# Chapter 6

# Classification

We will now transition to **classification** problems, which constitute an extremely important class of problems in data analytics.

At a high level: a classification procedure, or *classifier* for short, processes a given set of objects that are *known* to belong to one of two (or more) classes. Then, it attempts to predict the class membership of a new object whose class membership is unknown. (The idea of processing a new, potential unseen object is key here.)

Classification problems arise in widespread applications ranging from automatic image tagging, to document categorization, to spam filtering, to sensor network monitoring, to software bug detection – you name it. It is one of the canonical pattern recognition problems. We will discuss a range of techniques for classification, starting with one that is particularly easy to implement.

## $k$-nearest neighbors

We study the binary classification problem. Here, and below, we will assume as input:

- A set of *labeled* training samples: $X = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where each $x_i \in \mathbb{R}^d$ represent data vectors, each $y_i \in \{-1, 1\}$ represent class labels. (Here, the $\pm 1$ don't represent numerical values; they could equally well be 0 or 1 with little change in the algorithms.)
- An *unlabeled* test data point $x \in \mathbb{R}$. Our goal is to output the label of $x$, denoted by $y$.

A reasonable hypothesis is that data points which have opposite labels are far away in the data space, and points with similar labels are nearby. Therefore, an obvious solution for the above problem can be obtained via nearest neighbors.

1. Compute distances $d(x, x_i)$ for each $i = 1, 2, \ldots, n$. Here, think of $d$ as the Euclidean norm or the $\ell_1$ norm. (The exact choice of distance is critical.)
2. Compute the nearest neighbor to $x$, i.e., find $i^* = \arg\min_i d(x, x_i)$.
3. Output $y = y_{i^*}$.

Therefore, this algorithm is a straightforward application of nearest neighbors. The running time (for each test point) is $O(nd)$.

While conceptually simple, there are several issues with such a nearest neighbors approach. First, it is notoriously sensitive to errors and outliers. For example, in an image recognition example, say we have a bunch of (labelled) cat and dog images, but some of the dog images are mislabeled as cats (and vice versa). Then, a new dog image that is closest to a mislabeled dog image will also get a wrong label assignment using the above algorithm.

The typical way to use this is via *$k$-nearest neighbors*. Find the $k$-nearest data points to the query $x$, look at their labels, and output the majority among the labels. For convenience of defining majority, $k$ can be chosen to be an odd integer. (Even is fine as well, but 50-50 ties may occur and there should be a consistent rule to break them.) This helps mitigate outliers to some extent.

The running time of the $k$-NN algorithm is again $O(nd)$ for each query. This brings us to the second issue with NN (and $k$-NN): the *per-query* running time is quadratic (i.e., it scales with both $n$ and $d$). This quickly becomes infeasible for applications involving lots of query points (e.g. imagine a spam classification algorithm trying to label millions of messages).

## Occam's razor and linear separators

Every (deterministic) binary classification algorithm $A$, given a training data set $X$, automatically partitions the data space into two subsets – the set of points that will be labeled as $+1$ and the set of points that will be labeled as $-1$. These are called *decision regions* of the classifier. Really, the only information you need for classification are the decision regions (the role of training data is to help build these regions.)

The third issue with $k$-NN is that the decision regions are overly complex. The boundary of the two classes using NN classifiers is an irregular hypersurface that depends on the training data.

As a guiding principle in machine learning, it is customary to follow:

> Occam's Razor: Simpler hypotheses usually perform better than more complex ones.

One family of "simple" classifiers are *linear* classifiers, i.e., algorithms that try to lean decision regions with a straight line as the separating boundary. We will try to find such classifiers.

## The perceptron algorithm

The perceptron algorithm was an early attempt to solve the problem of artificial intelligence. Indeed, after its initiation in the early 1950s, people believed that perfect classification methods were not far off. (Of course, that didn't quite work out yet.)

The goal of the perceptron algorithm is to learn a linear separator between two classes. The algorithm is simple and parameter-free: no excessive tuning is required. Later, we will see that the algorithm is an instance of a more general family of learning methods known as *stochastic gradient descent*.

In particular, the perceptron will output a vector $w \in \mathbb{R}$ and a scalar $b \in \mathbb{R}$ such that for each input data vector $x$, its predicted label is:

$$y = \text{sign}(\langle w, x_i \rangle + b).$$

Below, without loss of generality, we will assume that the learned separator $w$ is homogeneous and passes through the origin (i.e., $b = 0$), and that the data is normalized such that $\|x\| \leq 1$. Geometrically, the boundary of the decision regions is the hyperplane:

$$\langle w, x \rangle = 0$$

and $w$ denotes any vector normal to this hyperplane.

We will also assume that the training data is indeed perfectly separable. (Of course, in practice we are not guaranteed that this is the case. We will see how to fix this later.)

**Input**: Training samples $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$.

**Output**: A vector $w$ such that $y_i = \text{sign}(\langle w, x_i \rangle)$ for all $(x_i, y_i) \in S$.

```
0. Initialize $w_0 = 0$.
1. Repeat:
    a. For each $(x_i, y_i) \in S$, if sign($\langle w_{t-1}, x_i \rangle$) $\neq y_i$, update
           $w_t \leftarrow w_{t-1} + y_i x_i$
       and increment $t$.
    b. If no change in $w_t$ after sweeping through the data, terminate.
   While epoch $\leq 1, .., maxepochs$.
```

The high level idea is that each time a training sample $x_i$ is mislabeled, we incrementally adjust the current estimate of the separator, $w$, in the direction of $x_i$ to correct for it.

# Analysis of the perceptron

We now discuss algorithm correctness and efficiency.

The per-iteration complexity of the algorithm is $O(nd)$: we need to sweep through each of the $n$ data points $x_i$, compute the $d$-dimensional dot-product $\langle w, x_i \rangle$ and compare its sign with the true label.

The algorithm makes an update only when a training sample is mislabeled, and hence terminates only when every point is labeled correctly. So it might seem that the number of iterations can be very large. We will provide an upper bound for this.

For any given dataset and a separator $w$, define the *margin* as the minimum projection distance between any data point to the hyperplane $\langle w, x \rangle = 0$ i.e.,

$$\gamma = \arg \min_{i \in 1, \ldots, n} |\langle w, x_i \rangle|.$$

The optimal separator is defined as the vector $w_*$ that maximizes $\gamma$ over all valid separators. Without loss of genelarity, assume that the Euclidean norm of $w_*$ is 1.

We prove that the perceptron will terminate after $1/\gamma^2$ updates.

First, consider the quantity $\langle w_t, w_* \rangle$, i.e., the dot-product between the current estimate and the optimal separator. If the label $y_i$ is positive, then $\langle w_{t+1}, w_* \rangle = \langle w_t + x_i, w_* \rangle$, which by linearity of the dot-product is larger than $\langle w_t, w_* \rangle$ by at least $\gamma$ (by definition of the margin). The same result holds if the label is negative; easy to check.

Therefore, after each update step, $\langle w, w_* \rangle$ increases by at least $\gamma$.

Next, consider the quantity $\|w_t\|^2 = \langle w_t, w_t \rangle$. After each update, this quantity changes to

$$\begin{aligned}\|w_{t+1}\|^2 &= \langle w_{t+1}, w_{t+1} \rangle \\ &= \langle w_t + x_i, w_t + x_i \rangle \\ &\leq \|w_t\|^2 + \|x_i\|^2 \\ &\leq \|w_t\|^2 + 1,\end{aligned}$$

since we have assumed that $\|x_i\| \leq 1$ for all $i$.

Putting the two together, we have, after $T$ update steps:

$$\langle w_T, w_* \rangle \geq \gamma T$$

and

$$\|w_T\|^2 \leq T \rightarrow \|w_T\| \leq \sqrt{T}.$$

By the Cauchy Schwartz inequality and the fact that $\|w_*\| = 1$, we know that

$$\langle w_T, w_* \rangle \leq \|w_T\| \|w_*\| = \|w_T\|.$$

Therefore,

$$\gamma T \leq \sqrt{T} \rightarrow T \leq 1/\gamma^2.$$