ColumbiaX: Machine Learning
Lecture 7

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

CLASSIFICATION

Terminology and Notation

Input: As with regression, in a *classification problem* we start with measurements $x_1, \ldots, x_n$ in an input space $\mathcal{X}$. (Again think $\mathcal{X} = \mathbb{R}^d$)

Output: The *discrete* output space $\mathcal{Y}$ is composed of K possible *classes*:

- $\mathcal{Y} = \{-1, +1\}$ or $\{0, 1\}$ is called binary classification.
- $\mathcal{Y} = \{1, \ldots, K\}$ is called multiclass classification

Instead of a real-valued response, classification assigns x to a category.

- Regression: For pair (x, y), y is the response of x.
- Classification: For pair (x, y), y is the class of x.

Classification problem

Defining a classifier

Classification uses a function f (called a *classifier*) to map input x to class y.

$$y = f(x) \ : \ f \text{ takes in } x \in \mathcal{X} \text{ and declares its class to be } y \in \mathcal{Y}$$

As with regression, the problem is two-fold:

- Define the classifier f and its parameters.
- Learn the classification rule using a training set of “labeled data.”

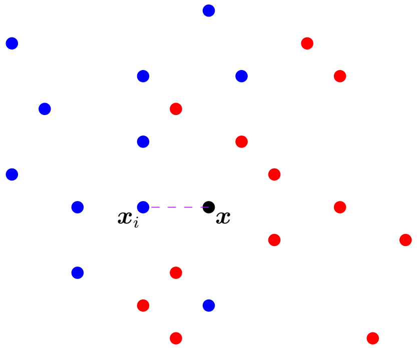
NEAREST NEIGHBOR CLASSIFIERS

Nearest neighbor (NN) classifier

Given data $(x_1, y_1), \ldots, (x_n, y_n)$, construct classifier $\hat{f}(x) \to y$ as follows:

For an input x not in the training data,

1. Let x_i be the point among $x_1, x_2, \ldots, x_n$ that is "closest" to x.
2. Return its label y_i.



DISTANCES

Question: How should we measure distance between points?

The default distance for data in $\mathbb{R}^d$ is the Euclidean one:

$$\|u - v\|_2 = \left(\sum_{i=1}^{d} (u_i - v_i)^2 \right)^{\frac{1}{2}} \quad \text{(line-of-sight distance)}$$

But there are other options that may sometimes be better:

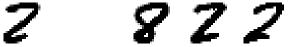
- ℓ_p for $p \in [1, \infty]$: $\|u - v\|_p = \left(\sum_{i=1}^{d} |u_i - v_i|^p \right)^{\frac{1}{p}}$.
- Edit distance (for strings): How many add/delete/substitutions are required to transform one string to the other.
- Correlation distance (for signal): Measures how correlated two vectors are for signal detection.

Example: OCR with NN classifier

- **Handwritten digits data**: grayscale 28×28 images, treated as vectors in $\mathbb{R}^{784}$, with labels indicating the digit they represent.

- Split into training set $\mathcal{S}$ (60K points) and testing set $\mathcal{T}$ (10K points).
- **Training error**: $\text{err}(\hat{f}, \mathcal{S}) = 0 \leftarrow$ declare its class to be its own class!
  Test error: $\text{err}(\hat{f}, \mathcal{T}) = 0.0309 \leftarrow$ using ℓ_2 distance
- Examples of mistakes: (left) test point, (right) nearest neighbor in $\mathcal{S}$:

- **Observation**: First mistake might have been avoided by looking at three nearest neighbors (whose labels are '8', '2', '2') …



test point three nearest neighbors

k-NEAREST NEIGHBORS CLASSIFIER

Given data $(x_1, y_1), \ldots, (x_n, y_n)$, construct the k-NN classifier as follows:

For a new input x,

1. Return the k points closest to x, indexed as $x_{i_1}, \ldots, x_{i_k}$.
2. Return the majority-vote of $y_{i_1}, y_{i_2}, \ldots, y_{i_k}$.

(Break ties in both steps arbitrarily.)

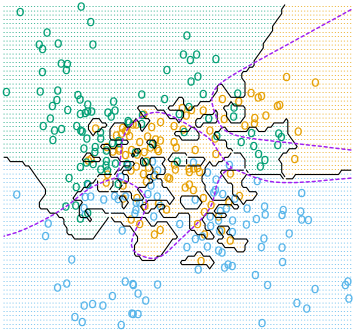
Example: OCR with k-NN classifier

k	1	3	5	7	9
$\text{err}(\hat{f}_k, T)$	0.0309	0.0295	0.0312	0.0306	0.0341

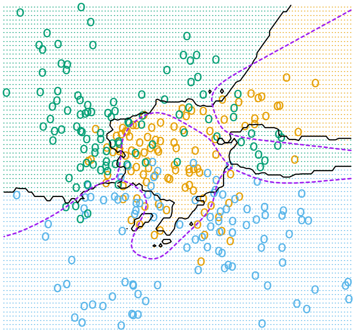
Effect of k

In general:

- Smaller $k \Rightarrow$ smaller training error.
- Larger $k \Rightarrow$ predictions are more "stable" due to voting.



1-NN



15-NN

Purple dotted lines : Can ignore for now.
Black solid lines : k-NN's decision boundaries.

STATISTICAL SETTING

How do we measure the quality of a classifier?

For any classifier we care about two sides of the same coin:

- Prediction accuracy: $P(f(x) = y)$.
- Prediction error: $\mathrm{err}(f) = P(f(x) \neq y)$.

To calculate these values, we assume there is a distribution $\mathcal{P}$ over the space of labeled examples generating the data

$$(x_i, y_i) \overset{iid}{\sim} \mathcal{P}, \qquad i = 1, \ldots, n.$$

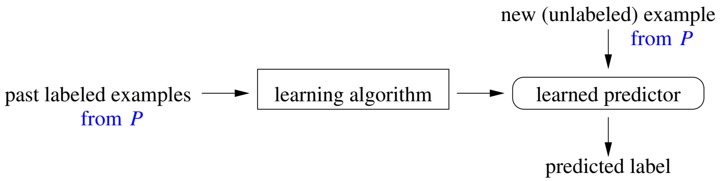
We don't know what $\mathcal{P}$ is, but can still talk about it in abstract terms.

Statistical learning

When is there any hope for finding an accurate classifier?

Key assumption: Data $(x_1, y_1), \ldots, (x_n, y_n)$ are i.i.d. random labeled examples with distribution $\mathcal{P}$.

This assumption allows us to say that the past should look like the future.



Regression makes similar assumptions.

BAYES CLASSIFIERS

Optimal classifiers

Can we talk about what an "optimal" classifier looks like?

Assume that $(X, Y) \overset{iid}{\sim} \mathcal{P}$. (Again, we don't know $\mathcal{P}$)

Some probability equalities with $\mathcal{P}$:

1. The expectation of an indicator of an event is the probability of the event, e.g.,

$$\mathbb{E}_P[\mathbb{1}(Y = 1)] = P(Y = 1), \quad \leftarrow \mathbb{1}(\cdot) = 0 \text{ or } 1 \text{ depending if } \cdot \text{ is true}$$

2. Conditional expectations can be random variables, and their expectations remove the randomness,

$$C = \mathbb{E}[\,A \,|\, B\,] : \quad A \text{ and } B \text{ are } \textit{both} \text{ random, so } C \text{ is random}$$

$$\mathbb{E}[\,C\,] = \mathbb{E}[\,\mathbb{E}[A \,|\, B\,]\,] = \mathbb{E}[\,A\,] \quad \text{“tower property” of expectation}$$

OPTIMAL CLASSIFIERS

For any classifier $f\colon \mathcal{X} \to \mathcal{Y}$, its prediction error is

$$P(f(X) \neq Y) = \mathbb{E}[\mathbb{1}(f(X) \neq Y)] = \mathbb{E}[\underbrace{\mathbb{E}[\mathbb{1}(f(X) \neq Y)\,|X]}_{\text{a random variable}}] \qquad (\dagger)$$

OPTIMAL CLASSIFIERS

For any classifier $f\colon \mathcal{X} \to \mathcal{Y}$, its prediction error is

$$P(f(X) \neq Y) = \mathbb{E}[\mathbb{1}(f(X) \neq Y)] = \mathbb{E}[\underbrace{\mathbb{E}[\mathbb{1}(f(X) \neq Y)\,|X]}_{\text{a random variable}}] \qquad (\dagger)$$

For each $x \in \mathcal{X}$,

$$\mathbb{E}[\mathbb{1}(f(X) \neq Y)\,|\,X = x] = \sum_{y \in \mathcal{Y}} P(Y = y\,|\,X = x) \cdot \mathbb{1}(f(x) \neq y), \qquad (\ddagger)$$

Optimal classifiers

For any classifier $f\colon \mathcal{X} \to \mathcal{Y}$, its prediction error is

$$P(f(X) \neq Y) = \mathbb{E}[\mathbb{1}(f(X) \neq Y)] = \mathbb{E}[\underbrace{\mathbb{E}[\mathbb{1}(f(X) \neq Y) \,|X]}_{\text{a random variable}}] \tag{$\dagger$}$$

For each $x \in \mathcal{X}$,

$$\mathbb{E}[\mathbb{1}(f(X) \neq Y) \,|\, X = x] = \sum_{y \in \mathcal{Y}} P(Y = y \,|\, X = x) \cdot \mathbb{1}(f(x) \neq y), \tag{$\ddagger$}$$

The above quantity ($\ddagger$) is minimized for this particular $x \in \mathcal{X}$ when

$$f(x) = \arg\max_{y \in \mathcal{Y}} \; P(Y = y|X = x). \tag{$\star$}$$

The classifier f with property ($\star$) for all $x \in \mathcal{X}$ is called the *Bayes classifier*, and it has the smallest prediction error ($\dagger$) among *all classifiers*.

The Bayes classifier

Under the assumption $(X, Y) \overset{iid}{\sim} \mathcal{P}$, the optimal classifier is

$$f^\star(x) := \arg\max_{y \in \mathcal{Y}} P(Y = y | X = x).$$

From Bayes rule we equivalently have

$$f^\star(x) = \arg\max_{y \in \mathcal{Y}} \underbrace{P(Y = y)}_{class\ prior} \times \underbrace{P(X = x | Y = y)}_{data\ likelihood\ |\ class}.$$

- $P(Y = y)$ is called the *class prior*.
- $P(X = x | Y = y)$ is called the *class conditional distribution* of X.
- In practice we don't know either of these, so we approximate them.

Aside: If X is a continuous-valued random variable, replace $P(X = x | Y = y)$ with *class conditional density* $p(x | Y = y)$.

Example: Gaussian class conditional densities

Suppose $\mathcal{X} = \mathbb{R}$, $\mathcal{Y} = \{0, 1\}$, and the distribution $\mathcal{P}$ of (X, Y) is as follows.

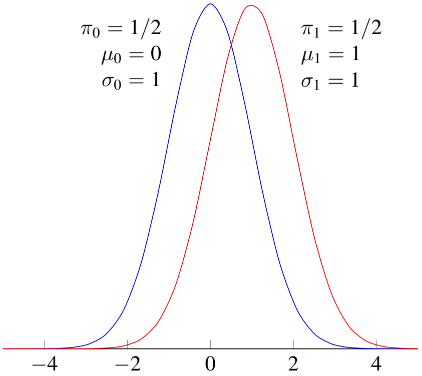
- **Class prior**: $P(Y = y) = \pi_y, \quad y \in \{0, 1\}$.
- **Class conditional density** for class $y \in \{0, 1\}$: $p_y(x) = N(x|\mu_y, \sigma_y^2)$.
- **Bayes classifier**:

$$
\begin{aligned}
f^\star(x) &= \underset{y \in \{0,1\}}{\operatorname{argmax}} \; p(X = x|Y = y)P(Y = y) \\
&= \begin{cases} 1 & \text{if } \frac{\pi_1}{\sigma_1} \exp\left[-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right] > \frac{\pi_0}{\sigma_0} \exp\left[-\frac{(x-\mu_0)^2}{2\sigma_0^2}\right] \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

This type of classifier is called a *generative* model.

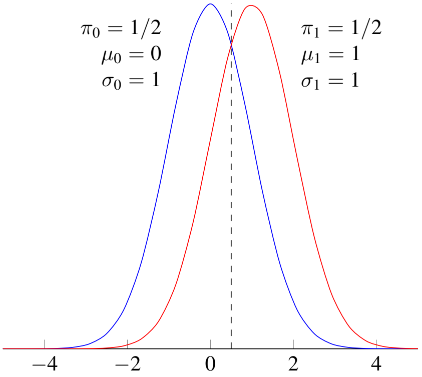
- **Generative model**: Model x and y with distributions.
- **Discriminative model**: Plug x into a distribution on y (used thus far).

Example: Gaussian class conditional densities



$1/2$ of x's from $N(0,1) \rightarrow y = 0$
$1/2$ of x's from $N(1,1) \rightarrow y = 1$

EXAMPLE: GAUSSIAN CLASS CONDITIONAL DENSITIES

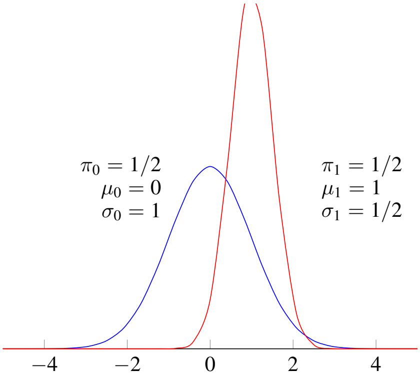


$1/2$ of x's from $N(0,1) \to y = 0$
$1/2$ of x's from $N(1,1) \to y = 1$

Bayes classifier:

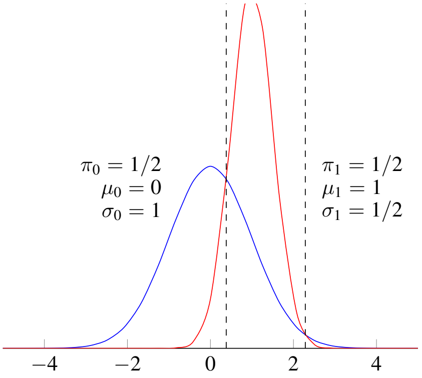
$$f^\star(x) = \begin{cases} 1 & \text{if } x > 1/2; \\ 0 & \text{otherwise.} \end{cases}$$

EXAMPLE: GAUSSIAN CLASS CONDITIONAL DENSITIES



$1/2$ of x's from $\mathcal{N}(0,1) \to y = 0$
$1/2$ of x's from $\mathcal{N}(1,1/4) \to y = 1$

Example: Gaussian class conditional densities



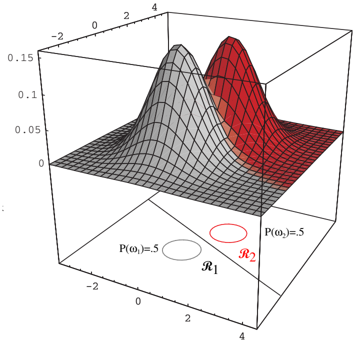
$1/2$ of x's from $\mathcal{N}(0,1) \to y = 0$
$1/2$ of x's from $\mathcal{N}(1,1/4) \to y = 1$

Bayes classifier:

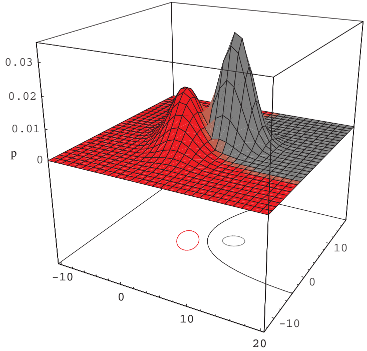
$$f^\star(x) = \begin{cases} 1 & \text{if } x \in [0.38, 2.29]; \\ 0 & \text{otherwise.} \end{cases}$$

EXAMPLE: MULTIVARIATE GAUSSIANS

Data: $\mathcal{X} = \mathbb{R}^2$, Label: $\mathcal{Y} = \{0, 1\}$
Class conditional densities are Gaussians in $\mathbb{R}^2$ with covariance Σ_0 and Σ_1.

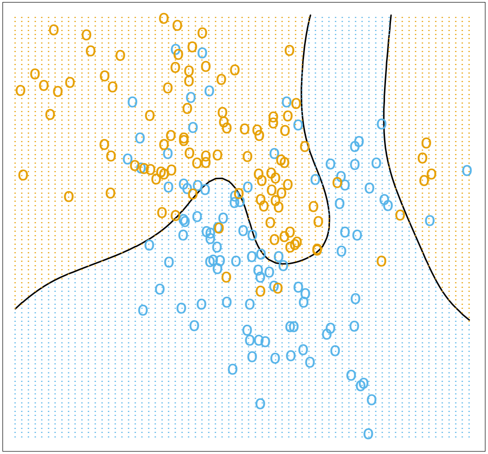


$\Sigma_0 = \Sigma_1$
Bayes classifier:
linear separator



$\Sigma_0 \neq \Sigma_1$
Bayes classifier:
quadratic separator

Bayes classifier in general



In general, the Bayes classifier may be rather complicated!
This one uses more than a single Gaussian for the class-conditional density.

Plug-in classifiers

Bayes classifier

The Bayes classifier has the smallest prediction error of *all* classifiers.

Problem: We can't construct the Bayes classifier without knowing $\mathcal{P}$.

- What is $P(Y = y|X = x)$, or equiv., $P(X = x|Y = y)$ and $P(Y = y)$?
- All we have are labeled examples drawn from the distribution $\mathcal{P}$.

Plug-in classifiers

Use the available data to approximate $P(Y = y)$ and $P(X = x|Y = y)$.

- Of course, the result may no longer give the best results among all the classifiers we can choose from (e.g., k-NN and those discussed later).

Example: Gaussian class conditional densities

Here, $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{1, \ldots, K\}$. Estimate Bayes classifier via MLE:

- **Class priors**: The MLE estimate of π_y is $\hat{\pi}_y = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}(y_i = y)$.
- **Class conditional density**: Choose $p(x|Y = y) = N(x|\mu_y, \Sigma_y)$. The MLE estimate of (μ_y, Σ_y) is

$$\hat{\mu}_y = \frac{1}{n_y} \sum_{i=1}^{n} \mathbb{1}(y_i = y) x_i,$$

$$\hat{\Sigma}_y = \frac{1}{n_y} \sum_{i=1}^{n} \mathbb{1}(y_i = y)(x_i - \hat{\mu}_y)(x_i - \hat{\mu}_y)^T.$$

  This is just the empirical mean and covariance of class y.

- **Plug-in classifier**:

$$\hat{f}(x) = \arg\max_{y \in \mathcal{Y}} \; \hat{\pi}_y |\hat{\Sigma}_y|^{-\frac{1}{2}} \exp\left\{ -\frac{1}{2}(x - \hat{\mu}_y)^T \hat{\Sigma}_y^{-1} (x - \hat{\mu}_y) \right\}.$$

Example: Spam Filtering

Representing emails

- **Input**: x, a vector of word counts. For example, if index $\{j \rightarrow \text{"car"}\}$ $x(j) = 3$ means that the word "car" occurs three times in the email.
- **Output**: $\mathcal{Y} = \{-1, +1\}$. Map $\{\text{email} \rightarrow -1, \text{spam} \rightarrow +1\}$

Example dimensions

	george	you	your	hp	free	work	!	our	re	click	remove
spam	0	4	1	0	4	0	5	5	1	3	2
email	1	3	4	1	1	4	0	1	1	0	0

Using a Bayes classifier

$$f(x) = \underset{y \in \{-1,+1\}}{\operatorname{argmax}} \; p(x|Y = y)P(Y = y)$$

NAIVE BAYES

We have to *define* $p(X = x|Y = y)$.

Simplifying assumption

Naive Bayes is a Bayes classifier that makes the assumption

$$p(X = x|Y = y) = \prod_{j=1}^{d} p_j(x(j)|Y = y),$$

i.e., it treats the dimensions of X as *conditionally independent* given y.

In spam example

- Correlations between words is ignored.
- Can help make it easier to define the distribution.

ESTIMATION

Class prior

The distribution $P(Y = y)$ is again easy to estimate from the training data:

$$P(Y = y) \; = \; \frac{\#\text{observations in class } y}{\#\text{observations}}$$

Class-conditional distributions

For the spam model we define

$$P(X = x|Y = y) \; = \; \prod_j p_j(x(j)|Y = y) \; = \; \prod_j \text{Poisson}(x(j)|\lambda_j^{(y)})$$

We then approximate each $\lambda_j^{(y)}$ from the data. For example, the MLE is

$$\lambda_j^{(y)} = \frac{\#\text{unique uses of word } j \text{ in observations from class } y}{\#\text{observations in class } y}$$

ColumbiaX: Machine Learning
Lecture 8

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

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.

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

$$
\begin{aligned}
p(x|y=1)\underbrace{P(y=1)}_{\pi_1} \quad &> \quad p(x|y=0)\underbrace{P(y=0)}_{\pi_0} \\
&\Updownarrow \\
\ln \frac{p(x|y=1)P(y=1)}{p(x|y=0)P(y=0)} \quad &> \quad 0
\end{aligned}
$$

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

A Bayes classifier

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)

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

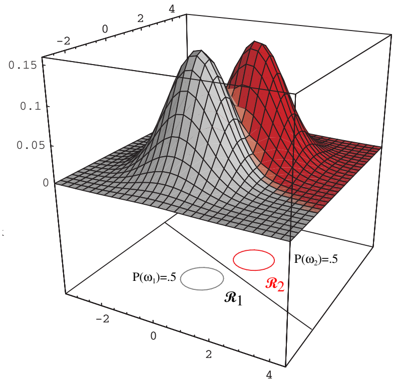
This is also called "linear discriminant analysis" (used to be called LDA).

A Bayes classifier

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

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

- 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 MLE 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.

Linear classifiers (binary case)

Definition: Binary linear classifier

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

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

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

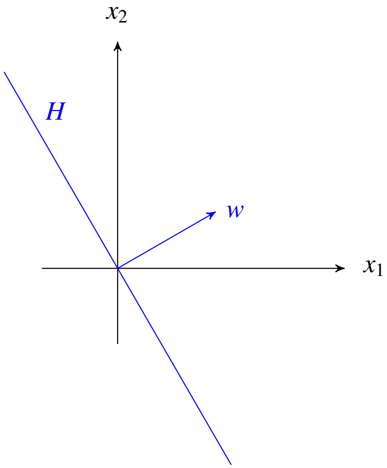
Two sets $A, B \subset \mathbb{R}^d$ are called *linearly separable* if

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

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

HYPERPLANES

Geometric interpretation of linear classifiers:



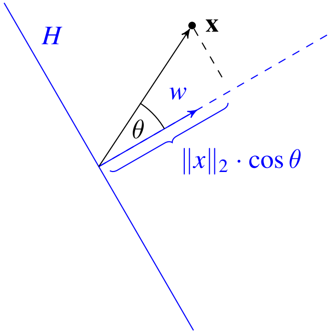
A *hyperplane* in $\mathbb{R}^d$ is a linear subspace of dimension $(d-1)$.

- 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:

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

WHICH SIDE OF THE PLANE ARE WE ON?



Distance from the plane

- 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

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

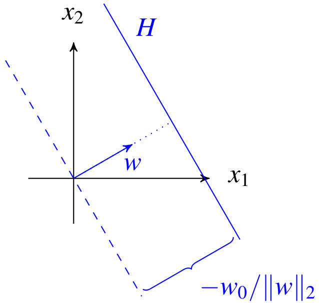
So $|x^T w|$ gives a sense of distance.

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

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

Affine Hyperplanes



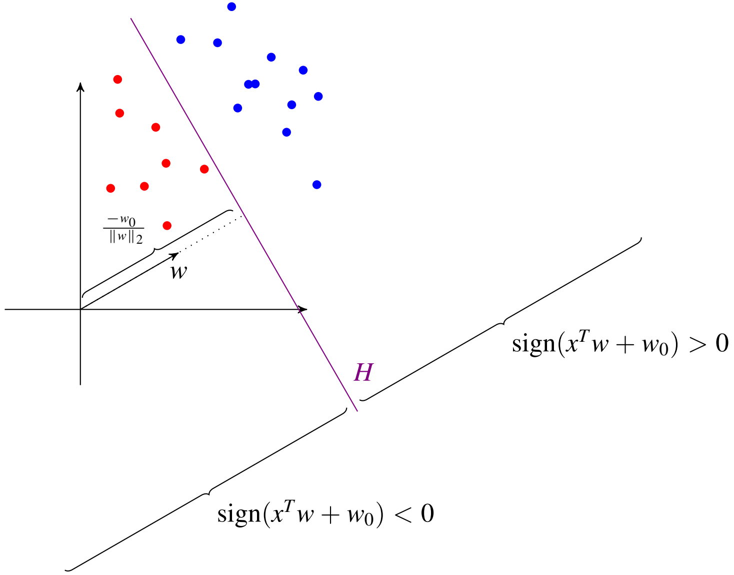
Affine Hyperplanes

- 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?

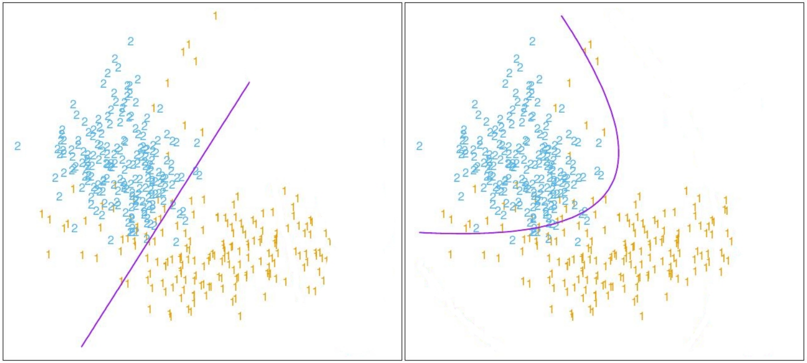
- The plane has been shifted by distance $\frac{-w_0}{\|w\|_2}$ in the direction w.
- For a given w, w_0 and input x the inequality $x^T w + w_0 > 0$ says that x is on the far side of an affine hyperplane H in the direction w points.

Classification with Affine Hyperplanes



$\frac{-w_0}{\|w\|_2}$
w
H
$\text{sign}(x^T w + w_0) > 0$
$\text{sign}(x^T w + w_0) < 0$

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 $\mathbb{R}^2$.

ANOTHER BAYES CLASSIFIER

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)

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

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

Another Bayes classifier

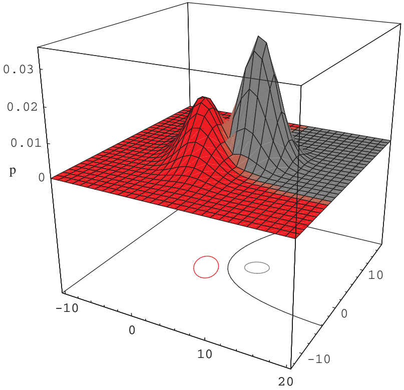
- We also saw this last lecture.
- Notice that

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

  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.

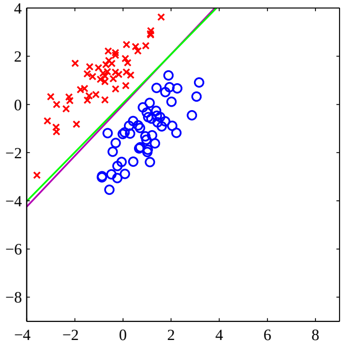
Least squares on $\{-1, +1\}$

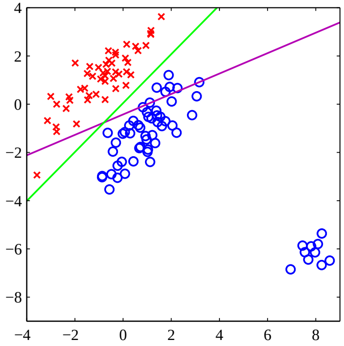
How do we define more general classifiers of the form

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

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



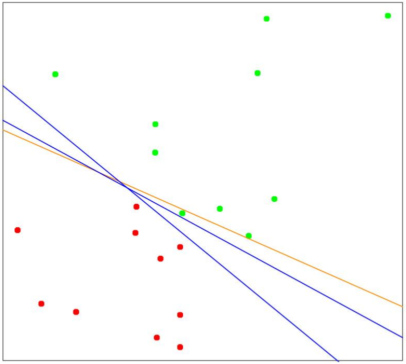


Least squares can do well, but it is sensitive to outliers. In general we can find better classifiers that focus more on the decision boundary.

- (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 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)

Using the linear classifier

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

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

LEARNING THE PERCEPTRON

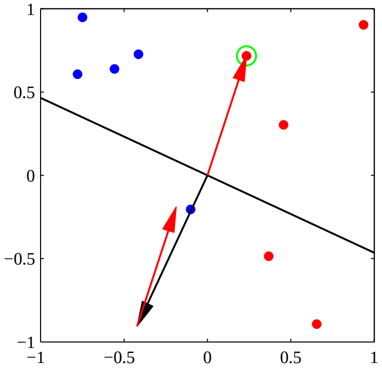
```
Input: Training data $(x_1, y_1), \dots, (x_n, y_n)$ and a positive step size $\eta$
  1. Set $w^{(1)} = \vec{0}$
  2. For step $t = 1, 2, \dots$ do
      a) Search for all examples $(x_i, y_i) \in \mathcal{D}$ such that $y_i \neq \text{sign}(x_i^T w^{(t)})$
      b) If such a $(x_i, y_i)$ exists, 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

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

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.

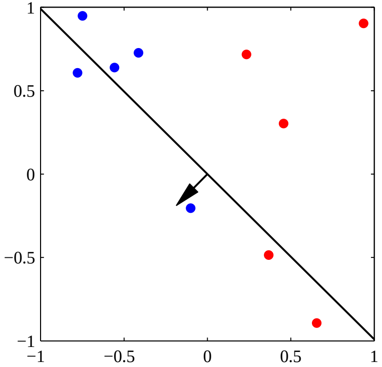
LEARNING THE PERCEPTRON



$\text{red} = +1, \quad \text{blue} = -1, \quad \eta = 1$

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

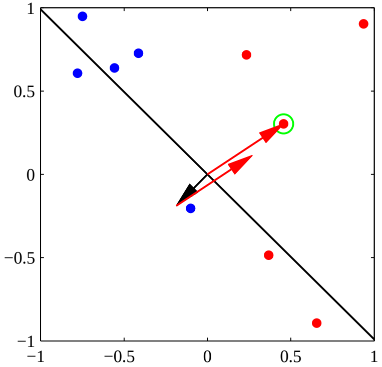
Learning the perceptron



$\color{red}{\text{red}} = +1, \quad \color{blue}{\text{blue}} = -1, \quad \eta = 1$

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

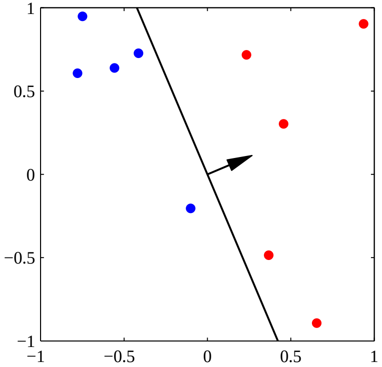
Learning the perceptron



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



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.
   - 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 directly learning the hyperplane w, but alters the objective function to fix these problems.