

CHAPTER 2

Linear classifiers

1 Classification

A binary *classifier* is a mapping from $\mathbb{R}^d \rightarrow \{-1, +1\}$. We'll often use the letter h (for hypothesis) to stand for a classifier, so the classification process looks like:

$$x \rightarrow \boxed{h} \rightarrow y.$$

Actually, general classifiers can have a range which is any discrete set, but we'll work with this specific case for a while.

Real life rarely gives us vectors of real numbers; the x we really want to classify is usually something like a song, image, or person. In that case, we'll have to define a function $\varphi(x)$, whose domain is \mathbb{R}^d , where φ represents *features* of x , like a person's height or the amount of bass in a song, and then let the $h : \varphi(x) \rightarrow \{-1, +1\}$. In much of the following, we'll omit explicit mention of φ and assume that the $x^{(i)}$ are in \mathbb{R}^d , but you should always have in mind that some additional process was almost surely required to go from the actual input examples to their feature representation.

In *supervised learning* we are given a training data set of the form

$$\mathcal{D}_n = \left\{ (x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)}) \right\}.$$

We will assume that each $x^{(i)}$ is a $d \times 1$ *column vector*. The intended meaning of this data is that, when given an input $x^{(i)}$, the learned hypothesis should generate output $y^{(i)}$.

What makes a classifier useful? That it works *well on new data*; that is, that it makes *good predictions on examples* it hasn't seen. But we don't know exactly what data this classifier might be *tested* on when we use it in the real world. So, we have to *assume* a connection between the *training data and testing data*; typically, they are drawn independently from the *same probability distribution*.

My favorite analogy is to problem sets. We evaluate a student's ability to *generalize* by putting questions on the exam that were not on the homework (training set).

Given a training set \mathcal{D}_n and a classifier h , we can define the *training error* of h to be

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

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*.

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

on n' new examples that were not used in the process of finding the classifier.

2 Learning algorithm

A **hypothesis class** \mathcal{H} is a set (finite or infinite) of possible classifiers, each of which represents a **mapping from** $\mathbb{R}^d \rightarrow \{-1, +1\}$.

A **learning algorithm** is a **procedure** that takes a data set \mathcal{D}_n as input and returns an element h of \mathcal{H} ; it looks like

$$\mathcal{D}_n \longrightarrow \boxed{\text{learning alg } (\mathcal{H})} \longrightarrow h \quad \text{classifier}$$

We will find that the choice of \mathcal{H} can have a **big impact** on the **test error of the h** that results from this process. One way to get h that generalizes well is to restrict the size, or “**expressiveness**” of \mathcal{H} .

3 Linear classifiers

We’ll start with the hypothesis class of **linear classifiers**. They are (relatively) easy to understand, simple in a mathematical sense, powerful on their own, and the basis for many other more sophisticated methods.

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

Given particular values for θ and θ_0 , **the classifier** is defined by

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

Remember that we can think of θ, θ_0 as **specifying a hyperplane**. It divides \mathbb{R}^d , the space our $x^{(i)}$ points live in, into two half-spaces. The one that is on the same side as the normal vector is the **positive half-space**, and we classify all points in that space as positive. The half-space on the other side is **negative** and all points in it are classified as negative.

Let’s be careful about dimensions. We have assumed that x and θ are both $d \times 1$ column vectors. So $\theta^T x$ is 1×1 , which in math (but not necessarily numpy) is the same as a scalar.

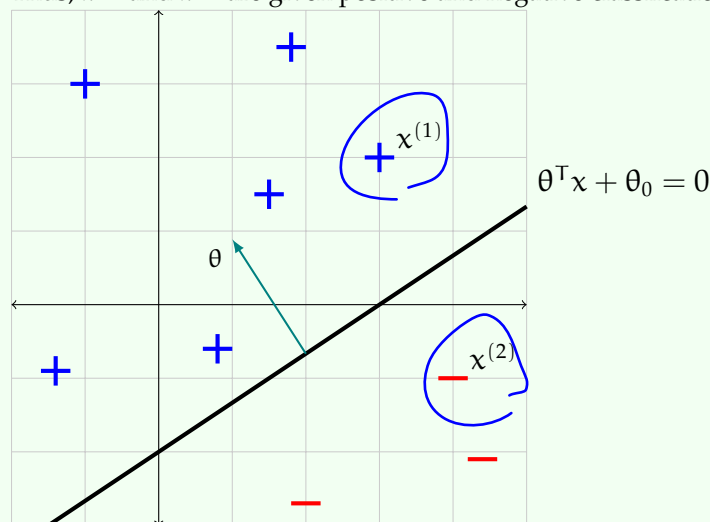
Example: Let h be the linear classifier defined by $\theta = \begin{bmatrix} -1 \\ 1.5 \end{bmatrix}$, $\theta_0 = 3$.

The diagram below shows several points classified by h . In particular, let $x^{(1)} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$ and $x^{(2)} = \begin{bmatrix} 4 \\ -1 \end{bmatrix}$.

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

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

Thus, $x^{(1)}$ and $x^{(2)}$ are given positive and negative classifications, respectively.



Study Question: What is the green vector normal to the hyperplane? Specify it as a column vector.

Study Question: What change would you have to make to θ, θ_0 if you wanted to have the separating hyperplane in the same place, but to classify all the points labeled '+' in the diagram as negative and all the points labeled '-' in the diagram as positive?

$$\begin{bmatrix} -1 \\ 1.5 \end{bmatrix} = \theta$$

$$-x_1 + 1.5x_2 + 3 = 0$$

$$\begin{bmatrix} 1 \\ -1.5 \end{bmatrix} - 3$$

4 Learning linear classifiers

Now, given a data set and the hypothesis class of linear classifiers, our objective will be to find the linear classifier with the smallest possible training error.

This is a well-formed optimization problem. But it's not computationally easy!

We'll start by considering a very simple learning algorithm. The idea is to generate k possible hypotheses by generating their parameter vectors at random. Then, we can evaluate the training-set error on each of the hypotheses and return the hypothesis that has the lowest training error (breaking ties arbitrarily).

It's a good idea to think of the "stupidest possible" solution to a problem, before trying to get clever. Here's a fairly (but not completely) stupid algorithm.

RANDOM-LINEAR-CLASSIFIER(\mathcal{D}_n, k)

```

1  for j = 1 to k
2      randomly sample  $(\theta^{(j)}, \theta_0^{(j)})$  from  $(\mathbb{R}^d, \mathbb{R})$ 
3   $j^* = \arg \min_{j \in \{1, \dots, k\}} \mathcal{E}_n(\theta^{(j)}, \theta_0^{(j)})$ 
4  return  $(\theta^{(j^*)}, \theta_0^{(j^*)})$ 

```

A note about notation.

Study Question: What do you think happens to $\mathcal{E}_n(h)$, where h is the hypothesis returned by RANDOM-LINEAR-CLASSIFIER, as k is increased?

Study Question: What properties of \mathcal{D}_n do you think will have an effect on $\mathcal{E}_n(h)$?

This might be new notation: $\arg \min_x f(x)$ means the value of x for which $f(x)$ is the smallest. Sometimes we write $\arg \min_{x \in \mathcal{X}} f(x)$ when we want to explicitly specify the set \mathcal{X} of values of x over which we want to minimize.

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.

How should we evaluate the *performance of a learning algorithm*? This is trickier. There are many potential sources of variability in the possible result of computing test error on a learned hypothesis h :

- Which particular *training examples* occurred in \mathcal{D}_n
- Which particular *testing examples* occurred in \mathcal{D}_n
- Randomization inside the *learning algorithm* itself

Generally, we would like to execute the following process multiple times:

- Train on a new training set *multiple classifier based on diff training set*
- Evaluate resulting h on a testing set that does not overlap the training set

Doing this *multiple times* controls for *possible poor choices of training set* or *unfortunate randomization inside the algorithm itself*.

One concern is that we might need a lot of data to do this, and in many applications data is expensive or difficult to acquire. We can re-use data with *cross validation* (but it's harder to do theoretical analysis).

CROSS-VALIDATE(\mathcal{D}, k)

```

1  divide  $\mathcal{D}$  into  $k$  chunks  $\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_k$  (of roughly equal size)
2  for i = 1 to k
3      train  $h_i$  on  $\mathcal{D} \setminus \mathcal{D}_i$  (withholding chunk  $\mathcal{D}_i$ )
4      compute "test" error  $\mathcal{E}_i(h_i)$  on withheld data  $\mathcal{D}_i$ 
5  return  $\frac{1}{k} \sum_{i=1}^k \mathcal{E}_i(h_i)$ 

```

It's very important to understand that cross-validation neither delivers nor evaluates a single particular hypothesis h . It evaluates the *algorithm* that produces hypotheses.