COMS 4721: Machine Learning for Data Science
Lecture 8, 2/14/2019

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

LINEAR CLASSIFICATION

BINARY CLASSIFICATION

We focus on binary classification, with input $x_i \in \mathbb{R}^d$ and output $y_i \in \{\pm 1\}$.

- We define a *classifier* f, which makes prediction $y_i = f(x_i, \Theta)$ based on a function of x_i and parameters Θ. In other words $f : \mathbb{R}^d \to \{-1, +1\}$.

Last lecture, we discussed the **Bayes classification** framework.

- Here, Θ contains: (1) class prior probabilities on y,
  (2) parameters for class-dependent distribution on x.

This lecture we'll introduce the **linear classification** framework.

- In this approach the prediction is linear in the parameters Θ.
- In fact, there is an intersection between the two that we discuss next.

Bayes decisions

With the Bayes classifier we predict the class of a new x to be the most probable label given the model and training data $(x_1, y_1), \dots, (x_n, y_n)$.

In the binary case, we declare class $y = 1$ if

$$p(x|y=1)\,\underbrace{P(y=1)}_{\pi_1} \quad > \quad p(x|y=0)\,\underbrace{P(y=0)}_{\pi_0}$$

$$\Updownarrow$$

$$\ln \frac{p(x|y=1)P(y=1)}{p(x|y=0)P(y=0)} \quad > \quad 0$$

This second line is referred to as the *log odds*.

Gaussian with shared covariance

Let's look at the log odds for the special case where

$$p(x|y) = N(x|\mu_y, \Sigma)$$

(i.e., a single Gaussian with a shared covariance matrix)

$$\ln \frac{p(x|y=1)P(y=1)}{p(x|y=0)P(y=0)} = \underbrace{\ln \frac{\pi_1}{\pi_0} - \frac{1}{2}(\mu_1 + \mu_0)^T \Sigma^{-1} (\mu_1 - \mu_0)}_{\text{a constant, call it } w_0} + x^T \underbrace{\Sigma^{-1}(\mu_1 - \mu_0)}_{\text{a vector, call it } w}$$

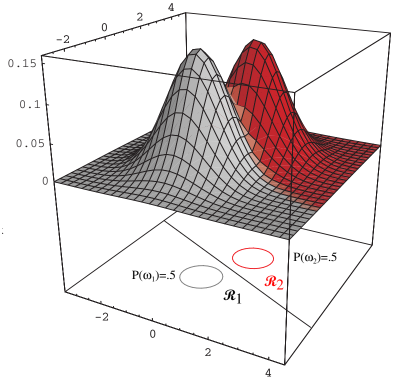
This is also called "linear discriminant analysis" (used to be called LDA).

A BAYES CLASSIFIER

So we can write the decision rule for this Bayes classifier as a linear one:

$$f(x) = \mathrm{sign}(x^T w + w_0).$$

- This is what we saw last lecture (but now class 0 is called -1)
- The Bayes classifier produced a linear decision boundary in the data space when $\Sigma_1 = \Sigma_0$.
- w and w_0 are obtained through a specific equation.



LINEAR CLASSIFIERS

This Bayes classifier is one instance of a linear classifier

$$f(x) = \text{sign}(x^T w + w_0)$$

where

$$\begin{aligned} w_0 &= \ln \frac{\pi_1}{\pi_0} - \frac{1}{2}(\mu_1 + \mu_0)^T \Sigma^{-1}(\mu_1 - \mu_0) \\ w &= \Sigma^{-1}(\mu_1 - \mu_0) \end{aligned}$$

With maximum likelihood used to find values for π_y, μ_y and Σ.

Setting w_0 and w this way may be too restrictive:

- This Bayes classifier assumes single Gaussian with shared covariance.
- Maybe if we relax what values w_0 and w can take we can do better.
- (Alternatively, we could also pick a more complex $p(x|y)$.)

LINEAR CLASSIFIERS (BINARY CASE)

Definition: Binary linear classifier

A *binary linear classifier* is a function of the form

$$f(x) = \mathrm{sign}(x^T w + w_0),$$

where $w \in \mathbb{R}^d$ and $w_0 \in \mathbb{R}$. Since we want to learn (w, w_0) from data, we are assuming that *linear separability* in x is an accurate property of the classes.

Definition: Linear separability

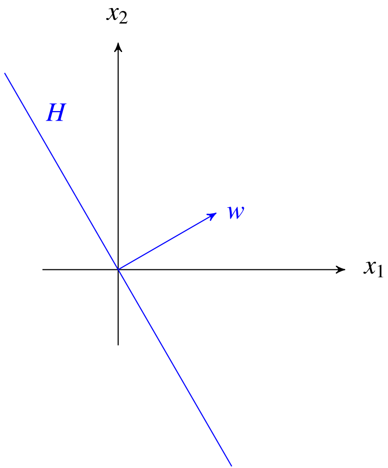
Two sets $A, B \subset \mathbb{R}^d$ are called *linearly separable* if for some (w, w_0),

$$x^T w + w_0 \quad \begin{cases} > 0 & \text{if } x \in A \ \ \text{(e.g, class } +1) \\ < 0 & \text{if } x \in B \ \ \text{(e.g, class } -1) \end{cases}$$

The pair (w, w_0) defines an *affine hyperplane*. It is very important to develop the right geometric understanding about what this is doing.

HYPERPLANES

Geometric interpretation of linear classifiers:



(The intuition is different from linear regression!)

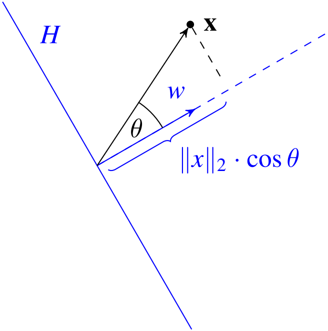
A *hyperplane* in $\mathbb{R}^d$ is a linear subspace of dimension $(d-1)$.

- An $\mathbb{R}^2$-hyperplane is a line.
- An $\mathbb{R}^3$-hyperplane is a plane.
- As a linear subspace, a hyperplane always contains the origin.

A hyperplane H can be represented by a vector w as follows:

$$H = \left\{ x \in \mathbb{R}^d \,|\, x^T w = 0 \right\} .$$

WHICH SIDE OF THE HYPERPLANE ARE WE ON?



Distance from the hyperplane

- How close is a point x to H?
- Cosine rule: $x^T w = \|x\|_2 \|w\|_2 \cos\theta$
- The distance of x to the hyperplane is

$$\|x\|_2 \cdot |\cos\theta| = |x^T w| / \|w\|_2 .$$

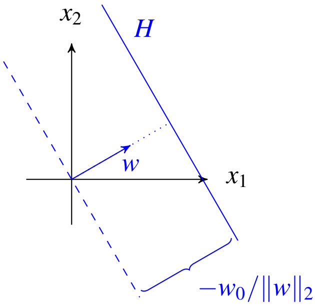
  So $|x^T w|$ gives a sense of distance.

Which side of the hyperplane?

- The cosine satisfies $\cos\theta > 0$ if $\theta \in (-\frac{\pi}{2}, \frac{\pi}{2})$.
- So the sign of $\cos(\cdot)$ tells us the side of H, and by the cosine rule

$$\operatorname{sign}(\cos\theta) = \operatorname{sign}(x^T w).$$

AFFINE HYPERPLANES



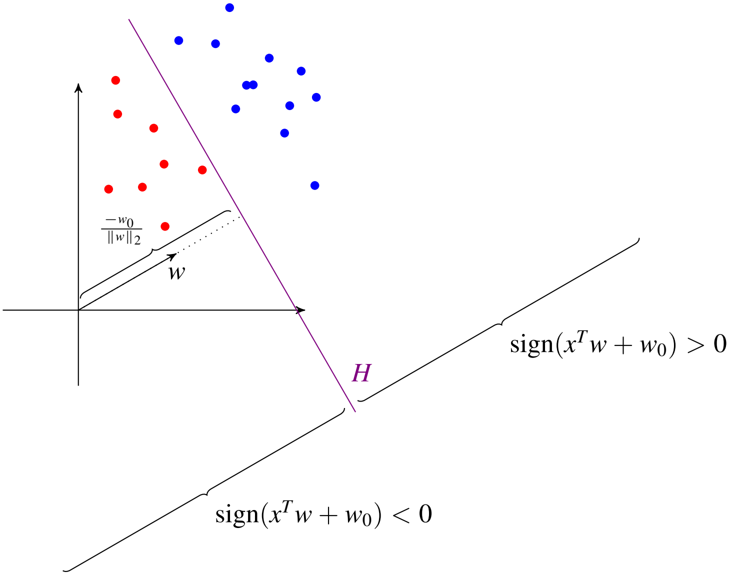
Affine Hyperplanes

- An *affine hyperplane* H is a hyperplane translated (shifted) using a scalar w_0.
- Think of: $H = x^T w + w_0 = 0$.
- Setting $w_0 > 0$ moves the hyperplane in the *opposite* direction of w. ($w_0 < 0$ in figure)

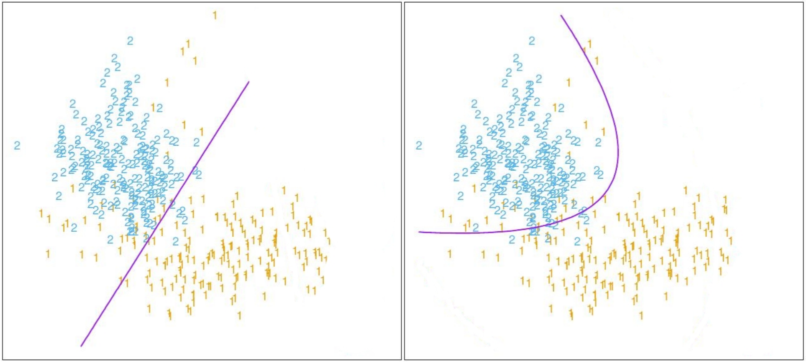
Which side of the hyperplane now?

- The plane has been shifted by distance $\frac{-w_0}{\|w\|_2}$ in the direction w.
- For a given (w, w_0) and input x the inequality $x^T w + w_0 > 0$ says that x is on the far side of an affine hyperplane H in the direction w points.

Classification with Affine Hyperplanes



POLYNOMIAL GENERALIZATIONS



The same generalizations from regression also hold for classification:

- (left) A linear classifier using $x = (x_1, x_2)$.
- (right) A linear classifier using $x = (x_1, x_2, x_1^2, x_2^2)$.
  The decision boundary is linear in $\mathbb{R}^4$, but isn't when plotted in $\mathbb{R}^2$.

Gaussian with different covariance

Let's look at the log odds for the general case where $p(x|y) = N(x|\mu_y, \Sigma_y)$ (i.e., now each class has its own covariance)

$$\ln \frac{p(x|y=1)P(y=1)}{p(x|y=0)P(y=0)} = \underbrace{\text{something complicated not involving } x}_{\text{a constant}} + \underbrace{x^T(\Sigma_1^{-1}\mu_1 - \Sigma_0^{-1}\mu_0)}_{\text{a part that's linear in } x} + \underbrace{x^T(\Sigma_0^{-1}/2 - \Sigma_1^{-1}/2)x}_{\text{a part that's quadratic in } x}$$

Also called "quadratic discriminant analysis," but it's *linear* in the weights.

Another Bayes classifier

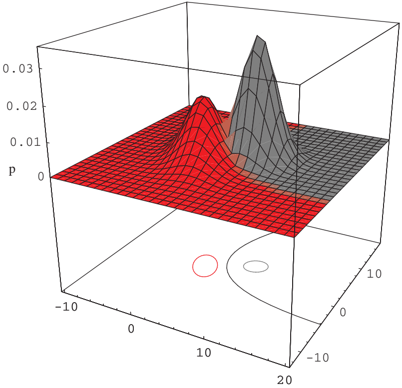
- We also saw this last lecture.
- Notice that

  $$f(x) = \text{sign}(x^T A x + x^T b + c)$$

  is linear in A, b, c.
- When $x \in \mathbb{R}^2$, rewrite as

  $$x \leftarrow (x_1, x_2, 2x_1x_2, x_1^2, x_2^2)$$

  and do linear classification in $\mathbb{R}^5$.



Whereas the Bayes classifier with shared covariance is a version of linear classification, using different covariances is like polynomial classification.

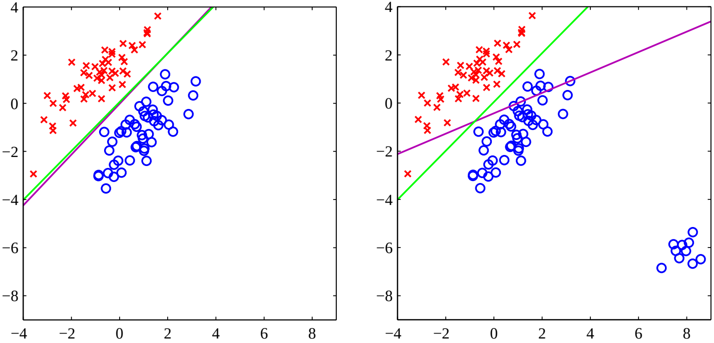
LEAST SQUARES ON $\{-1, +1\}$

How do we define more general classifiers of the form

$$f(x) = \text{sign}(x^T w + w_0)\,?$$

- One simple idea is to treat classification as a regression problem:
  1. Let $y = (y_1, \ldots, y_n)^T$, where $y_i \in \{-1, +1\}$ is the class of x_i.
  2. Add dimension equal to 1 to x_i and construct the matrix $X = [x_1, \ldots, x_n]^T$.
  3. Learn the least squares weight vector $w = (X^T X)^{-1} X^T y$.
  4. For a new point x_0 declare $y_0 = \text{sign}(x_0^T w) \longleftarrow w_0$ is included in w.
- Another option: Instead of LS, use ℓ_p regularization.
- These are "baseline" options. We can use them, along with k-NN, to get a quick sense what performance we're aiming to beat.

SENSITIVITY TO OUTLIERS

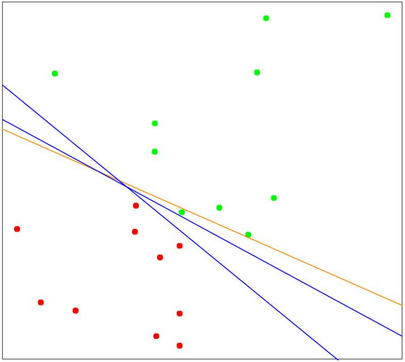


Least squares can do well, but it is sensitive to outliers. In general we can find better classifiers that focus more on the decision boundary.

- (left) Least squares (purple) does well compared with another method
- (right) Least squares does poorly because of outliers

THE PERCEPTRON ALGORITHM

EASY CASE: LINEARLY SEPARABLE DATA



(Assume data x_i has a 1 attached.)

Suppose there exists a linear classifier with zero *training* error:

$$y_i = \operatorname{sign}(x_i^T w), \quad \text{for all } i.$$

Then the data is linearly separable.

Left: Can separate classes with a line.
(Can find an infinite number of lines.)

PERCEPTRON (ROSENBLATT, 1958)

Using the linear classifier

$$y = f(x; w) = \text{sign}(x^T w),$$

the Perceptron seeks to minimize

$$\mathcal{L} = -\sum_{i=1}^{n} (y_i \cdot x_i^T w) \mathbb{1}\{y_i \neq \text{sign}(x_i^T w)\}.$$

Because $y \in \{-1, +1\}$,

$$y_i \cdot x_i^T w \quad \text{is} \quad \begin{cases} > 0 \ \text{ if } \ y_i = \text{sign}(x_i^T w) \\ < 0 \ \text{ if } \ y_i \neq \text{sign}(x_i^T w) \end{cases}$$

By minimizing $\mathcal{L}$ we're trying to always predict the correct label.

LEARNING THE PERCEPTRON

- Unlike other techniques we've talked about, we can't find the minimum of $\mathcal{L}$ by taking a derivative and setting to zero:

  $$\nabla_w \mathcal{L} = 0 \quad \text{cannot be solved for } w \text{ analytically.}$$

  However $\nabla_w \mathcal{L}$ does tell us the direction in which $\mathcal{L}$ is *increasing* in w.

- Therefore, for a sufficiently small η, if we update

  $$w' \leftarrow w - \eta \nabla_w \mathcal{L},$$

  then $\mathcal{L}(w') < \mathcal{L}(w)$ — i.e., we have a better value for w.

- This is a general method for trying to minimize an objective function called **gradient descent**. Perceptron uses a "stochastic" version of this.

Input: Training data $(x_1, y_1), \ldots, (x_n, y_n)$ and a positive step size η

1. **Initialize** $w^{(1)}$ in some way
2. **For iteration** $t = 1, 2, \ldots$ **do**
   a) **Search** for all examples $(x_i, y_i) \in \mathcal{D}$ such that $y_i \neq \text{sign}(x_i^T w^{(t)})$
   b) **If** such a (x_i, y_i) exists, randomly pick one and update

$$w^{(t+1)} = w^{(t)} + \eta y_i x_i,$$

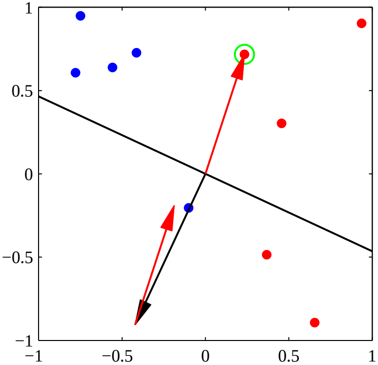
   Else: Return $w^{(t)}$ as the solution since everything is classified correctly.

If $\mathcal{M}_t$ indexes the misclassified observations at step t, then we have

$$\mathcal{L} = -\sum_{i=1}^{n} (y_i \cdot x_i^T w)\mathbb{1}\{y_i \neq \text{sign}(x_i^T w)\}, \qquad \nabla_w \mathcal{L} = -\sum_{i \in \mathcal{M}_t} y_i x_i \,.$$

The full gradient step is $w^{(t+1)} = w^{(t)} - \eta \nabla_w \mathcal{L}$. *Stochastic optimization* just picks out one element in $\nabla_w \mathcal{L}$—we could have also used the full summation.

Learning the perceptron

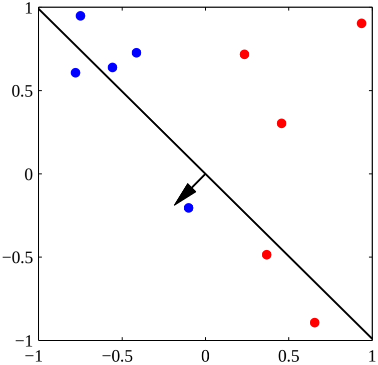


red $= +1$, blue $= -1$, $\eta = 1$

(This specific example sets $w_0 = 0$.)

1. Pick a misclassified (x_i, y_i)
2. Set $w \leftarrow w + \eta y_i x_i$

Learning the perceptron

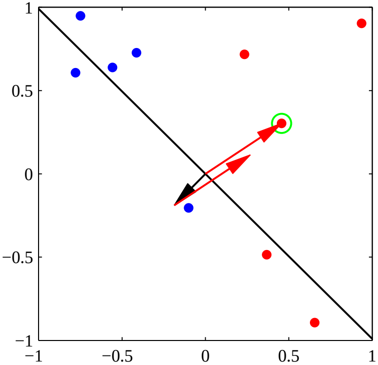


red $= +1$, blue $= -1$, $\eta = 1$

(This specific example sets $w_0 = 0$.)

The update to w defines a new decision boundary (hyperplane)

LEARNING THE PERCEPTRON

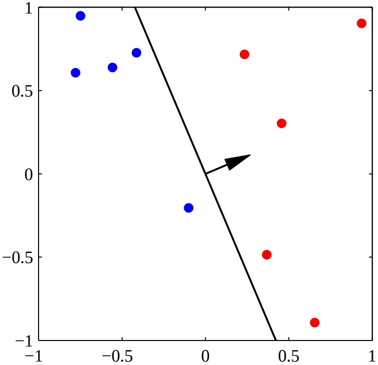


red $= +1$, blue $= -1$, $\eta = 1$

(This specific example sets $w_0 = 0$.)

1. Pick another misclassified (x_j, y_j)
2. Set $w \leftarrow w + \eta y_j x_j$

Learning the perceptron



red $= +1$, blue $= -1$, $\eta = 1$

(This specific example sets $w_0 = 0$.)

Again update w, i.e., the hyperplane

This time we're done.

Drawbacks of perceptron

The perceptron represents a first attempt at linear classification by directly learning an affine hyperplane defined by w. It has some drawbacks:

1. When the data is separable, there are an infinite $\#$ of hyperplanes.
   - We may think some are better than others, but this algorithm doesn't take "quality" into consideration. It converges to the first one it finds.
2. When the data isn't separable, the algorithm doesn't converge. The hyperplane of w is always moving around.
   - It's hard to detect this since it can take a long time for the algorithm to converge when the data is separable.

Later, we will discuss algorithms that use the same idea of learning an affine hyperplane w, but alter the objective function to fix these problems.