2 Linear Classifiers and Perceptrons

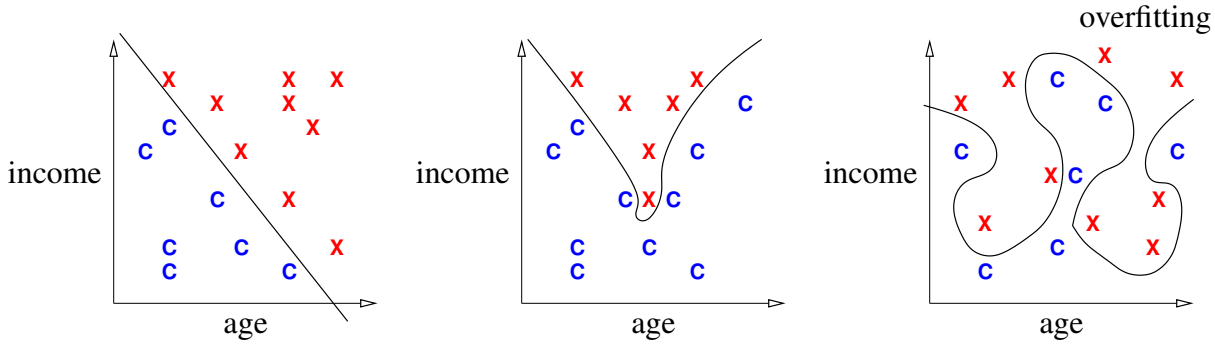
CLASSIFIERS

You are given sample of n observations, each with d features [aka predictors].
Some observations belong to class C; some do not.

Example: Observations are bank loans
Features are income & age ($d = 2$)
Some are in class "defaulted," some are not

Goal: Predict whether future borrowers will default,
based on their income & age.

Represent each observation as a point in d-dimensional space,
called a sample point / a feature vector / independent variables.



[Draw this by hand; decision boundaries last. classify3.pdf]

[We draw these lines/curves separating C's from X's. Then we use these curves to predict which future borrowers will default. In the last example, though, we're probably overfitting, which could hurt our predictions.]

decision boundary: the boundary chosen by our classifier to separate items in the class from those not.

overfitting: When sinuous decision boundary fits sample points so well that it doesn't classify future points well.

[A reminder that underlined phrases are definitions, worth memorizing.]

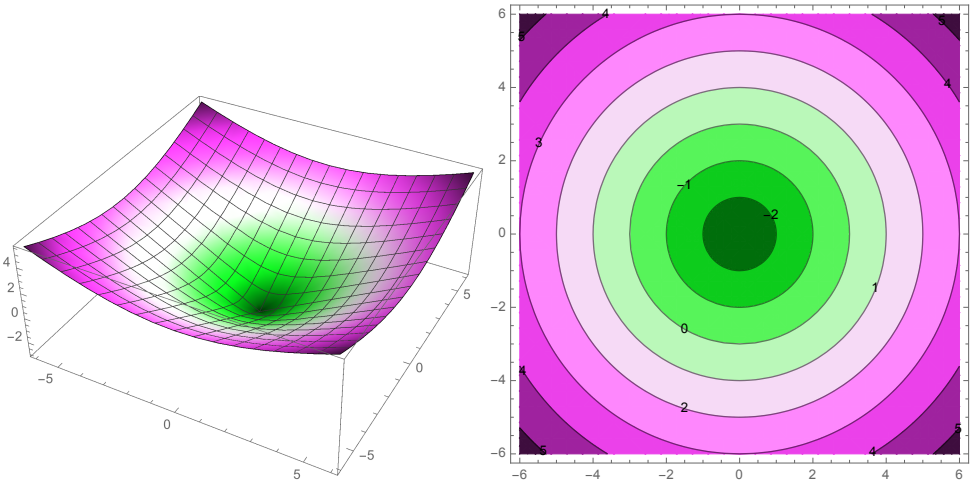
Some (not all) classifiers work by computing a

decision function: A function $f(x)$ that maps a point x to a scalar such that
$f(x) > 0$ if $x \in$ class C;
$f(x) \leq 0$ if $x \notin$ class C.
Aka predictor function.

For these classifiers, the decision boundary is $\{x \in \mathbb{R}^d : f(x) = 0\}$
[That is, the set of all points where the decision function is zero.]
Usually, this set is a $(d-1)$-dimensional surface in $\mathbb{R}^d$.

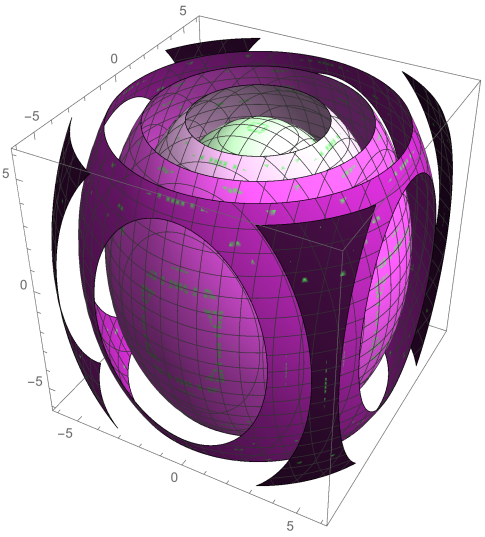
$\{x : f(x) = 0\}$ is also called an isosurface of f for the isovalue 0.

f has other isosurfaces for other isovalues, e.g., $\{x : f(x) = 1\}$.



radiusplot.pdf, radiusiso.pdf [3D plot and isocontour plot of the cone] $f(x, y) = \sqrt{x^2 + y^2} - 3$.

[Imagine a decision function in $\mathbb{R}^d$, and imagine its $(d-1)$-dimensional isosurfaces.]



radiusiso3d.pdf

[One of these spheres could be the decision boundary.]

linear classifier: The decision boundary is a line/plane.
Usually uses a linear decision function. [Sometimes no decision fn.]

Math Review

[I will write vectors in matrix notation.]

Vectors: $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = [x_1 \;\; x_2 \;\; x_3 \;\; x_4 \;\; x_5]^\top$

Think of x as a point in 5-dimensional space.

Conventions (often, but not always):

uppercase roman = matrix, random variable, set	X
lowercase roman = vector	x
Greek = scalar	α
Other scalars:	n = # of sample points
	d = # of features (per point)
	= dimension of sample points
	$i \; j \; k$ = indices
function (often scalar)	$f(\;), s(\;), \ldots$

inner product (aka dot product): $x \cdot y = x_1 y_1 + x_2 y_2 + ... + x_d y_d$

also written $x^\top y$

Clearly, $f(x) = w \cdot x + \alpha$ is a linear function in x.

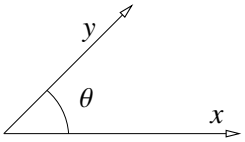
Euclidean norm: $\|x\| = \sqrt{x \cdot x} = \sqrt{x_1^2 + x_2^2 + ... + x_d^2}$

$\|x\|$ is the length (aka Euclidean length) of a vector x.

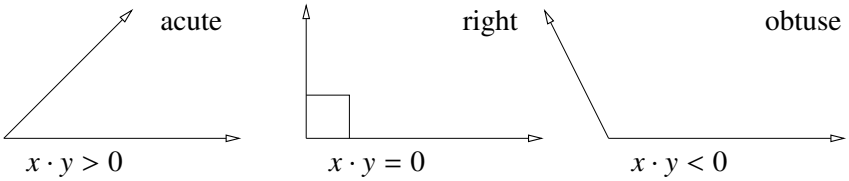
Given a vector $x \neq 0$, $\frac{x}{\|x\|}$ is a unit vector (length 1).

"Normalize a vector x": replace x with $\frac{x}{\|x\|}$.

Use dot products to compute angles:



$$\cos\theta = \frac{x \cdot y}{\|x\| \, \|y\|} = \underbrace{\frac{x}{\|x\|}}_{\text{length 1}} \cdot \underbrace{\frac{y}{\|y\|}}_{\text{length 1}}$$



Given a linear decision function $f(x) = w \cdot x + \alpha$, the decision boundary is

$H = \{x : w \cdot x = -\alpha\}$.

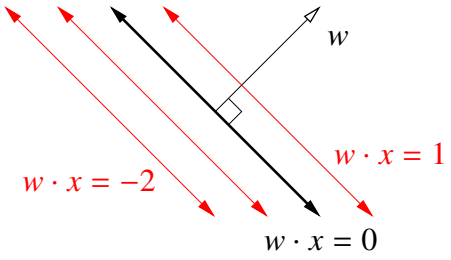
The set H is called a hyperplane. (A line in 2D, a plane in 3D.)

[A hyperplane is what you get when you generalize the idea of a plane to higher dimensions. The three most important things to understand about a hyperplane is (1) it has dimension $d-1$ and it cuts the d-dimensional space into two halves; (2) it's flat; and (3) it's infinite.]

Theorem: Let x, y be 2 points that lie on H. Then $w \cdot (y - x) = 0$.

Proof: $w \cdot (y - x) = -\alpha - (-\alpha) = 0$. [Therefore, w is orthogonal to any vector that lies on H.]

w is called the normal vector of H,
because (as the theorem shows) w is normal (perpendicular) to H.
[I.e., w is perpendicular to every line through any pair of points on H.]



[Draw black part first, then red parts. hyperplane.pdf]

If w is a unit vector, then $w \cdot x + \alpha$ is the signed distance from x to H.
I.e., positive on w's side of H; negative on other side.

Moreover, the distance from H to the origin is α. [How do we know that?]

Hence $\alpha = 0$ if and only if H passes through origin.

[w does not have to be a unit vector for the classifier to work.
If w is not a unit vector, $w \cdot x + \alpha$ is the signed distance times some real.
If you want to fix that, you can rescale the equation by computing $\|w\|$ and dividing both w and α by $\|w\|$.]

The coefficients in w, plus α, are called weights (or parameters or regression coefficients).

[That's why we call the vector w; "w" stands for "weights."]

The input data is linearly separable if there exists a hyperplane that separates all the sample points in class C from all the points NOT in class C.

[At the beginning of this lecture, I showed you one plot that's linearly separable and two that are not.]

[We will investigate some linear classifiers that only work for linearly separable data and some that do a decent job with non-separable data. Obviously, if your data are not linearly separable, a linear classifier cannot do a *perfect* job. But we're still happy if we can find a classifier that usually predicts correctly.]

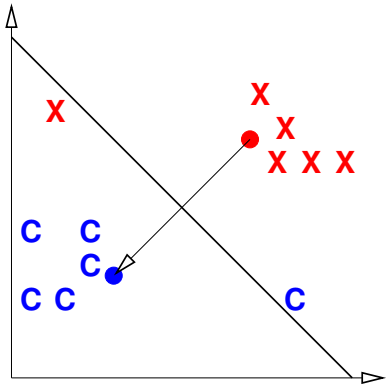
A Simple Classifier

Centroid method: compute mean μ_C of all training points in class C and mean μ_X of all points NOT in C.

We use the decision function

$$f(x) = \underbrace{(\mu_C - \mu_X)}_{\text{normal vector}} \cdot x - \underbrace{(\mu_C - \mu_X) \cdot \frac{\mu_C + \mu_X}{2}}_{\text{midpoint between } \mu_C, \mu_X}$$

so the decision boundary is the hyperplane that bisects line segment w/endpoints μ_C, μ_X.



[Draw data, then μ_C, μ_X, then line & normal. centroid.pdf]

[In this example, there's clearly a linear classifier that classifies every training point correctly, and the centroid method isn't it.
Note that this is hardly the worst example I could have given.
If you're in the mood for an easy puzzle, pull out a sheet of paper and think of an example, with lots of training points, where the centroid method misclassifies every training point but two.]

[Nevertheless, there are circumstances where this method works well, like when all your positive examples come from one Gaussian distribution, and all your negative examples come from another.]

[We can sometimes improve this classifier by adjusting the scalar term to minimize the number of misclassified points. Then the hyperplane has the same normal vector, but a different position.]

Perceptron Algorithm (Frank Rosenblatt, 1957)

Slow, but correct for linearly separable points.

Uses a numerical optimization algorithm, namely, gradient descent.

[Poll:
How many of you know what gradient descent is?
How many of you know what a linear program is?
How many of you know what the simplex algorithm for linear programming is?
How many of you know what a quadratic program is?

We're going to learn what most of these things are. As machine learning people, we will be heavy users of optimization methods. Unfortunately, I won't have time to teach you *algorithms* for many optimization problems, but we'll learn a few. To learn more, take EECS 127.]

Consider n sample points $X_1, X_2, ..., X_n$.

[The reason I'm using capital X here is because we typically store these vectors as rows of a matrix X. So the subscript picks out a row of X, representing a specific sample point.]

For each sample point, the label $y_i = \begin{cases} 1 & \text{if } X_i \in \text{class C, and} \\ -1 & \text{if } X_i \notin \text{C.} \end{cases}$

For simplicity, consider only decision boundaries that pass through the origin. (We'll fix this later.)

Goal: find weights w such that

$X_i \cdot w \geq 0$ if $y_i = 1$, and
$X_i \cdot w \leq 0$ if $y_i = -1$. [remember, $X_i \cdot w$ is the signed distance]

Equivalently: $y_i X_i \cdot w \geq 0$. ← inequality called a constraint.

Idea: We define a risk function R that is positive if some constraints are violated. Then we use optimization to choose w that minimizes R. [That's how we train a perceptron classifier.]

Define the loss function

$$L(z, y_i) = \begin{cases} 0 & \text{if } y_i z \geq 0, \text{ and} \\ -y_i z & \text{otherwise.} \end{cases}$$

[Here, z is the classifier's prediction, and y_i is the correct answer.]

If z has the same sign as y_i, the loss function is zero (happiness).
But if z has the wrong sign, the loss function is positive.

[For each training point, you want to get the loss function down to zero, or as close to zero as possible. It's called the "loss function" because the bigger it is, the bigger a loser your classifier is.]

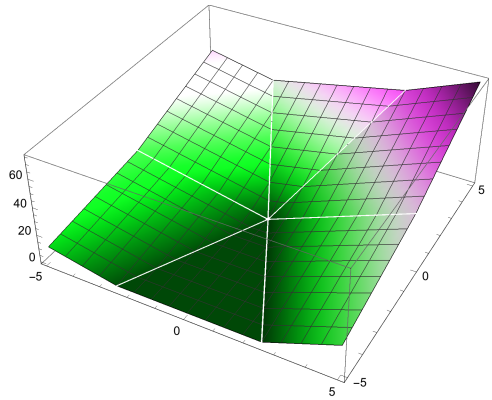
Define risk function (aka objective function or cost function)

$$\begin{aligned} R(w) &= \frac{1}{n} \sum_{i=1}^{n} L(X_i \cdot w, y_i) \\ &= \frac{1}{n} \sum_{i \in V} -y_i X_i \cdot w \qquad \text{where } V \text{ is the set of indices } i \text{ for which } y_i X_i \cdot w < 0. \end{aligned}$$

If w classifies all $X_1, \ldots, X_n$ correctly, then $R(w) = 0$.
Otherwise, $R(w)$ is positive, and we want to find a better w.

Goal: Solve this optimization problem:

Find w that minimizes $R(w)$.



riskplot.pdf [Plot of risk $R(w)$. Every point in the dark green flat spot is a minimum. We'll look at this more next lecture.]