Linear Predictions

Ju Sun*

October 6, 2024

Overview We first interpret supervised learning from the viewpoint of function approximation, and then survey classical linear prediction models and algorithms, including linear least squares for regression, and Perceptron, simple SVM, logistic regression for binary classification.

1 Function approximation view of supervised learning

Given: a data set $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$ (called *training set*) so that $\boldsymbol{y}_i \approx f_*(\boldsymbol{x}_i)\ \forall i$, where f_* is an unknown underlying function to be estimated. For all $i \in [N]$, the "$\approx$" sign in $\boldsymbol{y}_i \approx f_*(\boldsymbol{x}_i)$ is to allow noise or other errors over $\boldsymbol{y}_i = f_*(\boldsymbol{x}_i)$, e.g., $\boldsymbol{y}_i = f(\boldsymbol{x}_i) + \boldsymbol{\varepsilon}_i$ for Gaussian noise $\boldsymbol{\varepsilon}_i$. Some terminology:

$\boldsymbol{x}_i$ is called the **input**/**predictor** (in statistics)/**features** (in pattern recognition),
$\boldsymbol{y}_i$ is called the **output**/**response** (in statistics)/**label** (in pattern recognition).

There are three steps in a typical supervised learning workflow:

- **Step 1: Modeling**. Choose a family/set of functions $\mathcal{H}$, called a *hypothesis class* or *hypothesis set*, so that there exists an $f_\diamond \in \mathcal{H}$ that is "close" to f_*. Often, $\mathcal{H}$ should be *reasonably large* to ensure that there is indeed a good approximation $f_\diamond$ to f_*, and also *reasonably small/simple* so that such an $f_\diamond$ can be found efficiently in the computation.
- **Step 2: Computation (or Training)**. Design an algorithm to find such an $f_\diamond$. In modern machine learning, one often first formulates the learning problem as an optimization problem and then develops numerical optimization algorithms to solve it—this is why optimization is a crucial component of modern machine learning[1].
  - **Optimization formulation**. A family of natural and popular formulation is *structural risk minimization, or SRM* (it is called this under certain additional probabilistic assumptions on the training set; we will talk more about this when introducing statistical learning theory later.):

$$\min_{f \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^N \ell(\boldsymbol{y}_i, f(\boldsymbol{x}_i)) + \mathcal{R}(f). \tag{1.1}$$

*Department of Computer Science and Engineering, University of Minnesota at Twin Cities. Email: jusun@umn.edu.

[1]The department of Industrial and Systems Engineering (ISyE) of UMN offers the course *Optimization for Machine Learning* that covers popular scalable numerical optimization methods for solving large-scale machine learning problems. Other good resources include [Sra12, Sta].

Here, ℓ denotes the loss function chosen to measure the difference between $\boldsymbol{y}_i$ and $f(\boldsymbol{x}_i)$. Obviously, $\min_f \; 1/N \cdot \sum_{i=1}^{N} \ell(\boldsymbol{y}_i, f(\boldsymbol{x}_i))$ tries to ensure that $\boldsymbol{y}_i \approx f(\boldsymbol{x}_i)$ for all i. The second term $\mathcal{R}(\boldsymbol{x})$ is typically called *regularizer* or *regularization term*, which puts certain preferences on f to be found: this is often needed when there are multiple or even infinitely many $f_\diamond$ that are good—perhaps because the $\mathcal{H}$ we choose is larger than necessary—so that we have to restrict our search to certain f's that are practically interesting.

– **Optimization algorithm**. For very simple problems, Eq. (1.1) may admit a closed-form analytical solution. But in modern machine learning, this is very rare and iterative numerical optimization methods are almost always needed to solve Eq. (1.1). Depending on the learning problem and the choice of formulation,

(i) Eq. (1.1) can be an unconstrained or constrained optimization problem. In general, unconstrained optimization problems are (much) easier to solve than constrained ones. So for applications with large-scale datasets, one is often willing to make reasonable compromises during modeling and tries to formulate learning problems into unconstrained optimization problems or constrained optimization problems with very simple constraints so that scalable optimization algorithms can be developed;

(ii) Eq. (1.1) can be a convex or nonconvex problem. We will provide a quick review of convex analysis and optimization later. The analysis and optimization of convex problems are much more mature than that of nonconvex ones. So, there is an overall preference for convex formulations. But the revival of deep learning after 2010 has substantially changed this—optimization problems in deep learning are always highly nonconvex.

- **Step 3: Generalization (Testing)**. Measure how close the $f_\diamond$ found from **Step 2** is to f_*. Since we often do not know the true f_*, generalization can only be measured indirectly, e.g., by evaluating the average of

$$d(f_*(\boldsymbol{x}_j), f_\diamond(\boldsymbol{x}_j)) \approx d(\boldsymbol{y}_j, f_\diamond(\boldsymbol{x}_j)) \tag{1.2}$$

over unseen (i.e., test) dataset $\{(\boldsymbol{x}_j, \boldsymbol{y}_j)\}_{j=1}^{M}$. Here $d(\cdot,\cdot)$ is a difference (error) function that may or may not be the same as the ℓ above. To study generalization in a rigorous manner, especially to quantify the relationship between the size of training set (i.e., *sample complexity*) and generalization, we need to put additional assumptions on the training set, e.g., the data points are sampled iid (i.e., independent and identically distributed) from an underlying distribution. We will talk about this in later lectures on statistical learning theory.

Below, we start with linear regression and linear classification problems, and illustrate Steps 1 & 2. We will derive their generalization properties in the learning theory lectures.

When $\boldsymbol{y}_i$'s in the training set are categorical, i.e., indicating the memberships of the inputs $\boldsymbol{x}_i$'s in a set of categories (e.g., $\{\text{cat}, \text{dog}, \text{else}\}$ for image inputs, $\{\text{COVID}, \text{Non-COVID}\}$ given patients' symptoms), the learning problem is often modeled as classification. Otherwise, it will be modeled as regression.

2 Linear regression

For simplicity, we assume that $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$ for all $i \in [N]$ (remember that our convention is that scalars are nonbold small letters.).

2.1 Choosing the hypothesis class

In linear regression, we model the relationship between $\boldsymbol{x}$ and y as linear—arguably the simplest possible:

$$y_i \approx \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b \; \forall\, i \in [N], \quad \boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R}. \tag{2.1}$$

In other words, the hypothesis class is the set of all linear functions in $\boldsymbol{x}$, which we can write as[2]

$$\mathcal{H}_L = \left\{ \boldsymbol{x} \mapsto \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b : \boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R} \right\}. \tag{2.2}$$

2.2 Formulation

Once we decide on the hypothesis class, we are ready to formulate the problem as an optimization problem. When using the SRM framework, we need to choose an appropriate loss ℓ and regularizer $\mathcal{R}$. For simplicity, let us choose squared loss, i.e., $\ell(y_i, f(\boldsymbol{x}_i)) = (y_i - f(\boldsymbol{x}_i))^2$ for all i, and suppose that we do not need regularization now. This leads to a least-squares formulation:

$$\min_{\boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R}} \; \frac{1}{N} \sum_{i=1}^{N} (y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - b)^2. \tag{2.3}$$

Now we are to turn this formulation into an equivalent yet compact form using matrix notations—this facilitates more direct translation of the mathematical expressions into modern numerical programming languages that are optimized for matrix (or tensor) computations (e.g., `Numpy` in `Python`, `PyTorch`, `Julia`). We append each $\boldsymbol{x}_i$ with an additional coordinate 1, so that $\boldsymbol{x}_i' = \left[\begin{smallmatrix} \boldsymbol{x}_i \\ 1 \end{smallmatrix}\right] \in \mathbb{R}^{d+1}$; correspondingly, we concatenate $\boldsymbol{w}$ and b into $\boldsymbol{w}' = \left[\begin{smallmatrix} \boldsymbol{w} \\ b \end{smallmatrix}\right] \in \mathbb{R}^{d+1}$. It is easy to verify that

$$\langle \boldsymbol{w}', \boldsymbol{x}_i' \rangle = \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b \; \forall\, i \in [N]. \tag{2.4}$$

We call this the *homogeneous form of linear functions*. This allows us to write Eq. (2.3) as

$$\min_{\boldsymbol{w}' \in \mathbb{R}^{d+1}} \; \frac{1}{N} \sum_{i=1}^{N} (y_i - \langle \boldsymbol{w}', \boldsymbol{x}_i' \rangle)^2. \tag{2.5}$$

The prime notation $(\cdot)'$ looks messy. Since we often use the homogeneous form, **with slight abuse of notation**, we will just omit the $(\cdot)'$ from $\boldsymbol{w}'$ and $\boldsymbol{x}_i'$, knowing that the homogeneous notation will be explicitly stated or can be inferred from the dimension of $\boldsymbol{w}$ (i.e., $\boldsymbol{w} \in \mathbb{R}^{d+1}$). So we have

$$\min_{\boldsymbol{w} \in \mathbb{R}^{d+1}} \; \frac{1}{N} \sum_{i=1}^{N} (y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle)^2. \tag{2.6}$$

Last step, if we write

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^N \quad \text{and} \quad \boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^\intercal \\ \vdots \\ \boldsymbol{x}_N^\intercal \end{bmatrix} \in \mathbb{R}^{N \times (d+1)}, \tag{2.7}$$

[2] Recall that we typically represent a set as: {generic form of elements in the set : constraints on elements in the set if any}. In Eq. (2.2), a generic linear function in $\boldsymbol{x}$ can be represented as $\boldsymbol{x} \mapsto \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$, and the constraints are $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$—which are vacuous.

and recall the definition of vector ℓ_2 norm, we arrive at a compact form of Eq. (2.6):

$$\boxed{\min_{\boldsymbol{w} \in \mathbb{R}^{d+1}} \; g(\boldsymbol{w}) \doteq \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2,} \tag{2.8}$$

where we have omitted the factor $1/N$ in the objective, as it does not affect the solution. Our next job is to solve the optimization problem Eq. (2.8) to find a good linear model that fits the data well.

2.3 Solution via optimality condition

Our least-squares problem is unconstrained, and also the objective function $g(\boldsymbol{w})$ is relatively simple. So we shall first try the analytical approach, i.e., using optimality conditions to see if they lead to somewhere.

We now quickly review the optimality conditions for unconstrained optimization problems. Consider a minimization problem (it is sufficient to consider minimization problems, as maximization problems of the form $\max_{\boldsymbol{z} \in \mathbb{R}^n} f(\boldsymbol{z})$ are equivalent to $\min_{\boldsymbol{z} \in \mathbb{R}^n} -f(\boldsymbol{z})$—they have the same optimizer(s))

$$\min_{\boldsymbol{z} \in \mathbb{R}^n} \; f(\boldsymbol{z}). \tag{2.9}$$

A point $\boldsymbol{z}_0 \in \mathbb{R}^n$ is a *local minimizer* of $f(\boldsymbol{z})$ if there exists a radius η such that $f(\boldsymbol{z}_0) \le f(\boldsymbol{z})$ for all $\boldsymbol{z}$ satisfying $\|\boldsymbol{z} - \boldsymbol{z}_0\|_2 \le \eta$; in other words, if $f(\boldsymbol{z}_0)$ is no larger than any other $f(\boldsymbol{z})$ in an η-ball around $\boldsymbol{z}_0$. The value $f(\boldsymbol{z}_0)$ is called a *local minimum*. So, minimizers concern the optimization variable, and minimums (or minima) concern the objective value.

For minimization problems, optimality conditions are mathematical conditions that any local minimizer must satisfy, and hence they are helpful for locating local minimizers both analytically and numerically.

Theorem 2.1 (First-order necessary condition of optimality for unconstrained problems)**.** *Assume f is first-order differentiable at $\boldsymbol{z}_0$. If $\boldsymbol{z}_0$ is a local minimizer, then $\nabla f(\boldsymbol{z}_0) = \mathbf{0}$.*

The zero-gradient condition is a necessary condition for local minimizers, but not sufficient. A point where the gradient is zero (also called *first-order stationary point*, or FOSP) can be a local minimizer, a local maximizer, or a saddle point. It turns out the condition becomes sufficient for the family of convex functions—more on this when we talk about support vector machines and kernel methods. A salient feature of convex functions is that a local minimizer is also a global minimizer[3]. One way to tell convexity is through the Hessian.

Lemma 2.2 (Convexity through Hessian)**.** *Assume f is second-order differentiable. Then f is convex if and only if $\nabla^2 f(\boldsymbol{z}) \succeq \mathbf{0}$ for all $\boldsymbol{z}$.*

Here, $\succeq \mathbf{0}$ means being positive semidefinite ($\succ \mathbf{0}$ means being positive definite). Recall that for a symmetric matrix $\boldsymbol{M}$,

$$\boldsymbol{M} \succeq \mathbf{0} \Longleftrightarrow \boldsymbol{v}^\intercal \boldsymbol{M} \boldsymbol{v} \ge 0 \; \forall \boldsymbol{v} \quad \text{and} \quad \boldsymbol{M} \succ \mathbf{0} \Longleftrightarrow \boldsymbol{v}^\intercal \boldsymbol{M} \boldsymbol{v} > 0 \; \forall \boldsymbol{v} \ne \mathbf{0}. \tag{2.10}$$

Theorem 2.3 (First-order sufficient condition of optimality for unconstrained convex problems)**.** *Assume f is convex and first-order differentiable at $\boldsymbol{z}_0$. If $\nabla f(\boldsymbol{z}_0) = \mathbf{0}$, then $\boldsymbol{z}_0$ is a local and also global minimizer of f.*

[3] A convex function has a unique local minimum—which is also the global minimum, but could have multiple local minimizers that are also global minimizers.

There is a more refined characterization of local minimizers using both gradient and Hessian.

Theorem 2.4 (Second-order necessary condition of optimality for unconstrained problems)**.** *Assume f is second-order differentiable at $\boldsymbol{z}_0$. If $\boldsymbol{z}_0$ is a local minimizer, then $\nabla f(\boldsymbol{z}_0) = \mathbf{0}$, and $\nabla^2 f(\boldsymbol{z}_0) \succeq \mathbf{0}$, i.e., Hessian at $\boldsymbol{z}_0$ is positive semidefinite.*

A point $\boldsymbol{z}_0$ satisfying $\nabla f(\boldsymbol{z}_0) = \mathbf{0}$ and $\nabla^2 f(\boldsymbol{z}_0) \succeq \mathbf{0}$ is called a *second-order stationary point*, or SOSP. Similarly to a FOSP, a SOSP can be a local minimizer, a local maximizer, or a saddle point. A stronger condition can ensure a local minimizer.

Theorem 2.5 (Second-order sufficient condition of optimality for unconstrained problems)**.** *Assume f is second-order differentiable at $\boldsymbol{z}_0$. If $\nabla f(\boldsymbol{z}_0) = \mathbf{0}$, and $\nabla^2 f(\boldsymbol{z}_0) \succ \mathbf{0}$, i.e., Hessian at $\boldsymbol{z}_0$ is positive definite, then $\boldsymbol{z}_0$ is a local minimizer.*

Note that the gap between the second-order sufficient and necessary conditions lies in Hessian: $\nabla^2 f(\boldsymbol{z}_0) \succ \mathbf{0}$ vs. $\nabla^2 f(\boldsymbol{z}_0) \succeq \mathbf{0}$. When $\nabla f(\boldsymbol{z}_0) = \mathbf{0}$ and $\nabla^2 f(\boldsymbol{z}_0) \succeq \mathbf{0}$, the local landscape of the function can be shaped by higher-order derivatives, no matter how local. For example, $(0, 0)$ is an SOSP of the function $h(x, y) = x^3 - y^3$, but it is a saddle point, as there is a local descent direction of the function value.

For our problem Eq. (2.8), the least squares objective is a quadratic polynomial and is hence second-order differentiable. The Hessian is $2\boldsymbol{X}^\intercal \boldsymbol{X}$, which is positive semidefinite. So $g(\boldsymbol{w})$ is a convex function.

Invoking Theorem 2.3, we have

$$\nabla g(\boldsymbol{w}_0) = 2\boldsymbol{X}^\intercal(\boldsymbol{X}\boldsymbol{w}_0 - \boldsymbol{y}) = \mathbf{0} \Longrightarrow \boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{w}_0 = \boldsymbol{X}^\intercal \boldsymbol{y}. \tag{2.11}$$

- If $\boldsymbol{X} \in \mathbb{R}^{N \times (d+1)}$ has full column rank, i.e., with linearly independent columns, or equivalently the N data points in $\boldsymbol{X}$ span the $(d+1)$-dimensional input space, then $\boldsymbol{X}^\intercal \boldsymbol{X}$ has full rank and hence is invertible. Then

$$\boldsymbol{w}_0 = (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}. \tag{2.12}$$

  Obviously, $\boldsymbol{w}_0$ is uniquely defined by the right side of Eq. (2.12) and so is the unique global minimizer of $g(\boldsymbol{w})$.

- Otherwise, $\boldsymbol{X}^\intercal \boldsymbol{X}$ is not invertible and there are multiple (in fact, infinitely many) global minimizers. A particular global minimizer can be found through the *pseudo-inverse*. Recall that for any matrix $\boldsymbol{M} \in \mathbb{R}^{m \times n}$ of rank $r \le \min(m, n)$, its *compact SVD* can be written as $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal$ where $\boldsymbol{U} \in \mathbb{R}^{m \times r}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{r \times r}$ is diagonal, and $\boldsymbol{V} \in \mathbb{R}^{n \times r}$. The pseudo-inverse of $\boldsymbol{M}$ is then $\boldsymbol{M}^\dagger = \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^\intercal \in \mathbb{R}^{n \times m}$. Obviously, pseudo-inverse is defined for any matrix, square or not. A solution to Eq. (2.11), and hence a global minimizer to $g(\boldsymbol{w})$, is then

$$\boldsymbol{w}_0 = (\boldsymbol{X}^\intercal \boldsymbol{X})^\dagger \boldsymbol{X}^\intercal \boldsymbol{y} = \boldsymbol{X}^\dagger \boldsymbol{y}, \tag{2.13}$$

  where at the last equality we have used the fact $(\boldsymbol{X}^\intercal \boldsymbol{X})^\dagger \boldsymbol{X}^\intercal = \boldsymbol{X}^\dagger$[4]. For a square invertible matrix, its pseudo-inverse coincides with its inverse. So, in fact, this provides a generic form of global minimizer to $g(\boldsymbol{w})$, whether $\boldsymbol{X}^\intercal \boldsymbol{X}$ is invertible or not. The set of all global minimizers is $\boldsymbol{w}_0 + \mathrm{null}(\boldsymbol{X}) = \{\boldsymbol{X}^\dagger \boldsymbol{y} + \boldsymbol{z} : \boldsymbol{z} \in \mathrm{null}(\boldsymbol{X})\}$.

The closed-form solution we get here seems nice, but probably only for small-scale problems. The cost of calculating $\boldsymbol{X}^\intercal \boldsymbol{X}$ is $O(d^2 N)$, and inverting $\boldsymbol{X}^\intercal \boldsymbol{X} \in \mathbb{R}^{(d+1)\times(d+1)}$ costs $O(d^3)$. This is daunting when d and N are large. Can we do better?

[4]To see it, let the compact SVD of $\boldsymbol{X}$ be $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal$. We have $(\boldsymbol{X}^\intercal \boldsymbol{X})^\dagger \boldsymbol{X}^\intercal = (\boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^\intercal \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal)^\dagger \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^\intercal = (\boldsymbol{V}\boldsymbol{\Sigma}^2\boldsymbol{V}^\intercal)^\dagger \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^\intercal = \boldsymbol{V}\boldsymbol{\Sigma}^{-2}\boldsymbol{V}^\intercal \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^\intercal = \boldsymbol{V}\boldsymbol{\Sigma}^{-2}\boldsymbol{\Sigma}\boldsymbol{U}^\intercal = \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^\intercal = \boldsymbol{X}^\dagger$.

2.4 Solution via iterative optimization

In numerical optimization, iterative methods start with an initial guess (i.e., *initialization*) and produce a sequence of points (i.e., *iterates*) that gradually approach a potential solution. Gradient descent (GD) is a basic but powerful iterative method. For an unconstrained problem $\min_{\boldsymbol{z} \in \mathbb{R}^n} \ f(\boldsymbol{z})$, GD runs like this:

Algorithm 1 Gradient descent for minimizing $f(\boldsymbol{z})$

Input: initialization $\boldsymbol{z}^{(0)}$, $k = 0$, stopping precision $\varepsilon > 0$

while $\|\nabla f(\boldsymbol{z}^{(k)})\|_2 > \varepsilon$ **do**
  choose a step size $t^{(k)}$
  update the estimate: $\boldsymbol{z}^{(k+1)} = \boldsymbol{z}^{(k)} - t^{(k)} \nabla f(\boldsymbol{z}^{(k)})$
  update the counter: $k = k + 1$
end while

When we apply GD to our least-squares problem, the gradient update step is:

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - t^{(k)} \nabla g(\boldsymbol{w}^{(k)}) = \boldsymbol{w}^{(k)} - 2t^{(k)} \boldsymbol{X}^{\intercal}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}^{(k)}), \tag{2.14}$$

which costs $O(dN)$. So, the cost for each iteration is $O(dN)$, and the total cost is $O(dNT)$ if T is the total number of iterations taken to find an approximate minimizer. When $T \ll \min(d, N)$, GD is computationally favorable for our least-squares problem compared to computing the solution using the closed-form formula Eq. (2.12) or Eq. (2.13).

Now, let us think about three basic questions about GD.

- **Why move in the negative gradient direction?** Intuitively, to find a local minimizer, one hopes to construct a sequence of iterates with monotonically decreasing function values. Suppose that our current iterate is $\boldsymbol{z}$. If we make a small movement $t\boldsymbol{d}$ from $\boldsymbol{z}$—where $\boldsymbol{d}$ is the *direction of movement* and t is the step size that we can adjust as desired to control the overall *magnitude of movement* (i.e., magnitude of $t\boldsymbol{d}$), Taylor's theorem says when $t\boldsymbol{d}$ is small,

$$f(\boldsymbol{z} + t\boldsymbol{d}) \approx f(\boldsymbol{z}) + t \langle \nabla f(\boldsymbol{z}), \boldsymbol{d} \rangle \Longrightarrow f(\boldsymbol{z} + \boldsymbol{d}) - f(\boldsymbol{z}) \approx t \langle \nabla f(\boldsymbol{z}), \boldsymbol{d} \rangle . \tag{2.15}$$

  We hope to make $f(\boldsymbol{z} + \boldsymbol{d}) - f(\boldsymbol{z})$ as negative as possible to make rapid progress toward a local minimizer, so we can try to make $t \langle \nabla f(\boldsymbol{z}), \boldsymbol{d} \rangle$ as negative as possible. Now for any fixed t, we want to minimize $\langle \nabla f(\boldsymbol{z}), \boldsymbol{d} \rangle$. Recall that $\boldsymbol{d}$ is only a direction, so it makes sense to restrict its norm to avoid trivial solutions—easy to see that so long as $\nabla f(\boldsymbol{z}) \neq \boldsymbol{0}$, we can make $\langle \nabla f(\boldsymbol{z}), \boldsymbol{d} \rangle$ approach $-\infty$. A natural choice is $\|\boldsymbol{d}\|_2 = 1$, leading to

$$\min_{\boldsymbol{d}:\, \|\boldsymbol{d}\|_2 = 1} \langle \nabla f(\boldsymbol{z}), \boldsymbol{d} \rangle , \tag{2.16}$$

  whose solution is $\boldsymbol{d}_* = -\frac{\nabla f(\boldsymbol{z})}{\|\nabla f(\boldsymbol{z})\|_2}$ as long as $\nabla f(\boldsymbol{z}) \neq \boldsymbol{0}$ (Note that choosing other norms will lead to different directions). This is where the $-\nabla f(\boldsymbol{z})$ direction in GD comes from.

- **Which step size?** For step sizes $t^{(k)}$'s, one one hand, we hope to make them as large as possible to allow fast progress. On the other, they should be reasonably small to make the first-order Taylor approximation in Eq. (2.15) reasonably accurate to guarantee the descent of the function value. There are two popular strategies for choosing the step sizes:

  - **Fixed step size** Choose a sufficiently small constant as the step size for all iterations. If the value is not sufficiently small, typically the objective value will blow up after a while. **Pros**: simple; **Cons**: could be conservative for most iterations.

– **Adaptive step size via backtracking line search** Search for an appropriate (large) step size adapted to local landscape of the function.

Algorithm 2 Gradient descent for minimizing $f(\boldsymbol{z})$ with backtracking line search

Input: initialization $\boldsymbol{z}^{(0)}$, $k = 0$, stopping precision $\varepsilon > 0$ close to 0

while $\|\nabla f(\boldsymbol{z}^{(k)})\|_2 > \varepsilon$ **do**
    choose initial step size $t = 1$, $\rho \in (0, 1)$, and $\eta \in (0, 1)$
    while $f(\boldsymbol{z}^{(k)} - t\nabla f(\boldsymbol{z}^{(k)})) - f(\boldsymbol{z}^{(k)}) > -\eta t \|\nabla f(\boldsymbol{z}^{(k)})\|_2^2$ **do**
        decrease the step size: $t = \rho t$
    end while
    set the step size: $t^{(k)} = t$
    update the estimate: $\boldsymbol{z}^{(k+1)} = \boldsymbol{z}^{(k)} - t^{(k)} \nabla f(\boldsymbol{z}^{(k)})$
    update the counter: $k = k + 1$
end while

Pros: relatively large step size, and hence fast movement and rapid convergence
Cons: slightly more computation each iteration for searching the good step size

The backtracking line-search strategy is highly recommended for practical implementation of GD.

To see the intuition behind the backtracking line search strategy, suppose that the current iterate is $\boldsymbol{z}$ and so the gradient is $\nabla f(\boldsymbol{z})$. We want to choose a step size t so that $f(\boldsymbol{z} - t\nabla f(\boldsymbol{z})) - f(\boldsymbol{z})$ is as negative as possible to quickly minimize the objective. Since we assume that f is first-order differentiable, Taylor's theorem tells us

$$f(\boldsymbol{z} - t\nabla f(\boldsymbol{z})) - f(\boldsymbol{z}) = -t\|\nabla f(\boldsymbol{z})\|_2^2 + o(t\|\nabla f(\boldsymbol{z})\|_2) \tag{2.17}$$

as $t \to 0$. Now, the linear term $-t\|\nabla f(\boldsymbol{z})\|_2^2$ is negative, and the lower-order term $o(t\|\nabla f(\boldsymbol{z})\|_2)$ may be positive or negative. In any case, when $t > 0$ is sufficiently small, $-t\|\nabla f(\boldsymbol{z})\|_2^2$ will dominate the right side of Eq. (2.17), and we can reach a level so that

$$-t\|\nabla f(\boldsymbol{z})\|_2^2 + o(t\|\nabla f(\boldsymbol{z})\|_2) \le -\eta t \|\nabla f(\boldsymbol{z})\|_2^2 \tag{2.18}$$

for a pre-fixed $\eta \in (0, 1)$. Of course, any smaller t still satisfies this. But since we hope to set t to be the largest possible, our line search is backward: we start with a large t and gradually decrease it whenever Eq. (2.18) is violated.

- **When to stop?** Since $\nabla f(\boldsymbol{z}) = \mathbf{0}$ is the necessary condition for $\boldsymbol{z}$ being a local minimizer, we set the stopping criterion as checking if $\|\nabla f(\boldsymbol{z})\|_2$ is sufficiently close to 0. Another possibility is to check the increment of the function value—stopping when there is not much progress, e.g., $|f(\boldsymbol{z}^{(k+1)}) - f(\boldsymbol{z}^{(k)})|$, which tends to be less reliable.

2.5 Popular variants of linear regression

When $\boldsymbol{X}^\intercal \boldsymbol{X}$ is not invertible, there are infinitely many global minimizers to our least-squares problem. In particular, this happens when $N < d + 1$, i.e., the number of data points is smaller than the input dimension. In statistics, this belongs to the family of the so-called *high-dimensional problems* where typically a regularization term is added.

- **Ridge regression** takes the form

$$\min_{\boldsymbol{w}} \ \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda \|\boldsymbol{w}\|_2^2 \tag{2.19}$$

for a certain $\lambda > 0$. Since our linear model is $y \approx \langle \boldsymbol{w}, \boldsymbol{x} \rangle = \sum_j w_j x_j$, regularizing $\|\boldsymbol{w}\|_2^2$ ensures that the entries in $\boldsymbol{w}$ are all reasonably small, so that any change in the input $\boldsymbol{x}$ only causes a small change in the predicted value. In other words, the learned model is stable. Since its Hessian $2(\boldsymbol{X}^\intercal \boldsymbol{X} + \lambda \boldsymbol{I}) \succeq \boldsymbol{0}$ everywhere, the objective is convex. In fact, $2(\boldsymbol{X}^\intercal \boldsymbol{X} + \lambda \boldsymbol{I}) \succeq 2\lambda \boldsymbol{I}$[5] and so the objective is *strongly* convex. Thus, ridge regression has a unique global minimizer.

- **Lasso** takes the form

$$\min_{\boldsymbol{w}} \ \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda \|\boldsymbol{w}\|_1 \tag{2.20}$$

for a certain $\lambda > 0$. Compared to $\|\boldsymbol{w}\|_2^2 = \sum_j w_j^2$, $\|\boldsymbol{w}\|_1 = \sum_j |w_j|$ penalizes large entries in $\boldsymbol{w}$ much less and small entries much more. The net effect is that regularizing using $\|\boldsymbol{w}\|_1$ tends to produce a solution that is sparse—containing very few large entries and the rest negligible in magnitude. This is useful when one is interested in selecting only a few most important features (i.e., columns) from $\boldsymbol{X}$. Both $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2$ and $\lambda \|\boldsymbol{w}\|_1$ are convex, and so the positive combination is convex. However, in general, Lasso does not have a unique global minimizer either. **Elastic net**

$$\min_{\boldsymbol{w}} \ \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda_1 \|\boldsymbol{w}\|_1 + \lambda_2 \|\boldsymbol{w}\|_2^2 \tag{2.21}$$

which integrates Lasso and ridge regression is a fix to this and has a unique global minimizer, alongside other benefits over Lasso, such as stability when selecting correlated features.

Comparisons of various popular models for linear regression and classification can be found at `https://scikit-learn.org/stable/modules/linear_model.html`.

3 Review of subspaces and hyperplanes

Consider a line L in $\mathbb{R}^2$, as illustrated in Fig. 1 (left).

- If L passes through the origin, there are two ways to represent L. One way is to find a vector $\boldsymbol{v} \in \mathbb{R}^2$ aligned with L, then $L = \{\lambda \boldsymbol{v} : \lambda \in \mathbb{R}\}$. This is called *basis representation*. The other way is to find a vector $\boldsymbol{w}$ that is orthogonal (i.e., normal) to L, i.e., orthogonal to all vectors in L, and then $L = \{\boldsymbol{x} \in \mathbb{R}^2 : \langle \boldsymbol{w}, \boldsymbol{x} \rangle = 0\}$, called *normal representation*.

- If L does not pass through the origin, we can find an arbitrary point $\boldsymbol{x}_0 \in L$ and write $L = \boldsymbol{x}_0 + L' = \{\boldsymbol{x}_0 + \boldsymbol{x} : \boldsymbol{x} \in L'\}$ for an L' that is parallel to L and passes through the origin. For any basis vector $\boldsymbol{v}$ and any normal vector $\boldsymbol{w}$ for L', we can represent L as

$$L = \underbrace{\{\boldsymbol{x}_0 + \lambda \boldsymbol{v} : \lambda \in \mathbb{R}\}}_{\text{basis representation}} = \underbrace{\left\{\boldsymbol{x} \in \mathbb{R}^2 : \langle \boldsymbol{w}, \boldsymbol{x} \rangle = \langle \boldsymbol{w}, \boldsymbol{x}_0 \rangle\right\}}_{\text{normal representation}}, \tag{3.1}$$

[5] For two symmetric matrices $\boldsymbol{M}_1, \boldsymbol{M}_2 \in \mathbb{R}^{n \times n}$, $\boldsymbol{M}_1 \succeq \boldsymbol{M}_2$ means $\boldsymbol{M}_1 - \boldsymbol{M}_2 \succeq \boldsymbol{0}$.

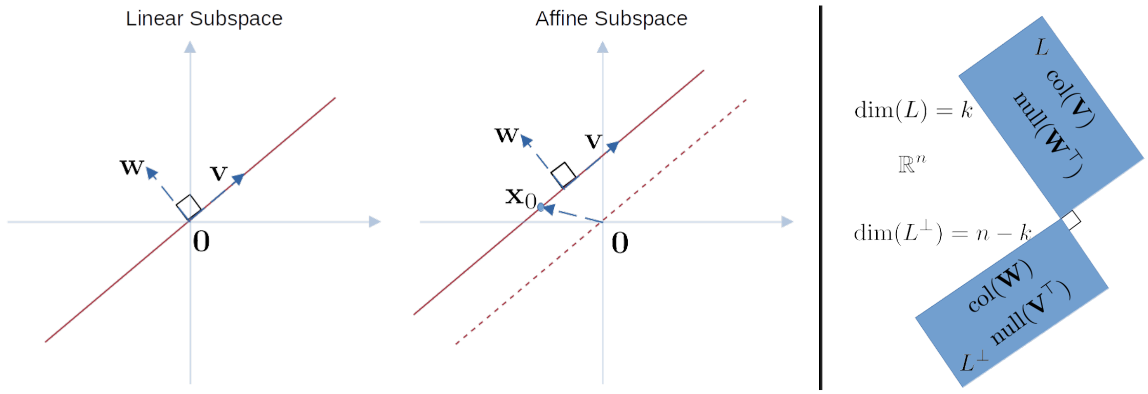

Figure 1: (left) Illustration of subspaces and hyperplanes; (right) Geometric picture of basis and normal representations. The picture is adapted from Sec 4.1 of the famous linear algebra book [Str16].

where to derive the normal representation, we know that for any $\boldsymbol{x} \in L$, $\boldsymbol{x} - \boldsymbol{x}_0 \in L' \iff \langle \boldsymbol{w}, \boldsymbol{x} - \boldsymbol{x}_0 \rangle = 0$.

Now we generalize these to subspaces. Recall that a set $S \subset \mathbb{R}^n$ is called a subspace of $\mathbb{R}^n$ if and only if for all $\boldsymbol{u}, \boldsymbol{v} \in S$ and all $\alpha, \beta \in \mathbb{R}$, $\alpha \boldsymbol{u} + \beta \boldsymbol{v} \in S$, i.e., all linear combinations of elements in S stay in S. Geometrically, subspaces can be thought of as high-dimensional "flats" in $\mathbb{R}^n$, and they are natural generalizations of lines. Subspaces also admit both basis and normal representations that generalize the corresponding representation for lines: for any k-dimensional subspace $L \subset \mathbb{R}^n$,

- **basis representation**: for any k linearly independent vectors $\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_k\}$ that span L, i.e., $\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_k\}$ is a basis for L,

$$L = \left\{ \sum_{i=1}^{k} \alpha_i \boldsymbol{v}_i : \alpha_i \in \mathbb{R} \ \forall\, i \right\} = \left\{ \boldsymbol{V}\boldsymbol{\alpha} : \boldsymbol{\alpha} \in \mathbb{R}^k \right\} = \mathrm{col}(\boldsymbol{V}), \tag{3.2}$$

  where $\boldsymbol{V} \doteq [\boldsymbol{v}_1 \ \dots \ \boldsymbol{v}_k] \in \mathbb{R}^{n \times k}$ and $\mathrm{col}(\cdot)$ indicates the column space.

- **normal representation**: for any $n-k$ linearly independent vectors $\{\boldsymbol{w}_1, \dots, \boldsymbol{w}_{n-k}\}$ that are orthogonal to L, i.e., $\langle \boldsymbol{w}_j, \boldsymbol{x} \rangle = 0$ for all $j \in [n-k]$ and all $\boldsymbol{x} \in L$,

$$L = \{\boldsymbol{x} \in \mathbb{R}^n : \langle \boldsymbol{x}, \boldsymbol{w}_j \rangle = 0 \ \forall j \in [n-k]\} = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{W}^\intercal \boldsymbol{x} = \boldsymbol{0}\} = \mathrm{null}(\boldsymbol{W}^\intercal), \tag{3.3}$$

  where $\boldsymbol{W} \doteq [\boldsymbol{w}_1 \ \dots \ \boldsymbol{w}_{n-k}] \in \mathbb{R}^{n \times (n-k)}$ and $\mathrm{null}(\cdot)$ denotes the null space. Moreover, $\boldsymbol{W}$ spans the unique $(n-k)$-dimensional orthogonal subspace $L^\perp$ of L. [6]

The geometric aspect of the discussion is summarized in Fig. 1 (right).

When the subspace $L \subset \mathbb{R}^n$ has dimension $n-1$, it deserves a special name—*hyperplane*, which is a critical element of machine learning; in the next section, we need this object for linear classification.

All subspaces that we speak of contain the origin; if we want to emphasize this fact, we prefix the adjective *linear*, i.e., calling them *linear subspaces*. This is also to distinguish them with flats that do not necessarily pass through the origin—as generalization of lines that do not necessarily; we call these flats *affine subspaces*.

[6]Two subspaces L and L' are said to be orthogonal to each other if and only if $\boldsymbol{z}$ and $\boldsymbol{z}'$ are orthogonal to each other for all $\boldsymbol{z} \in L$ and $\boldsymbol{z}' \in L'$.

Similarly to the way we represent "affine" lines, we can think of any affine subspace L as a shifted linear subspace, i.e., $L = \boldsymbol{x}_0 + L'$ for certain $\boldsymbol{x}_0 \in L$ and a "parallel" linear subspace L'. This implies the following basis and normal representations for L:

- **basis representation**: assume that $\boldsymbol{V} \in \mathbb{R}^{n\times k}$ spans L' and $\boldsymbol{x}_0 \in L$, then

$$L = \{\boldsymbol{x}_0 + \boldsymbol{x} : \boldsymbol{x} \in L'\} = \left\{\boldsymbol{x}_0 + \boldsymbol{V}\boldsymbol{\alpha} : \boldsymbol{\alpha} \in \mathbb{R}^k\right\} = \boldsymbol{x}_0 + \mathrm{col}(\boldsymbol{V}). \tag{3.4}$$

  Moreover, $\dim(L) = \dim(L') = k$.

- **normal representation**: assume $\boldsymbol{W}$ spans $(L')^\perp$ and $\boldsymbol{x}_0 \in L$, then

$$L = \{\boldsymbol{x}_0 + \boldsymbol{x} : \boldsymbol{W}^\intercal \boldsymbol{x} = \boldsymbol{0}\} = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{W}^\intercal \boldsymbol{x} = \boldsymbol{W}^\intercal \boldsymbol{x}_0\} = \boldsymbol{x}_0 + \mathrm{null}(\boldsymbol{W}^\intercal). \tag{3.5}$$

A natural question is whether $\boldsymbol{W}^\intercal \boldsymbol{x}_0$ is unique given that $\boldsymbol{x}_0$ is an arbitrary point on L. The answer is yes, as for any two points $\boldsymbol{x}_0, \boldsymbol{x}_0' \in L$, $\boldsymbol{W}^\intercal\boldsymbol{x}_0 - \boldsymbol{W}^\intercal\boldsymbol{x}_0' = \boldsymbol{W}^\intercal(\boldsymbol{x}_0 - \boldsymbol{x}_0') = \boldsymbol{W}^\intercal\boldsymbol{V}\boldsymbol{\alpha}$ for a certain $\boldsymbol{\alpha} \in \mathbb{R}^{n-k}$. But $\boldsymbol{W}^\intercal\boldsymbol{V} = \boldsymbol{0}$, implying that $\boldsymbol{W}^\intercal\boldsymbol{V}\boldsymbol{\alpha} = \boldsymbol{0}$ whatever the $\boldsymbol{\alpha}$ is.

Table 1: Summary of representations for linear and affine subspaces

	basis representation	normal representation
linear subspace L with basis $\boldsymbol{V}$ and normal basis $\boldsymbol{W}$ $(\dim(L) = k)$	$\{\boldsymbol{V}\boldsymbol{\alpha} : \boldsymbol{\alpha} \in \mathbb{R}^k\} = \mathrm{col}(\boldsymbol{V})$	$\{\boldsymbol{x} : \boldsymbol{W}^\intercal\boldsymbol{x} = \boldsymbol{0}\} = \mathrm{null}(\boldsymbol{W}^\intercal)$
affine subspace L with basis $\boldsymbol{V}$, point $\boldsymbol{x}_0 \in L$, and normal basis $\boldsymbol{W}$ $(\dim(L) = k)$	$\left\{\boldsymbol{x}_0 + \boldsymbol{V}\boldsymbol{\alpha} : \boldsymbol{\alpha} \in \mathbb{R}^k\right\} = \boldsymbol{x}_0 + \mathrm{col}(\boldsymbol{V})$	$\{\boldsymbol{x} : \boldsymbol{W}^\intercal\boldsymbol{x} = \boldsymbol{W}^\intercal\boldsymbol{x}_0\} = \boldsymbol{x}_0 + \mathrm{null}(\boldsymbol{W}^\intercal)$

As expected, *affine hyperplanes* are affine subspaces with dimensions one less than the ambient dimension. Of special interest in this case is the normal representation

$$\{\boldsymbol{x} \in \mathbb{R}^n : \langle \boldsymbol{w}, \boldsymbol{x}\rangle = \langle \boldsymbol{w}, \boldsymbol{x}_0\rangle\}. \tag{3.6}$$

Of course, linear hyperplanes take the form $\{\boldsymbol{x} \in \mathbb{R}^n : \langle \boldsymbol{w}, \boldsymbol{x}\rangle = \boldsymbol{0}\}$.

Obviously, an affine subspace can be a linear subspace in our definition. So henceforth, *subspaces are defaulted to affine subspaces, unless the word "linear" is appended; similarly for hyperplanes*. We are now ready to study linear classification.

4 Linear classification

We focus on binary classification with a training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{1, -1\}$ for $i \in [N]$. To fit the training set, the first idea is to use a linear function to map any input $\boldsymbol{x}$ to 1 or -1 as we do in linear regression. But this is unrealistic, as the output range of any linear function is a continuum, not a discrete set such as $\{1, -1\}$. Below, we describe two distinct approaches to dealing with the difficulty.

4.1 Approach I: Preceptron and linear SVM

Recall that the $\mathrm{sign}(\cdot)$ function takes value in $\{1, -1\}$. [7] So, we can consider the hypothesis class

$$\mathcal{H}_I = \left\{\boldsymbol{x} \mapsto \mathrm{sign}\left(\langle \boldsymbol{w}, \boldsymbol{x}\rangle + b\right) : \boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R}\right\}, \tag{4.1}$$

[7] We define the sign function as $\mathrm{sign}(z) = \begin{cases} 1 & z > 0 \\ -1 & z \le 0 \end{cases}$.

which is the set of all hyperplane classifiers (each separates the whole space into two half spaces).

A natural problem formulation is then to find a pair $(\boldsymbol{w}_0, b_0)$ so that

$$\operatorname{sign}\left(\langle \boldsymbol{w}_0, \boldsymbol{x}_i \rangle + b_0\right) = y_i \Longleftrightarrow y_i(\langle \boldsymbol{w}_0, \boldsymbol{x}_i \rangle + b_0) > 0 \;\forall\; i \in [N]. \tag{4.2}$$

Note that in this formulation, we do not perform minimization, but instead try to find a feasible solution to a set of constraints. This kind of problem is known as *feasibility problem*[8] in optimization. For convenience, we use the homogeneous representation again with abuse of the notation, and the goal is to:

$$\text{find } \boldsymbol{w} \in \mathbb{R}^{d+1} \;\text{ s.t. }\; y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle > 0 \;\forall\; i \in [N]. \tag{4.3}$$

Moreover, the training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ is said to be *linearly separable* if there exists a $\boldsymbol{w}_0 \in \mathbb{R}^{d+1}$ that solves problem (4.3), i.e., satisfies $y_i \langle \boldsymbol{w}_0, \boldsymbol{x}_i \rangle > 0$ for all $i \in [N]$.

4.1.1 A classical solution: Perceptron

Perceptron is a classical algorithm designed to solve problem (4.3). Invented by Frank Rosenblatt in 1958, it helped to fuel the first wave of excitement about neural networks around 60's, but later on it also helped kill the excitement and cause major setbacks for neural networks research due to the famous 1969 book [MM17] that elucidates the limitations of Perceptron. However, Perceptron is a critical milestone in the development of binary classification and online learning algorithms. We describe only the binary classification aspect here.

Algorithm 3 The Perceptron algorithm for binary classification

Input: training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, initialization $\boldsymbol{w}^{(0)} = \mathbf{0}$, $k = 0$

1: **while** $\exists i$ s. t. $y_i\langle \boldsymbol{w}^{(k)}, \boldsymbol{x}_i \rangle \le 0$ **do**
2: update the estimate: $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + y_i \boldsymbol{x}_i$
3: update the counter: $k = k + 1$
4: **end while**

To get a sense why the update step is sensible, note that for an $i \in N$ with $y_i\langle \boldsymbol{w}^{(k-1)}, \boldsymbol{x}_i \rangle \le 0$, after the update,

$$y_i \left\langle \boldsymbol{w}^{(k+1)}, \boldsymbol{x}_i \right\rangle = y_i \left\langle \boldsymbol{w}^{(k)} + y_i \boldsymbol{x}_i, \boldsymbol{x}_i \right\rangle = y_i \left\langle \boldsymbol{w}^{(k)}, \boldsymbol{x}_i \right\rangle + \|\boldsymbol{x}_i\|_2^2 > y_i \left\langle \boldsymbol{w}^{(k)}, \boldsymbol{x}_i \right\rangle, \tag{4.4}$$

i.e., the value moves toward positive and we are making progress. Here, we assume $\|\boldsymbol{x}_i\|_2 > 0$ for all $i \in [N]$. The convergence behavior of the Perceptron algorithm is captured by the following theorem.

Theorem 4.1. *Assume that the training set* $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ *is linearly separable and define* data radius $R \doteq \max_{i \in [N]} \|\boldsymbol{x}_i\|_2$ *and* margin parameter[9] $M = \min\{\|\boldsymbol{w}\|_2 : y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \ge 1 \;\forall i \in [N]\}$. *Then, the Perceptron algorithm will take at most* $(RM)^2$ *steps to find a feasible* $\boldsymbol{w}$*, when it stops.*

[8]To unify them with minimization problems, we can treat them as minimization with a constant objective function.
[9]This will become clear when we talk about the max-margin aspect of SVMs.

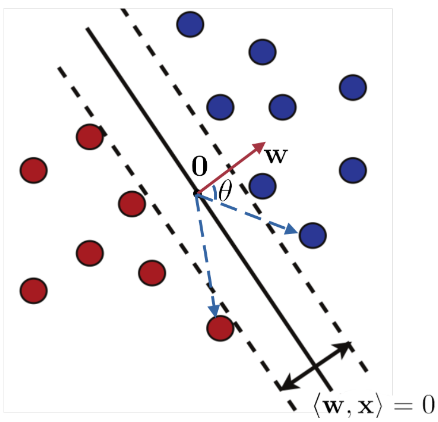

Figure 2: Separating hyperplane in binary classification and interpretation of the Perceptron convergence theorem.

Proof of this theorem is in Appendix A. The radius R here is not important, as we can always rescale all our $\boldsymbol{x}_i$'s to make $R = 1$. Interpreting M is slightly more tricky. The condition for M is $y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \geq 1$ for all $i \in [N]$, which requires not only $\boldsymbol{w}$ is feasible for problem (4.3), but also $|\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle| \geq 1$ for all $i \in [N]$, i.e.,

$$|\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle| = \|\boldsymbol{w}\|_2 \|\boldsymbol{x}_i\|_2 |\cos \angle(\boldsymbol{w}, \boldsymbol{x}_i)| \geq 1 \quad \forall\, i \in [N]. \tag{4.5}$$

Assume that $\boldsymbol{x}_i$'s are comparable in magnitude. For all i, the smaller the $|\cos \angle(\boldsymbol{w}, \boldsymbol{x}_i)|$ or the closer the angle $\angle(\boldsymbol{w}, \boldsymbol{x}_i)$ near $90°$, the larger the $\|\boldsymbol{w}\|_2$ needed to guarantee that Eq. (4.5) holds. In view of Fig. 2, when the positive and negative classes get closer and harder to separate, $|\cos \angle(\boldsymbol{w}, \boldsymbol{x}_i)|$ can be arbitrarily small for certain i, leading to an M—which is the measure of the linear separability of the training set—that can be exponentially large in dimension d.

So there are at least two limitations of Perceptron: 1) it cannot deal with linearly non-separable training data, and the algorithm will not even stop in those scenarios; 2) even if the training data are separable, the running time can be exponential in the worst case.

4.2 A modern solution: linear SVM

To address the limitations of the Perceptron algorithm, a solution is to reformulate problem (4.3) into a form that is amenable to numerical optimization methods.

There are two issues with the constraints in problem (4.3): (1) for any solution w_0, $\lambda \boldsymbol{w}_0$ is also a solution for all $\lambda > 0$. Although these solutions are equally good mathematically, they are not equally favored numerically: we want to avoid exceedingly large and small numbers in typical numerical computation to prevent overflows and underflows; (2) iterative methods often work with *compact* constraint sets so that the convergence limit remains in the set and convergence could be established. Strict inequalities in the constraints could lead to noncompact constraint sets.

First of all, to make the constraint set compact, note that for any $\boldsymbol{w}_0$ satisfying $\langle \boldsymbol{w}_0, \boldsymbol{x}_i \rangle > 0\ \forall\ i \in [N]$, there exists an $\eta > 0$ so that $\langle \boldsymbol{w}_0, \boldsymbol{x}_i \rangle \geq \eta\ \forall\ i \in N$. So there exists a $\lambda > 0$ so that $\langle \lambda \boldsymbol{w}_0, \boldsymbol{x}_i \rangle \geq 1\ \forall\, i \in [N]$. So problem (4.3) is equivalent to

$$\text{find } \boldsymbol{w} \in \mathbb{R}^{d+1} \ \text{ s. t. } \ y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \geq 1\ \forall\ i \in [N]. \tag{4.6}$$

For any feasible $\boldsymbol{w}_0$ for problem (4.6), obviously $\lambda \boldsymbol{w}_0$ for all $\lambda > 1$ is also feasible. We can further refine the formulation by controlling the magnitude of $\boldsymbol{w}$, say using

$$\min_{\boldsymbol{w}} \ \|\boldsymbol{w}\|_2^2 \ \text{ s. t. } \ y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \geq 1\ \forall\ i \in [N]. \tag{4.7}$$

This is the homogeneous form of *hard-margin SVM* that we will discuss later. Obviously, the ℓ_2 norm we use here is arbitrary; in principle, any function that is monotonically increasing in the "magnitude" of $\boldsymbol{w}$ can be used, e.g., all vector norms.

When the training set is not linearly separable, there is no feasible solution for problem (4.6). One can add in controlled slackness to allow slight constraint violation, e.g., via

$$\min_{\boldsymbol{w}} \ \|\boldsymbol{w}\|_2^2 + C \sum_{i=1}^{N} \xi_i \ \text{ s. t. } \ y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \geq 1 - \xi_i, \xi_i \geq 0\ \forall\ i \in [N]. \tag{4.8}$$

Here, the changed lower bounds $1 - \xi_i$ in the constraints introduce slackness, and the term $\sum_{i=1}^{N} \xi_i$ in the objective controls the size of the slackness. This is the homogeneous form of *soft-margin SVM*.

Both problems in Eqs. (4.7) and (4.8) are convex (quadratic) optimization problems and can be solved efficiently.

4.3 Approach II: Logistic regression

We consider the hypothesis class

$$\mathcal{H}_{II} = \left\{ \boldsymbol{x} \mapsto 2\phi(\boldsymbol{w}^\intercal \boldsymbol{x}) - 1 : \boldsymbol{w} \in \mathbb{R}^{d+1} \right\}, \tag{4.9}$$

where $\phi(\cdot)$ denotes the *logistic function* (i.e., *sigmoid*; see Fig. 3)

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{4.10}$$

that maps any input into the $[0, 1]$ interval. Now $\psi(z) \doteq 2\phi(z) - 1$ maps any input $\boldsymbol{z}$ into the $[-1, 1]$ interval, and also $\psi(z) \to 1$ when $z \to \infty$ and $\psi(z) \to -1$ when $z \to -\infty$. Moreover, since $\psi(z)$ is an approximation to the sign function (see Fig. 3), we can view $\mathcal{H}_{II}$ as an approximation to

$$\mathcal{H}_{I} = \left\{ \boldsymbol{x} \mapsto \operatorname{sign}(\boldsymbol{w}^\intercal \boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^{d+1} \right\}, \tag{4.11}$$

restated in the homogeneous form. After we find an appropriate function from $\mathcal{H}_{II}$, say parametrized by a certain $\boldsymbol{w}_0$, we can heuristically compose that function with the $\operatorname{sign}(\cdot)$ function to obtain a predictor $\operatorname{sign}(2\phi(\boldsymbol{w}_0^\intercal \boldsymbol{x}) - 1)$ that outputs from the discrete set $\{1, -1\}$ as desired.

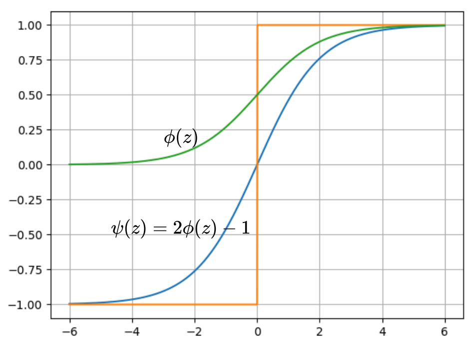

Figure 3: Graphs of the logistic function $\phi(z)$ (green), $\psi(z)$ (blue), and the sign function (orange)

Now we come to formulating the problem based on hypothesis class $\mathcal{H}_{II}$. To fit our training set as much as possible, we hope that

$$\text{when } y = 1: \quad 2\phi(\boldsymbol{w}^\intercal \boldsymbol{x}) - 1 = \frac{2}{1 + e^{-\boldsymbol{w}^\intercal \boldsymbol{x}}} - 1 \to 1 \text{ i.e., to be maximized,} \tag{4.12}$$

$$\text{when } y = -1: \quad 2\phi(\boldsymbol{w}^\intercal \boldsymbol{x}) - 1 = \frac{2}{1 + e^{-\boldsymbol{w}^\intercal \boldsymbol{x}}} - 1 \to -1 \text{ i.e., to be minimized.} \tag{4.13}$$

Equivalently, for whatever y, we hope to minimize $1 + e^{-y\boldsymbol{w}^\intercal \boldsymbol{x}}$. So, we can formulate the learning problem as

$$\min_{\boldsymbol{w}} \frac{1}{N} \sum_{i=1}^{N} \left(1 + e^{-y_i \boldsymbol{w}^\intercal \boldsymbol{x}_i}\right). \tag{4.14}$$

Although this is a convex problem, the exponential term can cause numerical issues, as the exponent $-y_i \boldsymbol{w}^\intercal \boldsymbol{x}_i$ may be large for certain i's. Because making $1 + \exp(-y_i \boldsymbol{w}^\intercal \boldsymbol{x}_i)$ small is equivalent to making $\log(1 + \exp(-y_i \boldsymbol{w}^\intercal \boldsymbol{x}_i))$ small, we arrive at our *logistic regression* formulation:

$$\min_{\boldsymbol{w}} \frac{1}{N} \sum_{i=1}^{N} \log\left(1 + e^{-y_i \boldsymbol{w}^\intercal \boldsymbol{x}_i}\right). \tag{4.15}$$

One can easily verify that problem (4.15) is also a convex problem by checking the Hessian.

Logistic regression can also be derived from the maximum likelihood principle, as we will explore in the homework.

Further reading

Main reference is Chapter 9 of [SSS14]. Chapters 2–4 of [HTF09] are good supplements. [NW06] is a highly recommended reference for numerical optimization.

Disclaimer

This set of notes is preliminary and has not been thoroughly proofread. Typos and factual errors are well expected, and hence use it with caution. Bug reports are very welcome and should be sent to Prof. Ju Sun via jusun@umn.edu.

A Proof of Theorem 4.1

Proof. Since the requirement $y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle > 0$ for all $i \in [N]$ is homogenous in $\boldsymbol{w}$, we care only about the direction $\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}$ instead of $\boldsymbol{w}$ itself. Suppose $\boldsymbol{w}_*$ satisfies $\langle \boldsymbol{w}_*, \boldsymbol{x}_i \rangle \geq 1$ for all $i \in [N]$ with $\|\boldsymbol{w}_*\|_2 = B$. We want to show that $\boldsymbol{w}^{(T)}$ aligns with $\boldsymbol{w}_*$, i.e.,

$$\frac{\left\langle \boldsymbol{w}_*, \boldsymbol{w}^{(T)} \right\rangle}{\|\boldsymbol{w}_*\|_2 \|\boldsymbol{w}^{(T)}\|_2} = 1 \tag{A.1}$$

when T is large enough.

First, for any k,

$$\begin{aligned} \left\|\boldsymbol{w}^{(k)}\right\|_2^2 = \left\|\boldsymbol{w}^{(k-1)} + y_i \boldsymbol{x}_i\right\|_2^2 = \left\|\boldsymbol{w}^{(k-1)}\right\|_2^2 + \underbrace{\|\boldsymbol{x}_i\|_2^2}_{\leq R^2} + \underbrace{2 y_i \left\langle \boldsymbol{w}^{(k-1)}, \boldsymbol{x}_i \right\rangle}_{\leq 0} \\ \leq \left\|\boldsymbol{w}^{(k-1)}\right\|_2^2 + R^2 \Longrightarrow \left\|\boldsymbol{w}^{(k)}\right\|_2^2 - \left\|\boldsymbol{w}^{(k-1)}\right\|_2^2 \leq R^2. \end{aligned} \tag{A.2}$$

Using telescoping summation, we obtain

$$\left\|\boldsymbol{w}^{(T)}\right\|_2^2 = \sum_{k=1}^{T} \left(\left\|\boldsymbol{w}^{(k)}\right\|_2^2 - \left\|\boldsymbol{w}^{(k-1)}\right\|_2^2 \right) \leq T R^2. \tag{A.3}$$

On the other hand, for any k,

$$\left\langle \boldsymbol{w}_*, \boldsymbol{w}^k \right\rangle - \left\langle \boldsymbol{w}_*, \boldsymbol{w}^{k-1} \right\rangle = \left\langle \boldsymbol{w}_*, \boldsymbol{w}^k - \boldsymbol{w}^{k-1} \right\rangle = y_i \langle \boldsymbol{w}_*, \boldsymbol{x}_i \rangle \geq 1. \tag{A.4}$$

Applying telescoping summation again, we obtain

$$\left\langle \boldsymbol{w}_*, \boldsymbol{w}^{(T)} \right\rangle = \sum_{k=1}^{T} \left(\left\langle \boldsymbol{w}_*, \boldsymbol{w}^{(k)} \right\rangle - \left\langle \boldsymbol{w}_*, \boldsymbol{w}^{(k-1)} \right\rangle \right) \geq T. \tag{A.5}$$

Combining Eqs. (A.3) and (A.5), we finally obtain

$$\frac{\left\langle \boldsymbol{w}_*, \boldsymbol{w}^{(T)} \right\rangle}{\|\boldsymbol{w}_*\|_2 \|\boldsymbol{w}^{(T)}\|_2} \geq \frac{T}{B\sqrt{T}R} = \frac{\sqrt{T}}{BR}. \tag{A.6}$$

When $T \geq (RB)^2$,

$$\frac{\sqrt{T}}{BR} \geq 1 \Longrightarrow \frac{\left\langle \boldsymbol{w}_*, \boldsymbol{w}^{(T)} \right\rangle}{\|\boldsymbol{w}_*\|_2 \left\|\boldsymbol{w}^{(T)}\right\|_2} \geq 1. \tag{A.7}$$

But $\left\langle \boldsymbol{w}_*, \boldsymbol{w}^{(T)} \right\rangle \leq \|\boldsymbol{w}_*\|_2 \left\|\boldsymbol{w}^{(T)}\right\|_2$ due to the Cauchy-Schwarz inequality, implying that $\frac{\left\langle \boldsymbol{w}_*, \boldsymbol{w}^{(T)} \right\rangle}{\|\boldsymbol{w}_*\|_2 \left\|\boldsymbol{w}^{(T)}\right\|_2} \leq 1$. Thus, it takes at most $T = (RB)^2$ iterations to attain Eq. (A.1), completing the proof. ■

References

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman, *The elements of statistical learning*, Springer New York, 2009.

[MM17] Seymour A. Papert Marvin Minsky, *Perceptrons, reissue of the 1988 expanded edition with a new foreword by léon bottou*, MIT Press Ltd, 2017.

[NW06] Jorge Nocedal and Stephen J. Wright, *Numerical optimization*, 2 ed., Springer New York, 2006.

[Sra12] Suvrit Sra, *Optimization for machine learning*, MIT Press, Cambridge, Mass, 2012.

[SSS14] Shai Ben-David Shai Shalev-Shwartz, *Understanding machine learning*, Cambridge University Press, 2014.

[Sta] *Stanford EE364b - Convex Optimization II*, http://stanford.edu/class/ee364b/.

[Str16] Gilbert Strang, *Introduction to linear algebra*, 5 ed., Cambridge Press, Wellesley, MA, 2016.