix} \right) = \left(\mathbf{y}\mathbf{y}^\top\right) \odot \left(XX^\top\right). \tag{65}$$

In summary, we obtain the following constrained optimization problem for $\boldsymbol{\lambda}$:

$$q(\boldsymbol{\lambda}) = -\frac{1}{2}\boldsymbol{\lambda}^\top D \boldsymbol{\lambda} + 1_{1\times n}\boldsymbol{\lambda} \rightarrow \max \quad \text{subject to} \tag{66}$$

$$\mathbf{y}^\top \boldsymbol{\lambda} = 0, \tag{67}$$

$$\boldsymbol{\lambda} \geq 0. \tag{68}$$

As The vector $\boldsymbol{\lambda}$ is found, the optimal $\mathbf{w}^*$ is found from (62). Then we select two support vectors

$$\mathbf{x}_{i_+} \quad \text{such that} \quad i_+ = \arg\max\{\lambda_i \mid 1 \le i \le n,\ y_i = 1\} \quad \text{and}$$
$$\mathbf{x}_{i_-} \quad \text{such that} \quad i_- = \arg\max\{\lambda_i \mid 1 \le i \le n,\ y_i = -1\},$$

and find b from the identities $(\mathbf{w}^*)^\top \mathbf{x}_{i_+} + b = 1$ and $(\mathbf{w}^*)^\top \mathbf{x}_{i_-} + b = -1$:

$$b = -\frac{1}{2}(\mathbf{w}^*)^\top \left(\mathbf{x}_{i_+} + \mathbf{x}_{i_-}\right).$$

Exercise Is the matrix D positive definite? Either prove the the statement, or give a counterexample.

The dual problem (66)–(70) has the following advantages.

- The vector $\boldsymbol{\lambda} = 0$ is always a feasible point for (66)–(70) as all constraints hold at it. Hence, we do not need to search for an initial guess.
- If the number of constraints is large, we can solve the problem iteratively where the dimensionality of the problem at each iteration is as modest as desired. We take a subset of data points $1, \dots, m$, $m < n$, and solve (66)–(70) for it. Most of the inequality constrains will be inactive, i.e., most of λ_i's will be zeros. Those data points $\mathbf{x}_i$ corresponding to $\lambda_i > 0$ are called *the support vectors* – this is why this approach is referred to as the *support-vector network* or the *support-vector machine*. Only these few support vectors are carried to the next iteration. The next m data points are added to them to form the constrains for the next problem-to-solve. And so on until there are no more data points to add.
- The dual approach is also generalizable for the case of soft margins.

  Exercise Derive the dual problem for (52)–(54).
- Finally, the dual approach is generalizable for nonlinear separating hypersurfaces by means of replacing the dot product with a kernel function: $\mathbf{x}_i^\top \mathbf{x}_j$ is replaced with $K(\mathbf{x}_i, \mathbf{x}_j)$ in the definition of the matrix D in (65) [1]:

  $$D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j).$$

  For example, $K(\mathbf{x}_i, \mathbf{x}_j)$ can be

  $$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j + 1)^p \quad \text{or} \quad K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left\{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2}\right\}.$$

Let us make a few implementational remarks.

(1) The dual problem is a constrained maximization problem. To convert it to a more accustomed constrained minimization problem, just take $-q(\boldsymbol{\lambda})$ as an objective function: $q(\boldsymbol{\lambda}) \mapsto -q(\boldsymbol{\lambda})$.

(2) We can get rid of the equality constraint $\mathbf{y}^\top \boldsymbol{\lambda} = 0$ by incorporating it into the objective function. Let us express λ_n via λ_i's, $1 \le i \le n-1$:

$$\lambda_n = -\frac{1}{y_n} \sum_{i=1}^{n-1} y_i \lambda_i = -\frac{1}{y_n} \mathbf{y}_{1:(n-1)}^\top \lambda, \tag{69}$$

where $\mathbf{y}_{1:(n-1)} := (y_1, \ldots, y_{n-1})^\top$ and $\lambda := (\lambda_1, \ldots, \lambda_{n-1})^\top$). Next, we write D as a block matrix separating its last column and last row:

$$D = \begin{bmatrix} D_{00} & D_{01} \\ D_{10} & D_{11} \end{bmatrix},$$

or, in Matlab notaion:

```
D00 = D(1:end-1,1:end-1);
D01 = D(1:end-1,end);
D10 = D(end,1:end-1);
D11 = D(end,end);
```

Plugging (69) into the quadratic form $\boldsymbol{\lambda}^\top D\boldsymbol{\lambda}$ we get:

$$\begin{aligned}\boldsymbol{\lambda}^\top D\boldsymbol{\lambda} &= [\lambda^\top, -\frac{1}{y_n}\mathbf{y}_{1:(n-1)}^\top\lambda] \begin{bmatrix} D_{00} & D_{01} \\ D_{10} & D_{11} \end{bmatrix} \begin{bmatrix} \lambda \\ -\frac{1}{y_n}\mathbf{y}_{1:(n-1)}^\top\lambda \end{bmatrix} \\ &= \lambda^\top D_{00}\lambda - \frac{1}{y_n}\lambda^\top \mathbf{y}_{1:(n-1)} D_{10}\lambda - \frac{1}{y_n}\lambda^\top D_{01}\mathbf{y}_{1:(n-1)}^\top\lambda + \frac{1}{y_n^2}\lambda^\top \mathbf{y}_{1:(n-1)} D_{11}\mathbf{y}_{1:(n-1)}^\top\lambda.\end{aligned}$$

Hence the matrix of the quadratic form in the new constrained minimization problem is:

$$H = D_{00} + \frac{1}{y_n^2}\mathbf{y}_{1:(n-1)}\mathbf{y}_{1:(n-1)}^\top D_{11} - \frac{1}{y_n}\mathbf{y}_{1:(n-1)} D_{10} - \frac{1}{y_n} D_{01}\mathbf{y}_{1:(n-1)}^\top. \tag{70}$$

The linear term of $q(\boldsymbol{\lambda})$ is modified to:

$$\mathbf{c} := 1_{(n-1)\times 1} - \frac{1}{y_n}\mathbf{y}_{1:(n-1)}. \tag{71}$$

The resulting problem for $\lambda := (\lambda_1, \ldots, \lambda_{n-1})^\top$ becomes:

$$q(\lambda) = \frac{1}{2}\lambda^\top H\lambda - \mathbf{c}^\top\lambda \to \min \quad \text{subject to} \tag{72}$$

$$\lambda \geq 0, \tag{73}$$

$$-\frac{1}{y_n}\mathbf{y}_{1:(n-1)}^\top\lambda \geq 0. \tag{74}$$

(3) The matrix H is symmetric but not, in general, positive definite. To make sure that every step of the active-set method reduces the objective function, we add a multiple of the identity matrix to H. Let $-\mu_0$ is the most negative eigenvalue of H. Then the matrix $\tilde{H} := H + \mu I$ where $\mu > \mu_0$ (I set $\mu = \mu_0 + 1$ in my code) is symmetric positive definite. Then the modified KKT matrix is invertible, and its inverse is [7] (see Section 3.3):

$$K^{-1} = \begin{bmatrix} \tilde{H} & A^\top \\ A & 0 \end{bmatrix}^{-1} = \begin{bmatrix} \tilde{H}^{-1} + \tilde{H}^{-1}A^\top S^{-1}A\tilde{H}^{-1} & -\tilde{H}^{-1}A^\top S^{-1} \\ -S^{-1}A\tilde{H}^{-1} & S^{-1} \end{bmatrix}, \tag{75}$$

$$S = -A\tilde{H}^{-1}A^\top \quad \text{is the Schur compliment.} \tag{76}$$

Exercise Let $(\mathbf{p}^*, \boldsymbol{\lambda}^*)$ be the solution to the modified KKT system

$$\begin{bmatrix} \tilde{H} & A^\top \\ A & 0 \end{bmatrix} = \begin{bmatrix} -\mathbf{p} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \nabla f \\ 0 \end{bmatrix}.$$

Show that $\mathbf{p}^*$ is a descend direction i.e., $\nabla f^\top \mathbf{p} < 0$ provided that columns of $A^\top$ are linearly independent and $n < d$. *Hint: First try to get it yourself. If you get stuck, look into* [7].

5. Unconstrained optimization problem for classification problems

Classification problems are often much more complicated than those where two sample sets can be separated by a hyperplane. Often one first needs to map the input data to a feature space using a nonlinear map, find a dividing hyperplane there, and then use its pre-image in the input space as the classifier. A simple nonlinear benchmark classification problem is the Swiss roll shown in Fig. 6(a). This example is set so that it is suffices to

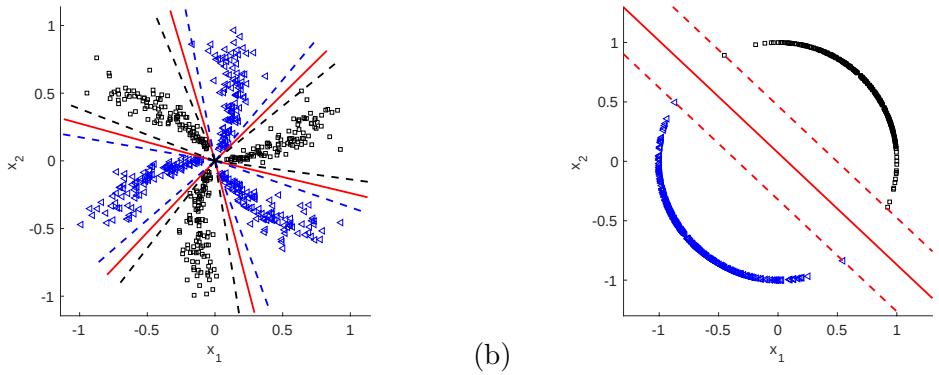

Figure 6. (a): Swiss roll dataset of 600 points and dividing rays (solid) together with margins (dashed). For "petal" k, the data are generated by adding a 2D Gaussian random variable with mean 0 and variance $0.1r$ to the curves $\phi_k(r) = {}^{k\pi}/_6 + 0.5r$. Here, (r, ϕ) are the polar coordinates. (b): The data mapped to the feature space by (77) where they are linearly separable.

define the feature space by taking polar angle of each data point $\mathbf{x}_i$, multiply by three (this operation will collapses all blue petals and all black petals, and take sines and cosines of these angles to avoid the issue with discontinuity and periodicity. In Matlab syntax, the map to the feature space is:

$$f(\mathbf{x}_i) := [\sin(3\phi(\mathbf{x})), \cos(3\phi(\mathbf{x}))]^\top, \tag{77}$$

where $\phi(\mathbf{x})$ is the polar angle of $\mathbf{x}$. Fig. 6(b) shows that this map to a 2D feature space is sufficient for making the data linearly separable. Finally, the SVM [1] is used to find the dividing lines and the margins that are mapped back to the data space (Fig. 6(a)).

While the solution in Fig. 6 looks nice, it is tuned only for this particular problem. Moreover, if we increase the twist in the curves around which we sample data points, e.g., we define $\phi_k(r) = {}^{k\pi}/_6 + r$ instead of $\phi_k(r) = {}^{k\pi}/_6 + 0.5r$, this classification method will break. We can do SVN with soft margins, but this will not help to adjust the shape of dividing surfaces. Adding r to the feature space won't help either (see Fig. 7).

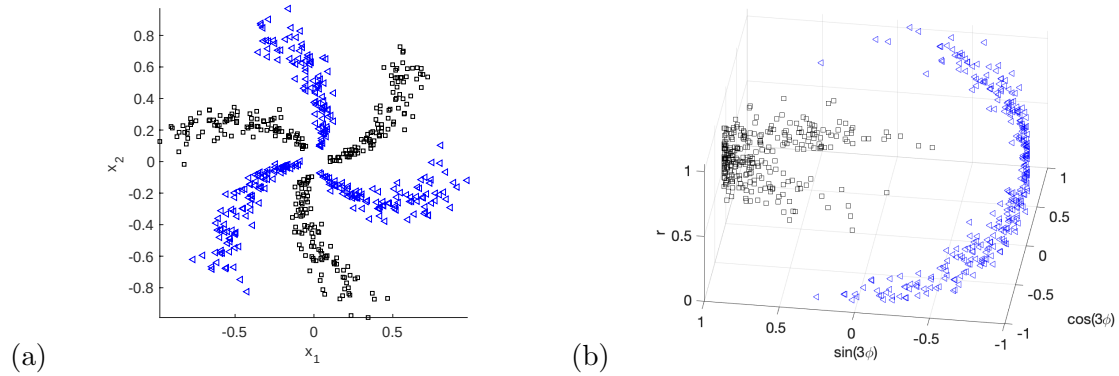

FIGURE 7. (a): Swiss roll dataset of 600 points. For "petal" k, the data are generated by adding a 2D Gaussian random variable with mean 0 and variance $0.1r$ to the curves $\phi_k(r) = {}^{k\pi}/_6 + r$. Here, (r, ϕ) are the polar coordinates. (b): The data mapped to the feature space: $\mathbf{x} \mapsto [\sin(3\phi(\mathbf{x})), \cos(3\phi(\mathbf{x})), r]^\top$ are not separable by a hyperplane.

We can custom-design a special feature space specifically for this problem, but this will not help to solve other problems. This problem is just a toy problem that is only of interest as a test problem for generic techniques.

The approach that proved its power is based on the use of deep neural networks (NN). One can set up a DNN with three hidden layers to solve this problem[3] as follows. The labels for class i will be the standard basis vector $\mathbf{e}_i \in \mathbb{R}^K$ where K is the number of classes; $K = 2$ in our case. Let the nonlinear mapping to the feature space be

$$\Phi(\mathbf{x}) = \sigma(A_3\sigma(A_2\sigma(A_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) + \mathbf{b}_3). \tag{78}$$

The nonlinear function σ (applied entrywise) can be chosen to be ReLU which is popular in image recognition, or a smooth function like $(1 + e^{-x})^{-1}$ or $\tanh(x)$ that are more preferable in other applications like solving PDEs and MD sampling. We expect the data in the feature space to be linearly separable by $A_4\Phi(\mathbf{x}) + \mathbf{b}_4$ where A_4 is $2 \times d_3$, and $\mathbf{b}_4 \in \mathbb{R}^2$. The matrices A_1, A_2, A_3 have dimensions $d_1 \times 2$, $d_2 \times d_1$, and $d_3 \times d_2$ respectively, and vectors $\mathbf{b}_i \in \mathbb{R}^{d_i}$, $i = 1, 2, 3$. The numbers d_i can be large (e.g., between 100 and 1000).

[3] I thank Avi Schwarzschild (AMSC, UMD) for this example who provided this example in his final report for AMSC664.

The model described here is called a *multilayer perceptron*. The loss function can be chosen as

$$\mathcal{L}(w) := \frac{1}{N}\sum_{i=1}^{n} \|h(\mathbf{x}_i; w) - \mathbf{y}_i\|^2, \tag{79}$$

where $w := \{(A_j, \mathbf{b}_j)\}_{i=1}^4$ is the vector of parameters comprising all matrices A_j and shifts $\mathbf{b}_j$, and $h(\mathbf{x}; w) := A_4\phi(\mathbf{x}; \tilde{w}) + \mathbf{b}_4$ is the prediction function, $\tilde{w} := \{(A_j, \mathbf{b}_j)\}_{i=1}^3$.

This example demonstrates how the classification problem can be reduced to an unconstrained minimization problem. Moreover, the objective function is of the form of sum of squares of nonlinear functions taking advantages of which can be exploited.

6. An overview of methods for unconstrained optimization

Key references for this section are [6] and [2]. The most widely used methods for unconstrained optimization are summarized in Fig. 8. Methods for unconstrained optimization

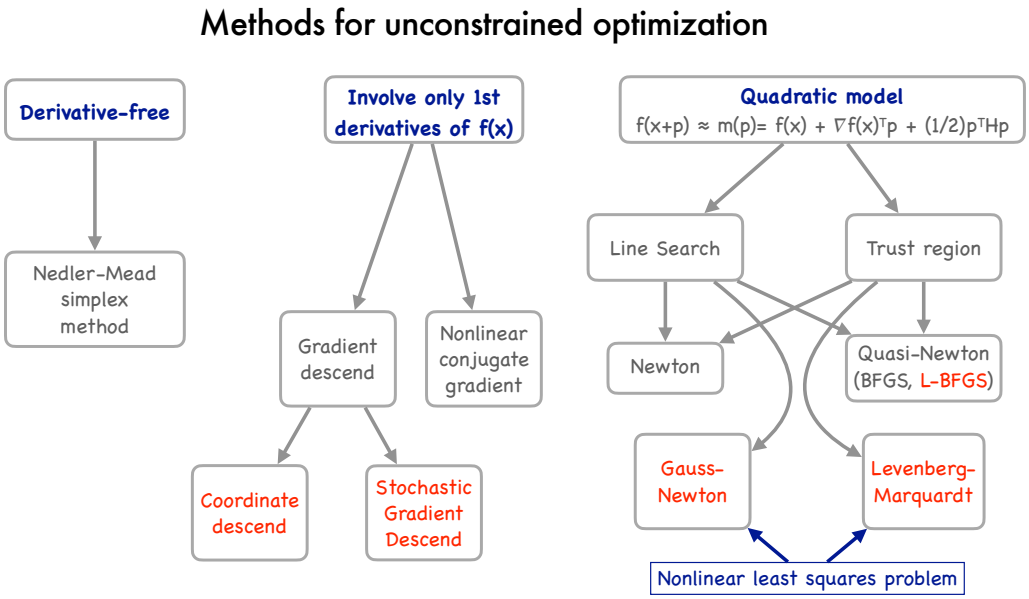

FIGURE 8. The most widely used methods for unconstrained optimization and their classification.

can be divided into three categories described in the three respective subsections below.

6.1. **Derivative-free methods.** Derivative-free methods only require function evaluations. A well-known method in this category is the Nedler-Mead downhill simplex method [6] (Section 9.5). Matlab's function `fminsearch`, very handy for small problems, employs a derivative-free method.

6.2. **Methods involving only first derivative evaluation.** The pivotal method in this category is *gradient descend* a.k.a. *steepest descend*. **Please read** D. Bindel's notes (Sections 8, 9, and 10) about gradient descend and related issues. This is the most straightforward minimization method (as the forward Euler method for solving ODEs), which, in the context of solving practical optimization problems can be attacked from two opposite directions. On one hand, its convergence is slow:

$$\|\mathbf{x}_k - \mathbf{x}^*\| = O(\rho^k) \quad \text{where} \quad \rho = 1 - \frac{2}{1+\kappa(A)},$$

where $A = \nabla f^2(\mathbf{x}^*)$ is the Hessian matrix of f at the local minimizer $\mathbf{x}^*$ and $\kappa(A) = \lambda_{\max}(A)/\lambda_{\min}(A)$. (Note that if $\mathbf{x}^*$ is a nondegenerate local minimizer, then A is symmetric positive definite). An example depicting how the gradient descend requires many iterations to converge when the problem is poorly scaled is shown in Fig. 9. Methods

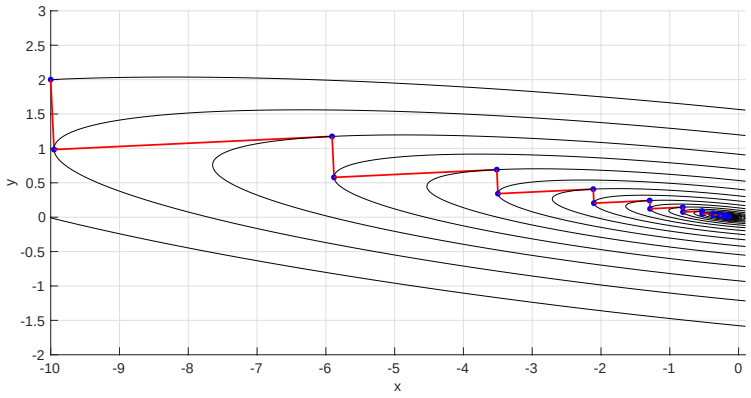

FIGURE 9. Iterations of the gradient descend starting from $\mathbf{x}_0 = (-10, 2)^\top$ and $f(\mathbf{x}) = \frac{1}{2}x_1^2 + 4x_1x_2 + 20x_2^2 + 0.1x_1 + 0.2x_2$.

for unconstrained optimization can be divided into three categories described in the three respective subsections below. Methods that use or build second derivative information converge significantly faster. Another group of methods in this category is the one of *nonlinear conjugate gradient* methods whose iterations are almost as cheap but they tend to perform better, especially on badly scaled problems. This group is inspired by the conjugate gradient method for solving large linear systems of algebraic equations with symmetric positive definite matrices.

However, slow convergence of gradient descend may be advantageous in some situations (see Section 9 in Bindel's notes linked above). On the other hand, the gradient of a function may be to costly to evaluate in large-scale optimization problems arising in machine learning. *Stochastic gradient descend* became a method of choice in TensorFlow and PyTorch. It is designed for case where the objective function is a sum of functions for each sample, it gives a good and cheap estimate for the gradient at each step, and its efficiency

is independent of the number of samples. The other method used in large-scale problems is *coordinate descend*. While this method, in contrast with gradient descend, is not guaranteed to converge to a stationary point of $f(\mathbf{x})$ (i.e., a point $\mathbf{x}^*$ such that $\nabla f(\mathbf{x}^*) = 0$ if f is nonconvex [8] (there is an example of a nonconvex continuously differentiable function of three variables for which a cyclic coordinate descend method, with step lengths chosen by exact one-dimensional minimization, cycles without converging to a stationary point). The appeal of coordinate descend is its easy implementation and cheap iterations. These facts made this method a subject of active contemporary research.

6.3. **Methods using quadratic models.** Finally, there are methods that a based on approximating f by a quadratic model at each step:

$$f(\mathbf{x}+\mathbf{p}) \sim m(\mathbf{p}) := f(\mathbf{x}) + \nabla f(\mathbf{x})^\top \mathbf{p} + \frac{1}{2}\mathbf{p}^\top H \mathbf{p},$$

where H is some approximation to the Hessian $\nabla^2 f(\mathbf{x})$. These methods are divided into two families according the strategy they use for choosing step direction and step length at their iterations: *line search methods* and *trust region methods*. Line search methods first pick search direction $\mathbf{p}$ and then choose step length. Certainly, all methods in the previous item belong to this family. Trust region methods iterate the other way around: they first pick the upper bound for step length Δ and then solve constraint minimization problem with the objective function being the current model $m(\mathbf{p})$ and the constraint $\|\mathbf{p}\| \le \Delta$.

Next, for each of these strategies, methods are classified according to the way they obtain the matrix H to build the quadratic model. If the dimensionality of the problem is not too large and the true Hessian is available and not too expensive to evaluate, *Netwon's method* is a good method to try. For finding local minima of Lennard-Jones clusters (for dimensions $\lesssim 45$), I evaluated the Hessian by finite differences: the exact Hessian was available but its evaluations was taking flops. For higher-dimensional and/or more complicated problems, evaluation of the Hessian becomes impractical. Then a good choice is *BFGS*, a quasi-newton method motivated by the secant method for solving 1D nonlinear equations. BFGS updates its approximate for the Hessian at each iteration incorporating the gained knowledge regarding the change in the gradient with the change in the argument (see [6], Section 2.2):

$$\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k) \approx H(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k).$$

There is a version of BFGS called *L-BFGS* (L stands for limited memory) that is designed for handling large-scale problems.

If the objective function has a special structure, for example

$$f(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{n} \|f_i(\mathbf{x})\|^2,$$

it can be exploited for a cheap approximation for the Hessian. We will discuss two methods that do it: it Gauss-Newton (line search) and *Levenberg-Marquardt* (trust region).

7. Stochastic gradient descend

The key reference for this section is [2]. Stochastic gradient descend originates from the

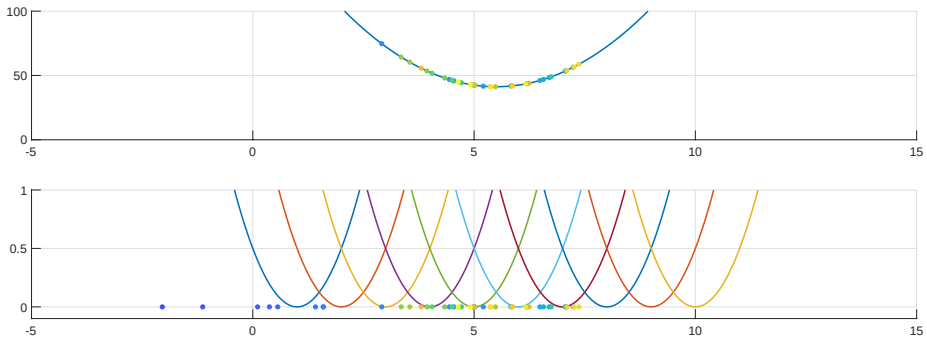

FIGURE 10. An example illustrating how stochastic gradient descend with a constant step length first rushes to the region where the minima of the individual functions are and then bounces around forever. Here, $f(x) = {}^1\!/_2 \sum_{i=1}^{10} (x-i)^2$, step length is 0.3, batch size is 1. The iteration numbers are indicated by the `parula` colormap going from blue to yellow.

work by Robbins and Monroe (1951) [9]. It is designed for minimizing functions of the form

$$f(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} f_i(\mathbf{x}), \quad \text{N is large.} \tag{80}$$

One picks the batch size $m \ll N$. Then, at each iteration, a subset of indices $\{i_1, \ldots, i_m\} \subset \{1, \ldots, N\}$ (a *batch* is randomly chosen so that index has an equal probability to be selected, and the following step is made:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \left[\frac{1}{m} \sum_{j=1}^{m} \nabla f_{i_j}(\mathbf{x}_k) \right], \tag{81}$$

The expression in the square brackets is a stochastic approximation to $\nabla f(\mathbf{x}_k)$. Why is this a reasonable approximation? Often, there are many samples $\mathbf{x}_i$ in the training set (each sample $\mathbf{x}_i$ defines the corresponding f_i), and these samples can be split into groups consisting of similar samples. In this case, using just a subset of samples for estimating the gradient will give almost as good result but will be cheaper by the factor ${}^m\!/_N$. Moreover, typically, the dimensionality of machine learning optimization problems is very large. Hence, evaluating all N gradients may cause a computer memory problem.

References

[1] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[2] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.

[3] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *Journal of Machine Learning Research*, vol. 5, pp. 361–397, 2004.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proc. Computer Vision and Pattern Recognition (CVPR)*, vol. 6, pp. 770–778, 2016.

[6] J. Nocedal and S. Wright, *Numerical Optimization*. Springer, 2 ed., 2006.

[7] M. Benzi, G. H. Golub, and J. Liesen, "Numerical solution of saddle point problems," *Acta Numerica*, pp. 1–137, 2005.

[8] M. J. D. Powell, "On search directions for minimization algorithms," *Mathematical Programming*, vol. 4, pp. 193–201, 1973.

[9] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, pp. 400–407, 1951.