# Chapter 2: Linear Classifiers

Latreche Sara

## Contents

**1 Classification** **2**

**2 Learning Algorithm** **3**

**3 Linear Classifiers** **3**

**4 Learning Linear Classifier Algorithm** **5**

**5 Evaluating a Learning Algorithm** **7**

# 1 Classification

**Concept**

A binary classifier is a mapping:

$$h : \mathbb{R}^d \to \{-1, +1\}$$

It assigns a label to an input vector $x \in \mathbb{R}^d$, typically derived from real-world data (images, sounds, etc.). We assume preprocessing has already been done, and work directly in feature space.

We denote a classifier by $h$, so the process is:

$$x \xrightarrow{h} y$$

In supervised learning, we are given a dataset:

$$D_n = \left\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\right\}$$

where $y^{(i)} \in \{-1, +1\}$.

The goal is to find a hypothesis $h$ that generalizes well. Given a training set $D_n$ and a classifier $h$, we can define the training error of $h$ to be:

$$E_n(h) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} 1 & \text{if } h(x^{(i)}) \neq y^{(i)} \\ 0 & \text{otherwise} \end{cases}$$

For now, we will try to find a classifier with small training error (later, with some added criteria) and hope it generalizes well to new data, and has a small test error:

$$E(h) = \frac{1}{n_0} \sum_{i=n+1}^{n+n_0} \begin{cases} 1 & \text{if } h(x^{(i)}) \neq y^{(i)} \\ 0 & \text{otherwise} \end{cases}$$

## 2 Learning Algorithm

A **hypothesis class** $\mathcal{H}$ is a set (finite or infinite) of possible classifiers, each of which represents a mapping:

$$h : \mathbb{R}^d \to \{-1, +1\}.$$

A **learning algorithm** is a procedure that takes a training dataset $D_n$ as input and returns a hypothesis $h \in \mathcal{H}$:

$$D_n \xrightarrow{\text{learning algorithm}(\mathcal{H})} h.$$

The choice of hypothesis class $\mathcal{H}$ can significantly impact the test error of the resulting classifier $h$. One common approach to improve generalization is to restrict the size or "expressiveness" of $\mathcal{H}$, thus avoiding overly complex models that might overfit the training data.

## 3 Linear Classifiers

We start with the hypothesis class of **linear classifiers**. These classifiers are relatively easy to understand, mathematically simple, powerful on their own, and form the basis of many more sophisticated methods.

A linear classifier in $d$ dimensions is defined by a vector of parameters $\theta \in \mathbb{R}^d$ and a scalar $\theta_0 \in \mathbb{R}$. Thus, the hypothesis class $\mathcal{H}$ of linear classifiers in $d$ dimensions is the set of all vectors in $\mathbb{R}^{d+1}$. We assume that $\theta$ is a $d \times 1$ column vector.

**Toolbox: Linear Classifier Decision Rule**

Given particular values for $\theta$ and $\theta_0$, the classifier is defined by:

$$h(x; \theta, \theta_0) = \text{sign}(\theta^\top x + \theta_0) = \begin{cases} +1 & \text{if } \theta^\top x + \theta_0 > 0, \\ -1 & \text{otherwise.} \end{cases}$$

This hyperplane splits the space into positive and negative half-spaces.

**Note on dimensions:** Since both $x$ and $\theta$ are $d \times 1$ column vectors, the product $\theta^\top x$ is $1 \times 1$, which mathematically corresponds to a scalar.

We can think of $\theta$ and $\theta_0$ as specifying a **hyperplane** in $\mathbb{R}^d$. This hyperplane divides the space into two half-spaces:

- The **positive half-space** contains all points $x$ such that $\theta^\top x + \theta_0 > 0$. Points in this space are classified as $+1$.
- The **negative half-space** contains all points where $\theta^\top x + \theta_0 \leq 0$, classified as $-1$.

The vector $\theta$ is **normal** (perpendicular) to the hyperplane and points toward the positive half-space.

**Toolbox: Example: Classification**

Consider the linear classifier defined by:

$$\theta = \begin{bmatrix} -1 \\ 1.5 \end{bmatrix}, \quad \theta_0 = 3.$$

We classify the points:

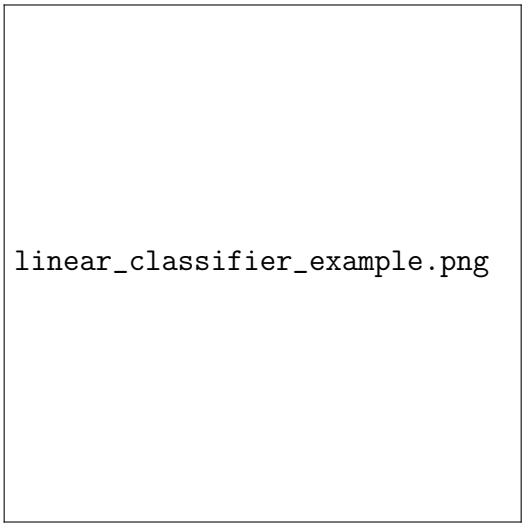
$$x^{(1)} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, \quad x^{(2)} = \begin{bmatrix} 4 \\ -1 \end{bmatrix}.$$

Calculate:

$$\begin{aligned} h(x^{(1)}; \theta, \theta_0) &= \text{sign}\left( \begin{bmatrix} -1 & 1.5 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix} + 3 \right) \\ &= \text{sign}(-3 + 3 + 3) \\ &= \text{sign}(3) = +1, \end{aligned}$$

$$\begin{aligned} h(x^{(2)}; \theta, \theta_0) &= \text{sign}\left( \begin{bmatrix} -1 & 1.5 \end{bmatrix} \begin{bmatrix} 4 \\ -1 \end{bmatrix} + 3 \right) \\ &= \text{sign}(-4 - 1.5 + 3) \\ &= \text{sign}(-2.5) = -1. \end{aligned}$$

Thus, $x^{(1)}$ is classified as positive and $x^{(2)}$ as negative.




**Toolbox: Study Questions**

1. What is the green vector normal to the hyperplane? Specify it as a column vector.
2. What change would you have to make to $\theta$ and $\theta_0$ to keep the separating hyperplane in the same place, but classify all points labeled $+$ in the diagram as negative, and all points labeled $-$ as positive?

# 4 Learning Linear Classifier Algorithm

Given a dataset and the hypothesis class of linear classifiers, our objective is to find the linear classifier with the smallest possible training error.

This is a well-formed optimization problem. However, it is not computationally easy! To build intuition, we will begin by exploring a very simple (and arguably naive) learning algorithm. It is often helpful to consider the "stupidest possible" solution before trying to get clever.

**Toolbox: Algorithm: Random Linear Classifier**

We generate $k$ possible hypotheses by randomly sampling their parameter vectors. Then, we evaluate the training-set error for each hypothesis and return the one with the lowest error (breaking ties arbitrarily).

**Input:**

- $D_n$: A training dataset of $n$ labeled examples in $\mathbb{R}^d$
- $k$: Number of hypotheses to try
- $d$: Dimensionality of feature vectors

**Output:**

- $\left(\theta^{(j^*)}, \theta_0^{(j^*)}\right)$: The hypothesis (parameter vector and offset) with the lowest training error

**Procedure:**

**RANDOM-LINEAR-CLASSIFIER**$(D_n, k, d)$
**for** $j = 1$ to $k$ **do**
  Randomly sample $(\theta^{(j)}, \theta_0^{(j)})$ from $(\mathbb{R}^d, \mathbb{R})$
$j^* = \arg \min_{j \in \{1,\ldots,k\}} E_n(\theta^{(j)}, \theta_0^{(j)})$
**return** $(\theta^{(j^*)}, \theta_0^{(j^*)})$

**Toolbox: Numerical Example: Evaluating Hypotheses**

Let the training data be:

$$D_3 = \left\{ \left( \begin{bmatrix} 1 \\ 2 \end{bmatrix}, +1 \right), \left( \begin{bmatrix} -1 \\ -1 \end{bmatrix}, -1 \right), \left( \begin{bmatrix} 2 \\ 0 \end{bmatrix}, +1 \right) \right\}$$

Suppose we randomly generate $k = 2$ hypotheses:

$$\theta^{(1)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \theta_0^{(1)} = -1 \quad \Rightarrow \quad h^{(1)}(x) = \text{sign}(\theta^{(1)} \cdot x + \theta_0^{(1)})$$

$$\theta^{(2)} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \quad \theta_0^{(2)} = 0.5 \quad \Rightarrow \quad h^{(2)}(x) = \text{sign}(\theta^{(2)} \cdot x + \theta_0^{(2)})$$

Evaluating predictions:
- $h^{(1)}$:

$$\begin{aligned} &\text{sign}(1 + 2 - 1) = +1 \quad \text{(correct)} \\ &\text{sign}(-1 - 1 - 1) = -1 \quad \text{(correct)} \\ &\text{sign}(2 + 0 - 1) = +1 \quad \text{(correct)} \\ \Rightarrow &\ 0 \text{ errors} \end{aligned}$$

- $h^{(2)}$:

$$\begin{aligned} &\text{sign}(-1 + 4 + 0.5) = +1 \quad \text{(correct)} \\ &\text{sign}(1 - 2 + 0.5) = -0.5 \Rightarrow -1 \quad \text{(correct)} \\ &\text{sign}(-2 + 0 + 0.5) = -1.5 \Rightarrow -1 \quad \textbf{(wrong)} \\ \Rightarrow &\ 1 \text{ error} \end{aligned}$$

Therefore, $j^* = 1$ and the algorithm returns $(\theta^{(1)}, \theta_0^{(1)})$.

# 5 Evaluating a Learning Algorithm

How should we evaluate the performance of a classifier $h$? The best method is to measure **test error** on data that was **not used to train it**.

However, evaluating the performance of a **learning algorithm** (not just a single classifier) is trickier. There are many potential sources of variability in the test error of a learned hypothesis $h$:

- The specific training examples in $D_n$
- The specific testing examples in $D_{n'}$
- Randomization inside the learning algorithm itself

To account for this variability, we ideally repeat the following procedure multiple times:

- Train on a newly sampled training set
- Evaluate the resulting hypothesis $h$ on a disjoint testing set

Doing this multiple times helps **control for poor choices of training data or randomness in the algorithm**. However, this can be **data-intensive**, which is a concern in applications where labeled data is **scarce or expensive**.

## Cross-Validation

A practical workaround is **cross-validation**, which allows data reuse by partitioning the dataset.

**Toolbox: Procedure: Cross-Validation**

Cross-validation allows us to estimate the generalization performance of a learning algorithm by systematically rotating training and testing roles across different data chunks.

**Input:**

- $D$: Full dataset
- $k$: Number of folds (chunks)

**Output:**

- An estimate of average generalization error

**Procedure:**

**CROSS-VALIDATE**$(D, k)$
Divide $D$ into $k$ chunks: $D_1, D_2, \ldots, D_k$
**for** $i = 1$ to $k$ **do**
  Train hypothesis $h_i$ on $D \setminus D_i$ (withholding $D_i$)
  Compute error $E_i(h_i)$ on $D_i$
**return** $\frac{1}{k} \sum_{i=1}^{k} E_i(h_i)$

**Important Note:** Cross-validation does **not** produce or evaluate a single hypothesis $h$. Instead, it evaluates the performance of the **learning algorithm** by testing it across multiple train-test splits.

## Study Questions

1. What is the green vector normal to the hyperplane? Specify it as a column vector.
2. What change would you have to make to $\theta$ and $\theta_0$ to keep the separating hyperplane in the same place, but classify all points labeled + in the diagram as negative, and all points labeled − as positive?

**Study Question**

What vector is normal to the hyperplane defined by $\theta^\top x + \theta_0 = 0$?
**Answer:** The vector $\theta$ is normal to the hyperplane.

# References


- MIT OpenCourseWare, *6.0002: Introduction to Computational Thinking and Data Science*, Fall 2016, Lecture 12.