SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.007 Machine Learning, Fall 2021
Lecture Notes for Week 1

2. Perceptron

Tuesday, 14 September, 2021

1 Learning Linear Classifiers

Previously, we started to explore linear classifiers through origin. These classifiers were defined as

$$h(x;\theta) = \text{sign}(\theta_1 x_1 + \ldots + \theta_d x_d) = \text{sign}(\theta \cdot x) = \begin{cases} +1, & \theta \cdot x \geq 0 \\ -1, & \theta \cdot x < 0 \end{cases} \tag{1}$$

where $\theta \cdot x = \theta^T x$ and $\theta = [\theta_1, \ldots, \theta_d]^T$ is a column vector of real valued parameters. The name linear classifier comes from the fact that the *decision boundary* is linear (line in 2d, plane in 3d, etc). Specifically, all $x \in \mathcal{R}^d$ that satisfy $\theta \cdot x = 0$ lie exactly on the decision boundary. Another way to understand binary classifiers is to explicitly evaluate how they divide the space (here $\mathcal{R}^d$) into two regions based on the label. For linear classifiers, both

$$\mathcal{X}^+(\theta) = \{x \in \mathcal{R}^d : h(x;\theta) = +1\} \tag{2}$$
$$\mathcal{X}^-(\theta) = \{x \in \mathcal{R}^d : h(x;\theta) = -1\} \tag{3}$$

are half-spaces, separated by the decision boundary. Note that, clearly, these half spaces as well as the decision boundary, depend on how we set θ, i.e., which classifier we choose.

Now that we have chosen a set of classifiers, we still need to choose one of them in response to the training set of labeled examples $S_n = \{(x(t), y(t)), t = 1, \ldots, n\}$. For simplicity, we assume that since our classifiers are linear, i.e., highly constrained, we can just find one that does well on the training set (we will revisit this issue later). For example, we could find θ that results in the fewest mistakes on the training set, i.e., we would minimize the training error

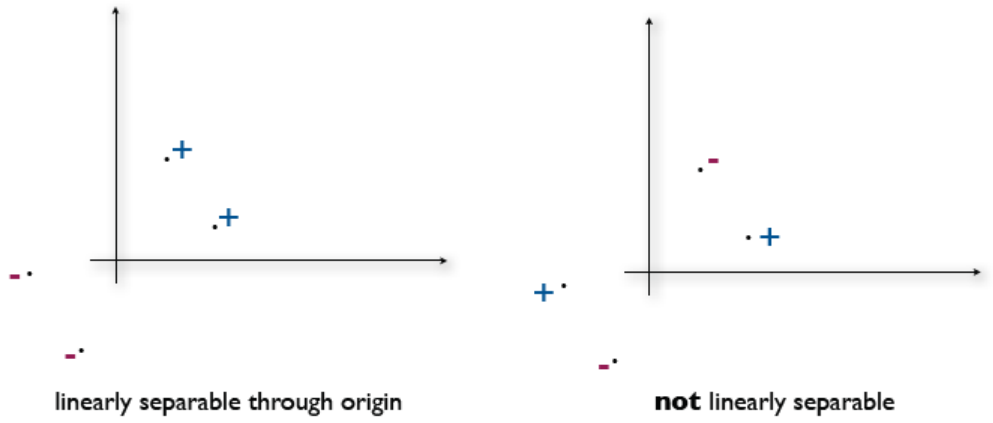
$$\mathcal{E}_n(\theta) = \frac{1}{n}\sum_{t=1}^{n} [\![y^{(t)} \neq h(x^{(t)};\theta)]\!] = \frac{1}{n}\sum_{t=1}^{n} [\![y^{(t)}(\theta \cdot x^{(t)}) \leq 0]\!] \tag{4}$$

where $[\![\cdot]\!]$ returns 1 if the logical expression in the argument is true, and zero otherwise. The training error here is the fraction of training examples for which the classifier with parameters θ predicts the wrong label. Note that incorrect prediction happens if $y(\theta \cdot x) \leq 0$, i.e., when the label y does not have the same sign as $\theta \cdot x$ or lies exactly on the decision boundary (which we will count as an error). The training error $\mathcal{E}_n(\theta)$ is calculated as a function of the parameters θ.

What would a reasonable algorithm be for finding $\hat{\theta}$ that minimizes $\mathcal{E}_n(\theta)$? Unfortunately, this is not an easy problem to solve in general, and we will have to settle for an algorithm that approximately minimizes the training error. However, for this lecture, we consider a special case where there exists a linear classifier (through origin) that achieves zero training error. This is also known as the *realizable case*. Note that "realizability" depends on both the training examples as well as the set of classifiers we have adopted. Specifically, for linear classifiers, we assume that the training examples are *linearly separable through origin*:

Definition 1.1 *Training examples* $S_n = \{(x^{(t)}, y^{(t)}), t = 1, \ldots, n\}$ *are linearly separable through origin if there exists a parameter vector* $\hat{\theta}$ *such that* $y^{(t)}(\hat{\theta} \cdot x^{(t)}) > 0$ *for all* $t = 1, \ldots, n$.

Here are a couple of examples:



2 The Perceptron Algorithm

We'll consider here an algorithm that is *mistake driven*. In other words, it starts with a simple classifier, e.g., $\theta = 0$ (zero vector), and successively tries to adjust the parameters, based on each training example, so as to correct any mistakes. The simplest algorithm of this type is the so-called *perceptron* update rule. In this algorithm, we set $\theta = 0$, and subsequently consider each training example one by one, cycling through all them, and adjusting the parameters according to:

$$\text{if } y^{(t)} \neq h(x^{(t)}; \theta^{(k)}) \text{ then} \tag{5}$$

$$\theta^{(k+1)} = \theta^{(k)} + y^{(t)} x^{(t)} \tag{6}$$

where $\theta^{(k)}$ denotes the parameters after k mistakes ($\theta^{(0)} = 0$). In other words, the parameters are changed only if we make a mistake, and we track the evolution of the parameters as a function of the mistakes. These updates do tend to correct mistakes. To see this, consider a simple two dimensional example in Figure 1. The points $x^{(1)}$ and $x^{(2)}$ in the figure are chosen such that the algorithm makes a mistake on both of them during it's first pass. As a result, the updates become:

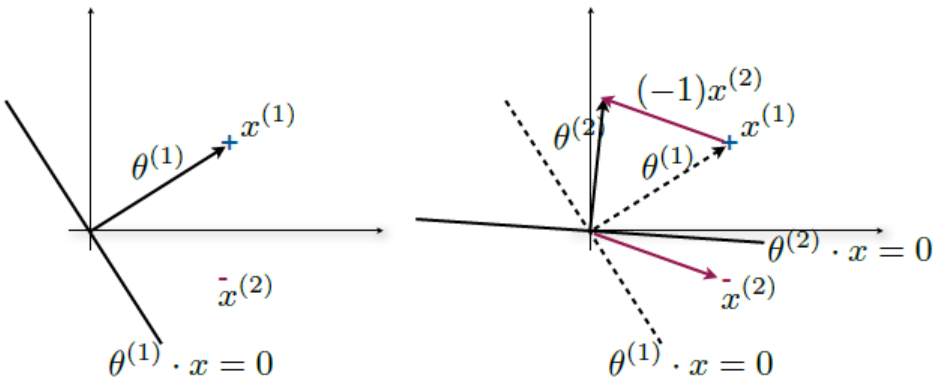


Figure 1: The perceptron update rule

$\theta^{(0)} = 0$ and

$$\theta^{(1)} = \theta^{(0)} + x^{(1)} \tag{7}$$

$$\theta^{(2)} = \theta^{(1)} + (-1)x^{(2)} \tag{8}$$

In this simple case, both updates result in correct classification of the respective examples and the algorithm would terminate. However, each update can also undershoot in the sense that the example that triggered the update would be misclassified even after the update. Can you construct a setting where an update would undershoot?

Let's look at the updates more algebraically. Note that when we make a mistake the sign of $(\theta^{(k)} \cdot x^{(t)})$ disagrees with $y^{(t)}$ and the product $y^{(t)}(\theta^{(k)} \cdot x^{(t)})$ is non-positive; the product is positive for correctly classified images. Suppose we make a mistake on $x^{(t)}$. Then the updated parameters are given by $\theta^{(k+1)} = \theta^{(k)} + y^{(t)}x^{(t)}$. If we consider classifying the same example $x^{(t)}$ after the update, using the new parameters $\theta(k+1)$, then

$$y^{(t)}(\theta^{(k+1)} \cdot x^{(t)}) = y^{(t)}(\theta^{(k)} + y^{(t)}x^{(t)}) \cdot x^{(t)} \tag{9}$$

$$= y^{(t)}(\theta^{(k)} \cdot x^{(t)}) + (y^{(t)})^2(x^{(t)} \cdot x^{(t)}) \tag{10}$$

$$= y^{(t)}(\theta^{(k)} \cdot x^{(t)}) + \left\|x^{(t)}\right\|^2 \tag{11}$$

In other words, the value of $y^{(t)}(\theta \cdot x^{(t)})$ increases as a result of the update (becomes more positive). If we consider the same example repeatedly, then we will necessarily change the parameters such that the example will be classified correctly, i.e., the value of $y^{(t)}(\theta \cdot x^{(t)})$ becomes positive. Of course, mistakes on other examples may steer the parameters in different directions, however, so it may not be clear that the algorithm converges to something useful if we repeatedly cycle through the training examples. The algorithm does converge in the realizable case:

Theorem 2.1 *The perceptron update rule converges after a finite number of mistakes when the training examples are linearly separable through origin.*

We will see later that the number of mistakes that the algorithm makes as it passes through the training examples depends on how easy or hard the classification task is. If the training examples are well-separated by a linear classifier (a notion which we will define formally later), the perceptron algorithm converges quickly, i.e., it makes only a few mistakes in total until all the training examples are correctly classified.

What if the training examples are not linearly separable? In this case, the algorithm cannot converge. There would always be a mistake in each pass through the training examples, and the parameters would be changed. Better algorithms exist, and will be discussed later on.

3 Linear Classifiers with Offset

We extend here the set of linear classifiers slightly by including a scalar offset parameter θ_0. This parameter will enable us to place the decision boundary anywhere in $\mathcal{R}^d$, not only through the origin. Specifically, a linear classifier with offset, or simply linear classifier, is defined as

$$h(x;\theta,\theta_0) = \text{sign}(\theta \cdot x + \theta_0) = \begin{cases} +1, & \theta \cdot x + \theta_0 \geq 0 \\ -1, & \theta \cdot x + \theta_0 < 0 \end{cases} \tag{12}$$

Clearly, if $\theta_0 = 0$, we obtain a linear classifier through origin. For a non-zero value of θ_0, the resulting decision boundary $\theta \cdot x + \theta_0 = 0$ no longer goes through the origin (see Figure 2 below). The hyper-plane (line in 2d) $\theta \cdot x + \theta_0 = 0$ is oriented parallel to $\theta \cdot x = 0$. If they were not, then there should be some x that satisfies both equations: $\theta \cdot x + \theta_0 = \theta \cdot x = 0$. This is possible only if $\theta_0 = 0$. We can conclude that vector θ is still orthogonal to the decision boundary, and also defines the positive direction in the sense that if we move x in this direction, the value of $\theta \cdot x + \theta_0$ increases. In the figure below, $\theta_0 < 0$ because we have to move from the origin (where $\theta \cdot x = 0$) in the direction of θ (increasing $\theta \cdot x$) until we hit $\theta \cdot x + \theta_0 = 0$.

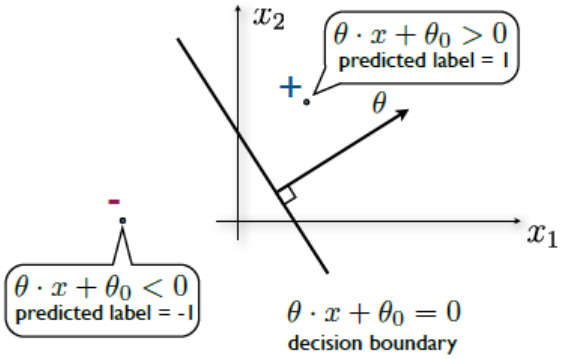


Figure 2: Linear classifier with offset parameter

Both linear separability and the perceptron algorithm for learning linear classifiers generalize easily to the case of linear classifiers with offset. Specifically,

Definition 3.1 *Training examples* $S_n = \{(x^{(t)}, y^{(t)}), t = 1; \ldots, n\}$ *are linearly separable if there exists a parameter vector* $\hat{\theta}$ *and offset parameter* $\hat{\theta}_0$ *such that* $y^{(t)}(\hat{\theta} \cdot x^{(t)} + \hat{\theta}_0) > 0$ *for all* $t = 1, \ldots, n$.

If training examples are linearly separable through origin, they are clearly also linearly separable. The converse is not true in general, however. Can you find such an example?

The perceptron algorithm is also modified only slightly: initialize $\theta^{(0)} = 0$ (vector) and $\theta_0^{(0)} = 0$ (scalar). Cycle through the training examples $t = 1, \ldots, n$ and update parameters according to

$$\text{if } y^{(t)} \neq h(x^{(t)}; \theta^k, \theta_0^k) \text{ then} \tag{13}$$

$$\theta^{(k+1)} = \theta^{(k)} + y^{(t)} x^{(t)} \tag{14}$$

$$\theta_0^{(k+1)} = \theta_0^{(k)} + y^{(t)} \tag{15}$$

Why is the offset parameter updated in this way? Think of it as a parameter associated with an additional coordinate that is set to 1 for all examples. If training examples are linearly separable (not necessarily through the origin), then the above perceptron algorithm converges after a finite number of mistakes.

Learning Objective

You need to know:

1. What is a linearly separable set of examples.
2. How the Perceptron algorithm works.
3. What is the guarantee of the Perceptron algorithm when the dataset is linearly separable.