COMS 4721: Machine Learning for Data Science

Lecture 8, 2/9/2017

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

f: maps $x_i, \Theta \to -1$ or $+1$

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.

in a more general

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.

what do these mean?

 (Motivate linear classification from a bayes classifier.)

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 $(x_1, y_1), \ldots, (x_n, y_n)$.

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

likelihood of that x given it's coming from class 1

prior of class1.

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

$$\Updownarrow \quad \ln(1)$$

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

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

 (what does log odds actually look like for Bayes classifier.)

class conditional density is a multi-variate Gaussian with a class-specific mean but shared co-variance sigma. So both classes have co-variant sigma, but there's a class-specific mean.

Gaussian with shared covariance

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

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

calculate log odds, plug in the values for these distributions:

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

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

← does not involve x

→ does depend on x

1 if +ve
0 if -ve

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:

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

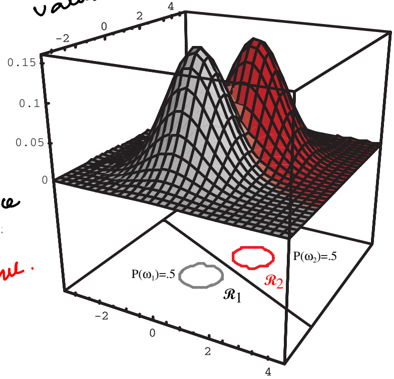
we declare the class to be the sign.

previous slides values

- 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 $\Sigma_1 = \Sigma_0$.

  shared covariance

  always true.
- w and w_0 are obtained through a specific equation.



This Bayes classifier is one instance of a linear classifier

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

where

the form of w & w_0 is restricted to having to be functions of class specific means

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

Shared covariance class wide

$$w = \Sigma^{-1}(\mu_1 - \mu_0)$$

With MLE used to find values for π_y, μ_y and Σ. → didn't understand the MLE reference.

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 pick a more complex p(y|x). Then we may not have a linear classifier.

 [Define binary linear classifier in general framework.]

Definition: Binary linear classifier

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

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

This assumes that there is a property in our data set called linear separability. For this to perform well, our data has to be linearly separable in the space x lives in

where $w \in \mathbb{R}^d$ and $w_0 \in \mathbb{R}$. Since the goal is 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

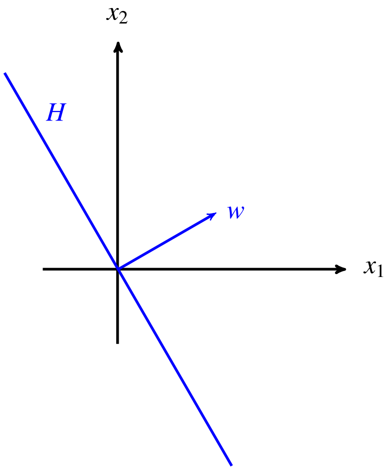
$$x^T w + w_0 \begin{cases} > 0 & \text{if } x \in A \ \ (\text{e.g, class } +1) \\ < 0 & \text{if } x \in B \ \ (\text{e.g, class } -1) \end{cases}$$

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

HYPERPLANES (help to understand what exactly that a linear classifier is trying to do with the data.)

Geometric interpretation of linear classifiers:

Every vector w in $\mathbb{R}^d$ defines hyperplane in $\mathbb{R}^{d-1}$.



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

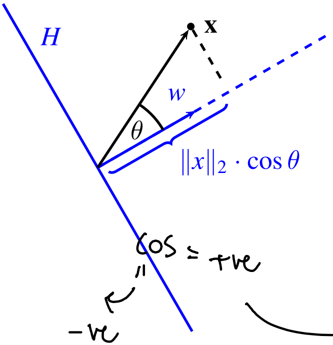
- A $\mathbb{R}^2$-hyperplane is a line.
- A $\mathbb{R}^3$-hyperplane is a plane.
- As a linear subspace, a hyperplane always contains the origin.

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

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

[give me all vectors x that are orthogonal to w.]

WHICH SIDE OF THE PLANE ARE WE ON? for an arbitrary point.



Distance from the plane

- How close is a point x to H? (Euclidean distance)
- Cosine rule: $x^T w = \|x\|_2 \|w\|_2 \cos\theta$ — 1
- 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?

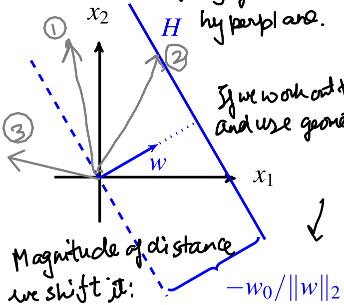
- The cosine satisfies $\cos\theta > 0$ if $\theta \in (-\frac{\pi}{2}, \frac{\pi}{2})$.
- So the sign of $\cos(\cdot)$ tells us the side of H, and by the cosine rule

$$\operatorname{sign}(\cos\theta) = \operatorname{sign}(x^T w).$$ from 1

 (just a shifted hyperplane)

→ we want to shift it either parallel in the dir. of w or in the opp. direction of w so that we can separate our data more easily.

w defines the hyperplane, w_0 then decides the shifting of the hyperplane.



Magnitude of distance we shift it:

If we work out the math and use geometry again.

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)

same direction

*

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.

* In this case, w_0 would be -ve no., which is shifting the hyperplane in the dir^n that w is pointing.

① Consider any vector x that's on the right of the dotted line and the left of the solid line. We know from the previous slide, that the dot product of a pt. x with w is going to be +ve. However, w_0 in this case is also -ve. So for all points within the slab → the +ve amt. that you add from the dot product is not greater than the -ve amount that you add from w_0. And so the net sum of those 2 values is still -ve.

② If I took the dot product of this vector with w, that would be equal to the -ve of w_0. I would add these 2 together I would get 0.

③ Every single pt. on that the left of the dotted line has a -ve dot product with w and then I'm adding a -ve no. and so of course, that is also -ve.

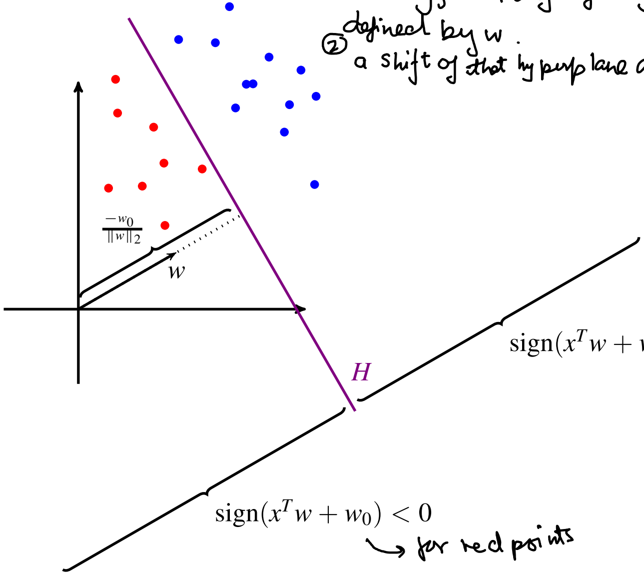
So we have affine hyperplane, w_0 is shifting the hyperplane, such that now it's this line that is defining what side is positive and what side is negative.

H

Classification with Affine Hyperplanes

① Asking for an angle of the hyperplane defined by w.

② a shift of that hyperplane defined by w_0

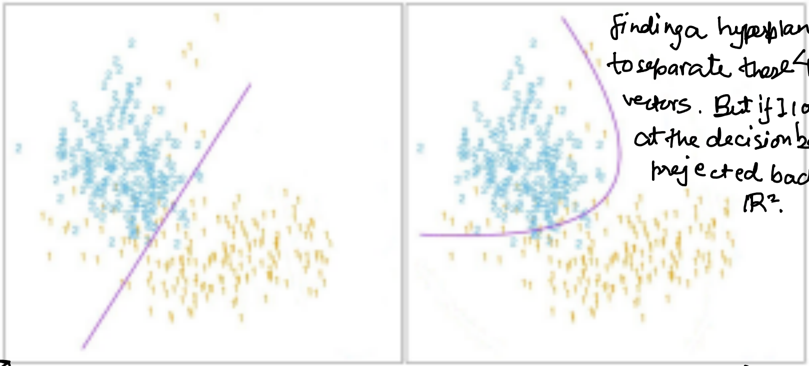


→ for blue points

→ for red points

Polynomial generalizations

(Generalises to non-linear hyperplanes using polynomial generalizations.)



Finding a hyperplane in $\mathbb{R}^4$ to separate these 4-dim vectors. But if I look at the decision boundary projected back to $\mathbb{R}^2$.

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 $\mathbb{R}^2$.

covariance specific to that class

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)

Assume that we 2 classes, class 0 & class 1.

$$\ln \frac{p(x|y=1)P(y=1)}{p(x|y=0)P(y=0)} = \underbrace{\text{something complicated not involving } x}_{\text{a constant}} \rightarrow C$$

$$+ \underbrace{x^T(\Sigma_1^{-1}\mu_1 - \Sigma_0^{-1}\mu_0)}_{\text{a part that's linear in } x} \rightarrow b$$

$$+ \underbrace{x^T(\Sigma_0^{-1}/2 - \Sigma_1^{-1}/2)x}_{\text{a part that's quadratic in } x} \rightarrow A$$

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

Another Bayes classifier

- We also saw this last lecture.
- Notice that

+ve, class 1
-ve, class 0

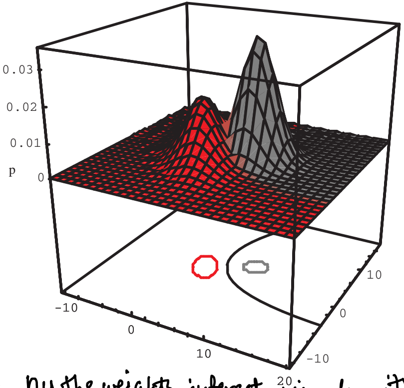
$$f(x) = \text{sign}(x^T A x + x^T b + c)$$

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 $\mathbb{R}^5$.



This is linear in all of the unknown weights. All the weights interact linearly with our data.

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

What we are actually doing with this quadratic function: is a linear classifier in higher dimensional problem. And so, that's where we see that learning a linear classifier in that 5 dimensional space maps to a non-linear function in the original 2 dimensional space.

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

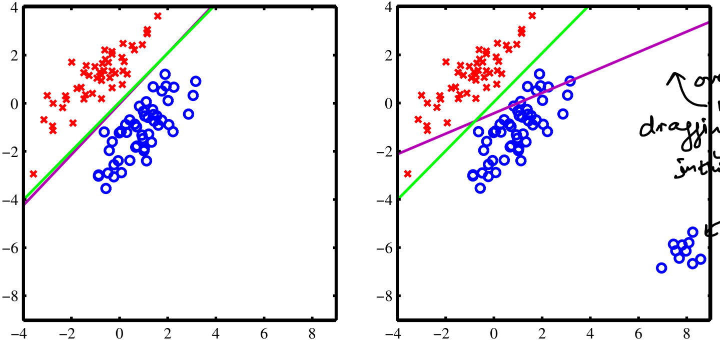
How do we define more general classifiers of the form

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

- One simple idea is to treat classification as a regression problem:
  1. Let $y = (y_1, \dots, y_n)^T$, 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 $X = [x_1, \dots, x_n]^T$.
  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

The issue with treating binary classification as a regression problem is that it's very sensitive to outliers.



overfit by dragging the line in this way.

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.

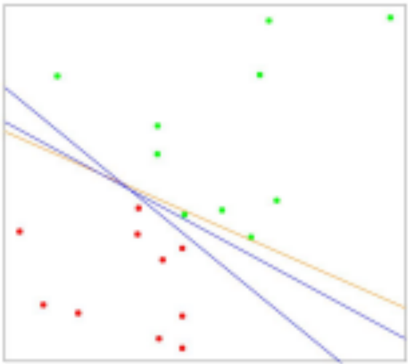
covariance? How does that effect it?

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

We want a method that's more robust. We want a method that doesn't care so much what values of the covariance x are. We care just what side of hyperplane they are

The Perceptron Algorithm

EASY CASE: LINEARLY SEPARABLE DATA



(Assume data x_i has a 1 attached.)

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

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

Then the data is "linearly separable"

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

PERCEPTRON (ROSENBLATT, 1958)

Assumes the classes are linearly separable. (not a good thing)

Using the linear classifier

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

the Perceptron seeks to minimize

objective function: *

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

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

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

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

* Summing up this value over all points we predict incorrectly acc. to some vector W. And, all of those points are going to be negative. And so, the sum of those incorrectly classified points will be -ve. We will put a -ve sign out front to make it +ve; and now we minimize that.

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

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

So we need some sort of an iterative algo.

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

The derivative at a particular point pt says move in this direction if you want to increase the objective func.

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

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

(opposite dir.^n of derivative scaled by tiny value

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

- This is a very general method for optimizing an objective functions called **gradient descent**. Perceptron uses a "stochastic" version of this.

LEARNING THE PERCEPTRON

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

1. **Set** $w^{(1)} = \vec{0}$
2. **For step** $t = 1, 2, \dots$ **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

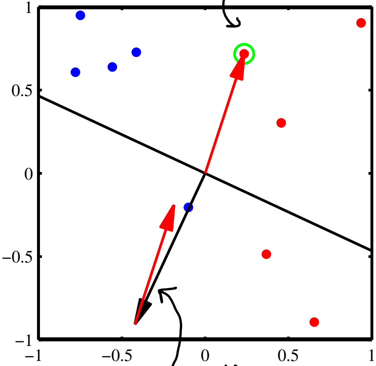
$$\mathcal{L} = -\sum_{i=1}^{n} \overbrace{(y_i \cdot x_i^T w)\mathbb{1}\{y_i \neq \text{sign}(x_i^T w)\}}^{\text{original objective f}^{n}}, \qquad \nabla_w \mathcal{L} = -\overset{\text{only wrong sample}}{\sum_{i \in \mathcal{M}_t}} y_i x_i \,.$$

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.

(randomly) instead of all of them.

LEARNING THE PERCEPTRON (example)

misclassified example (misclassified as blue class)

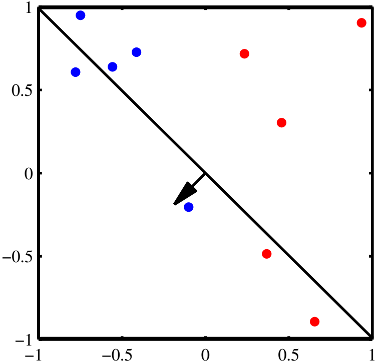


starting vector.

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

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

LEARNING THE PERCEPTRON

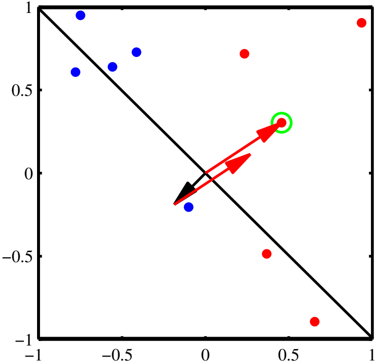


red $= +1$, blue $= -1$, $\eta = 1$

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

LEARNING THE PERCEPTRON

.

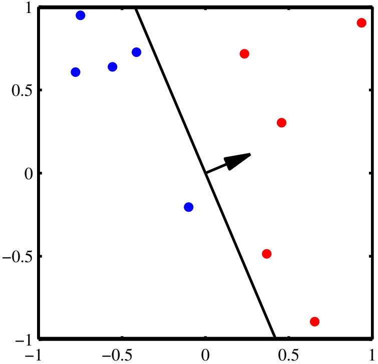


red $= +1$, blue $= -1$, $\eta = 1$

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

LEARNING THE PERCEPTRON

converged.



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

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 the hyperplane defined by w. It has some drawbacks:

1. When the data is separable, there are an infinite # of hyperplanes. that can separate the data,
   - 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. (Just continue to keep picking misclassified examples & updating the hyperplane but it's never going to converge.)
   - 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 directly learning the hyperplane w, but alters the objective function to fix these problems.