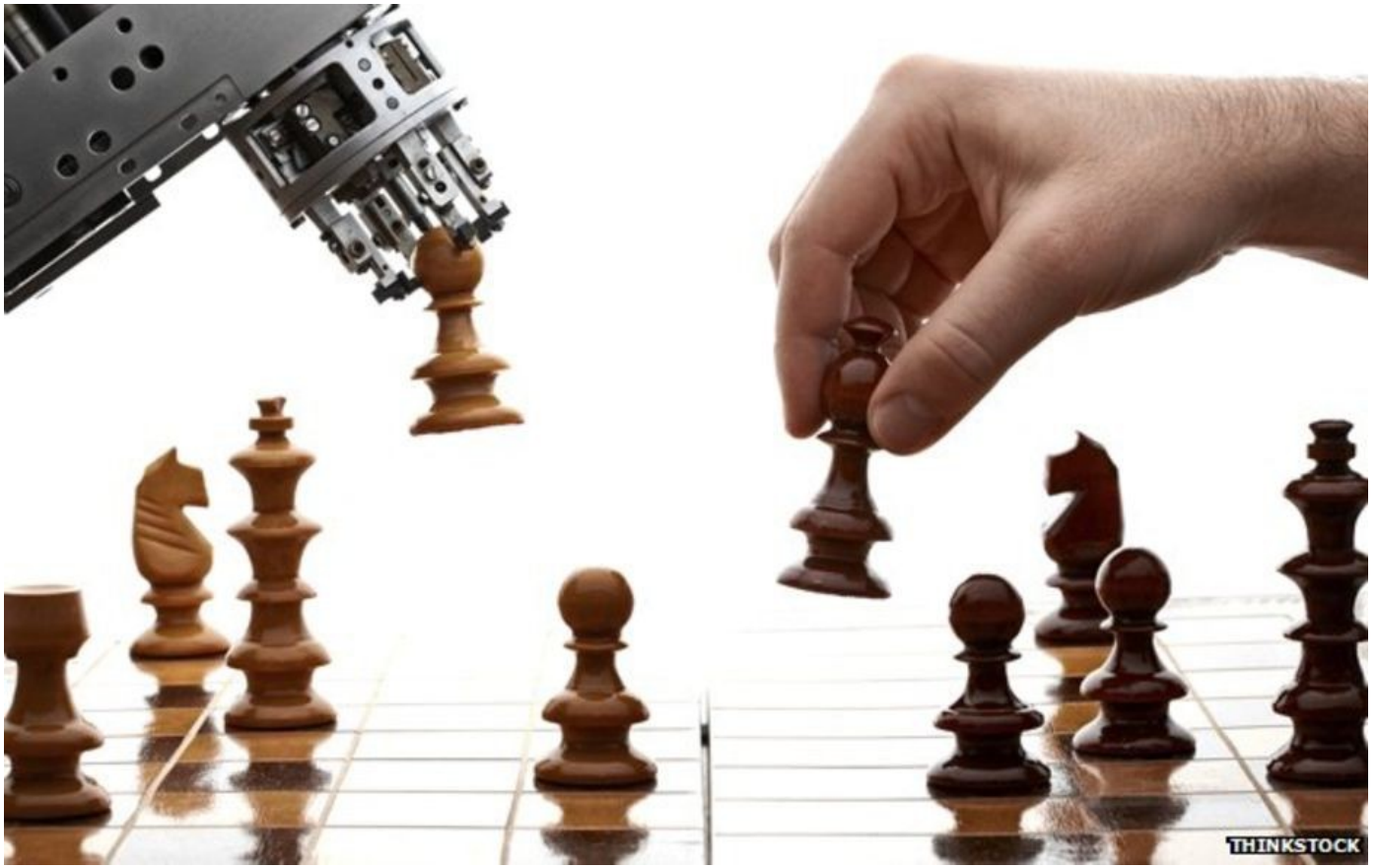


Introduction to Artificial Intelligence

COSC 4550 / COSC 5550

Professor Cheney

10/25/17

nonparametric classification

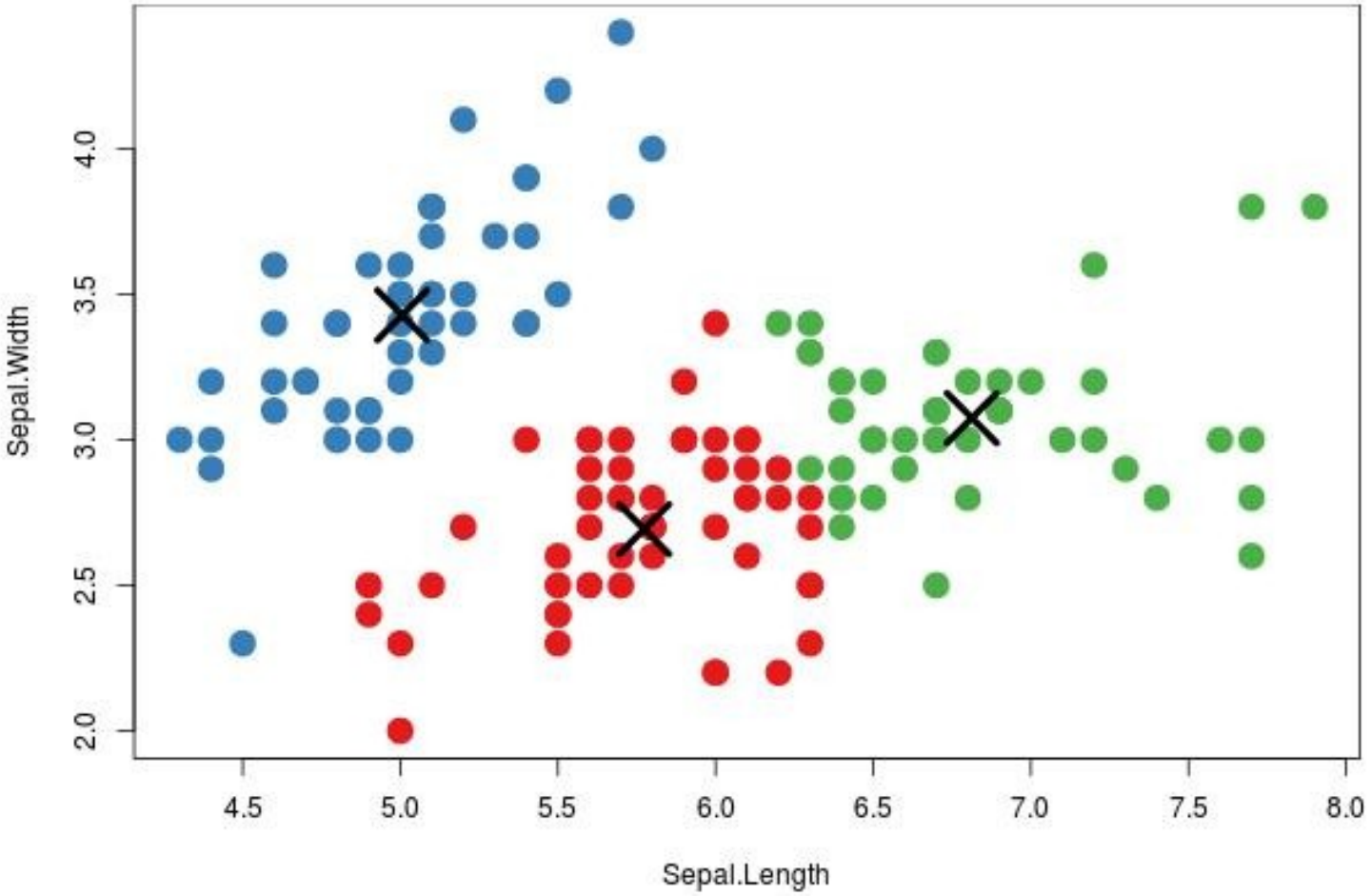
in classification and regression, we've aimed to find a single model that explains all the trends underlying our data

when we have to extrapolate and generalize from our data, the continuity enforced by having this model is critical

but there may also cases where it would be sufficient simply to interpolate in a data-rich problem

in these cases, it can be best to let the data speak for itself, rather than trying to fit a parameterized model to it

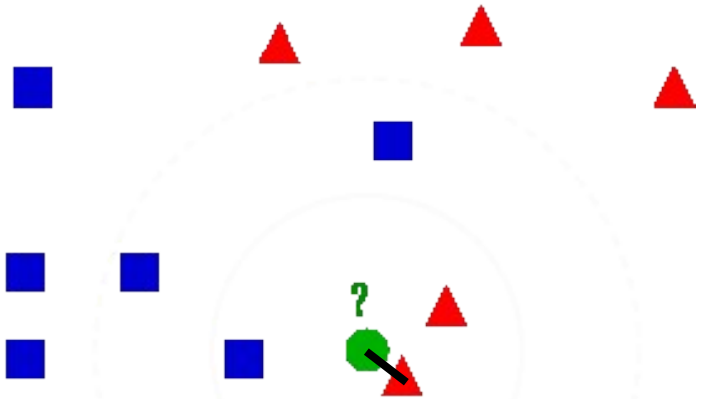
what class is each *x*?
how do you know?



much easier and cheaper to label just those data points than to create an entire model for the relationship between the features! (e.g. sepal width and length)

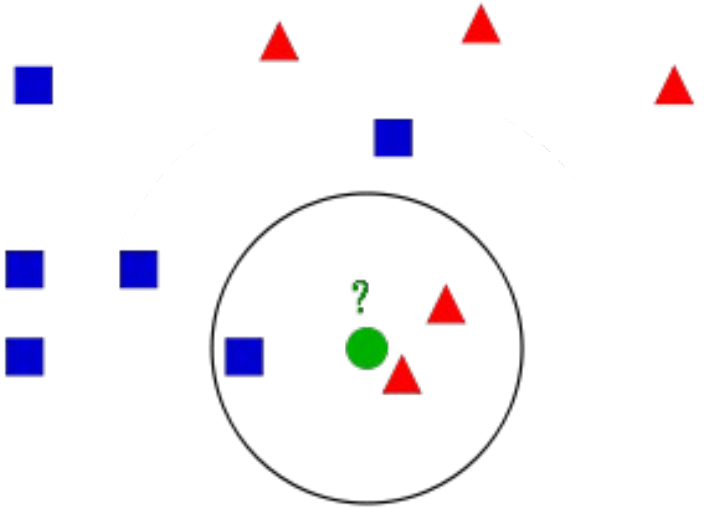
but assumes that you have good local data to provide contextual information around the new point(s) you want to classify

which class is the green dot?



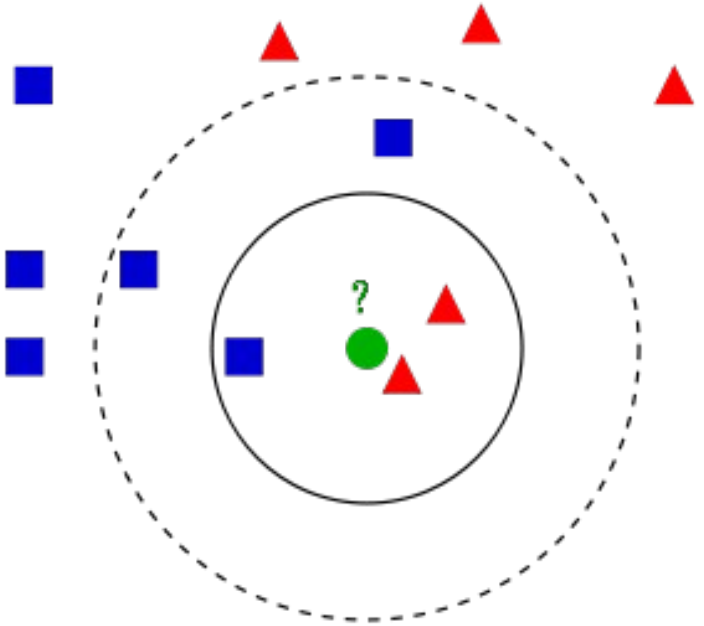
take the class of the labeled data point that is the closest match to the features of your new data point (“nearest neighbor classification”)

“k-nearest neighbors” classification



what if considered the 3 closest points?

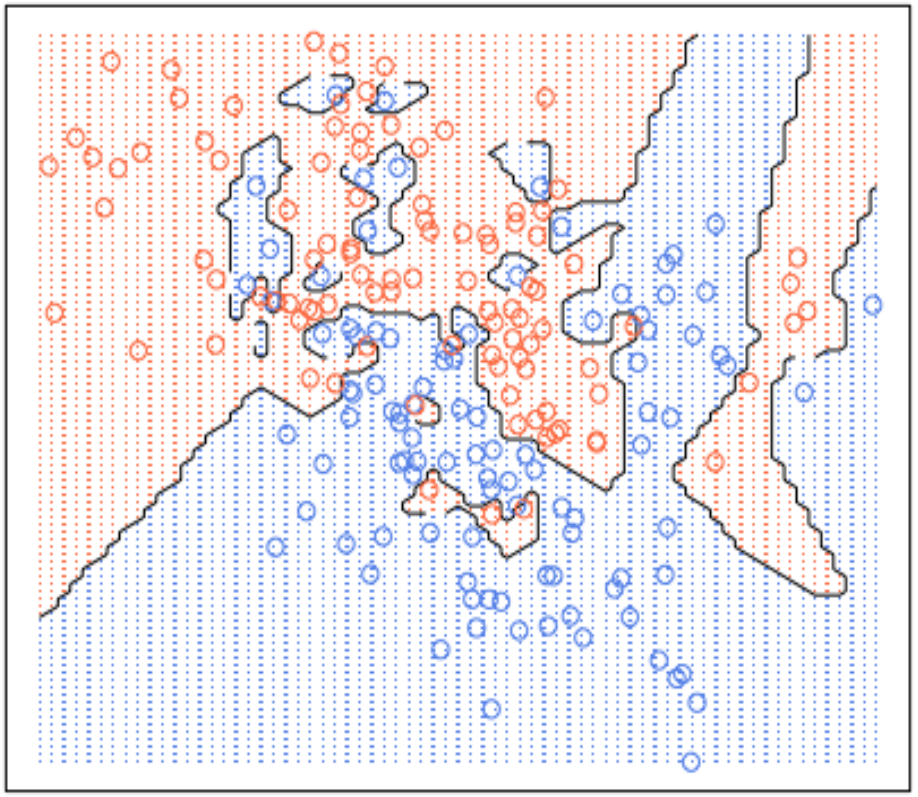
“k-nearest neighbors” classification

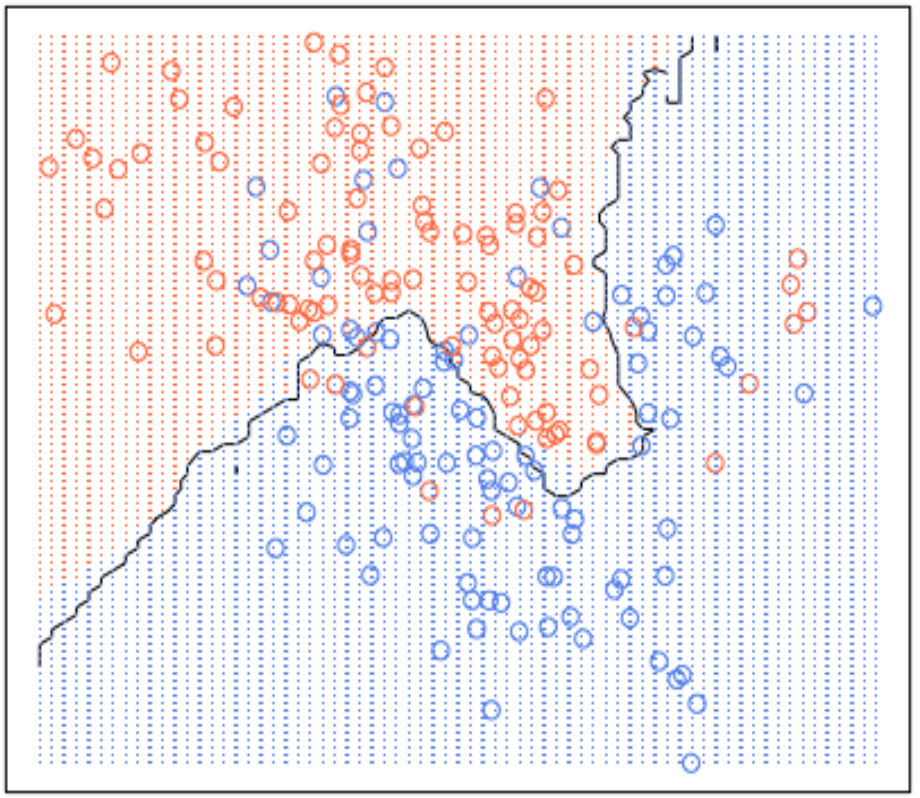


what if considered the 3 closest points?

how about the 5 closest points?

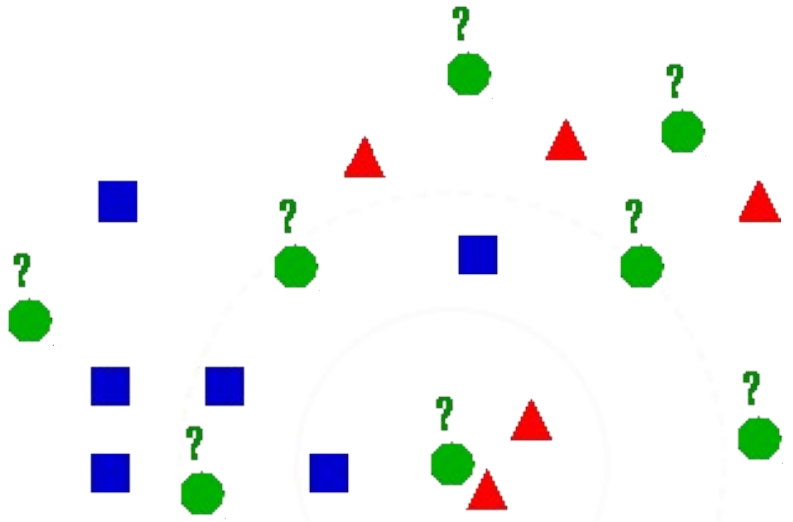
using more neighbors generalizes more





which one is better?

it's possible to overgeneralize



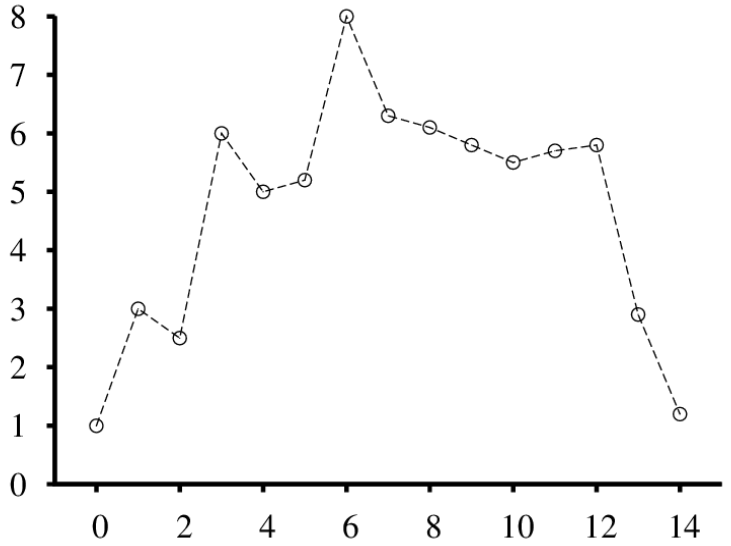
what if k = 11?

it doesn't matter where our new data point is!
(it is always classified as blue)

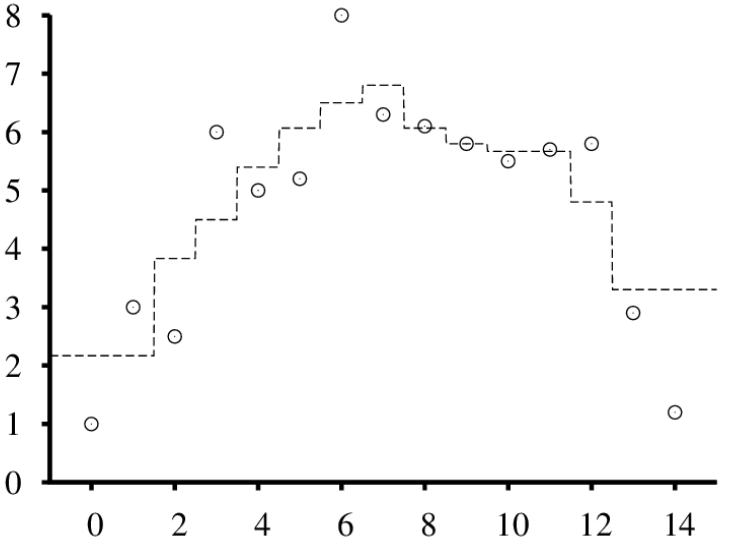
nonparametric regression

same idea as k-nearest neighbors (KNN) classification,
but rather voting for the class of a new data point,
use the nearby points only to fit a model
(e.g. linear regression)

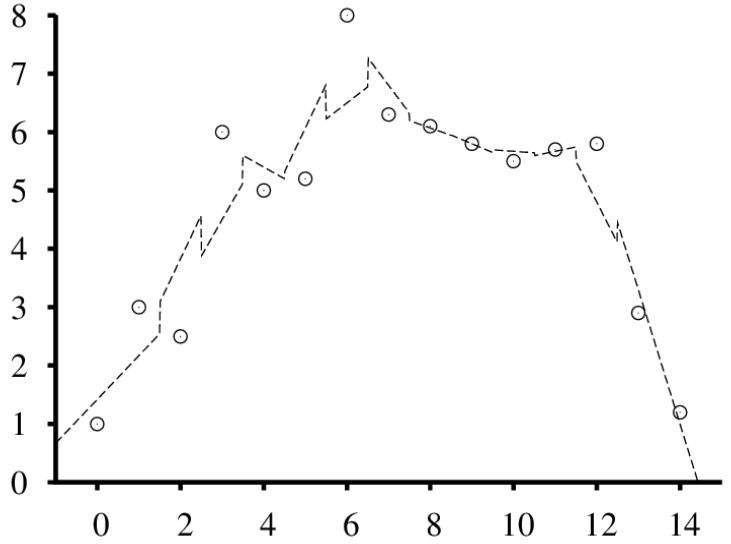
k=2 local regression



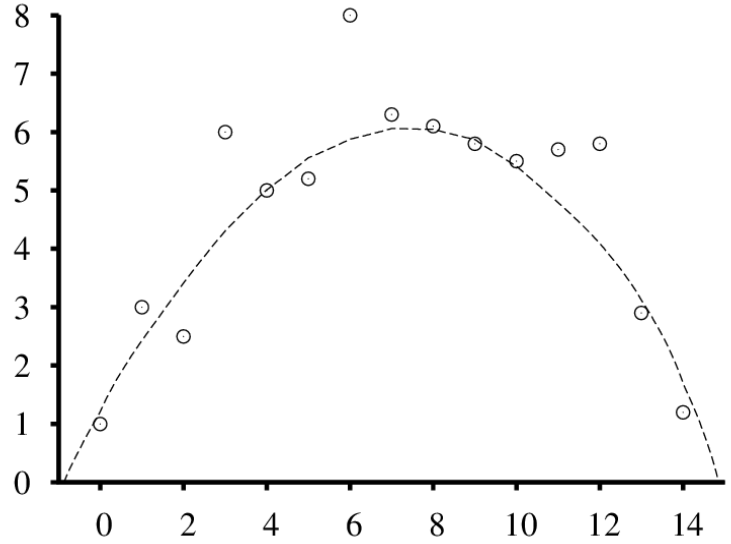
k=3 local mean



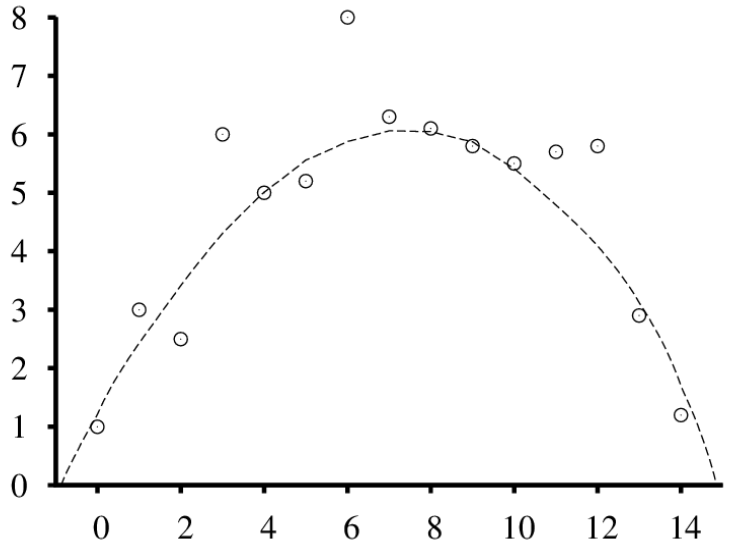
k=3 local regression

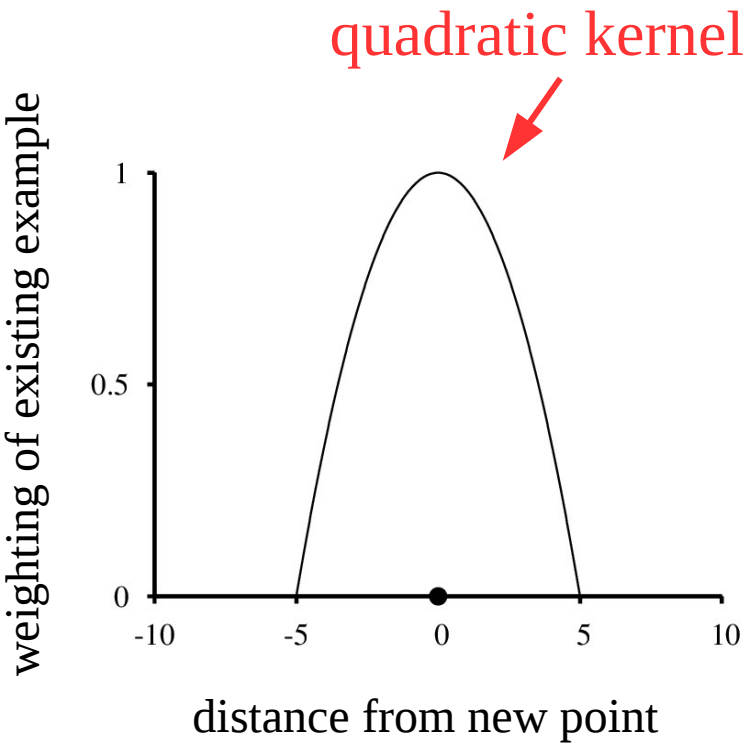


k=10 quadratic kernel



locally-weighted regression





for each query point (x_q),

$$w^* = \text{argmin}_w \Sigma_j (\text{Kernel}(\text{distance}(x_q, x_j)) * (y - w \circ x_j)^2)$$

in general, inverse distance weighting (IDW)
is a nice way to use closely related points more strongly
while still considering further data points

This can be combined with a “k” hard threshold on how far away you consider, but can be used without one too

(very useful in classification too!)

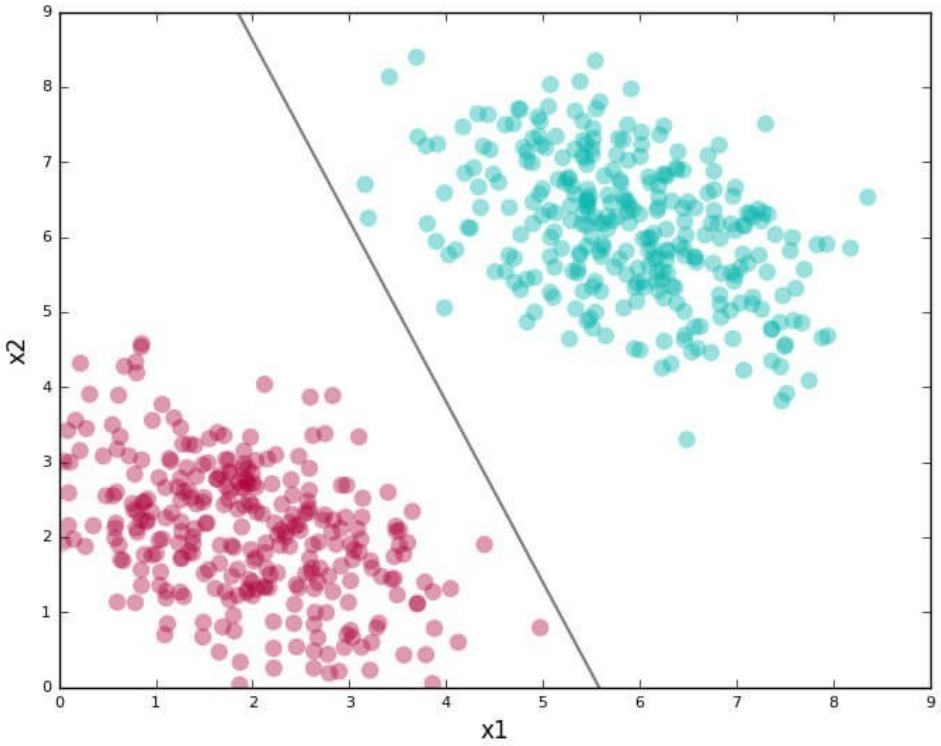
$$w_i(\mathbf{x}) = \frac{1}{d(\mathbf{x}, \mathbf{x}_i)^p}$$

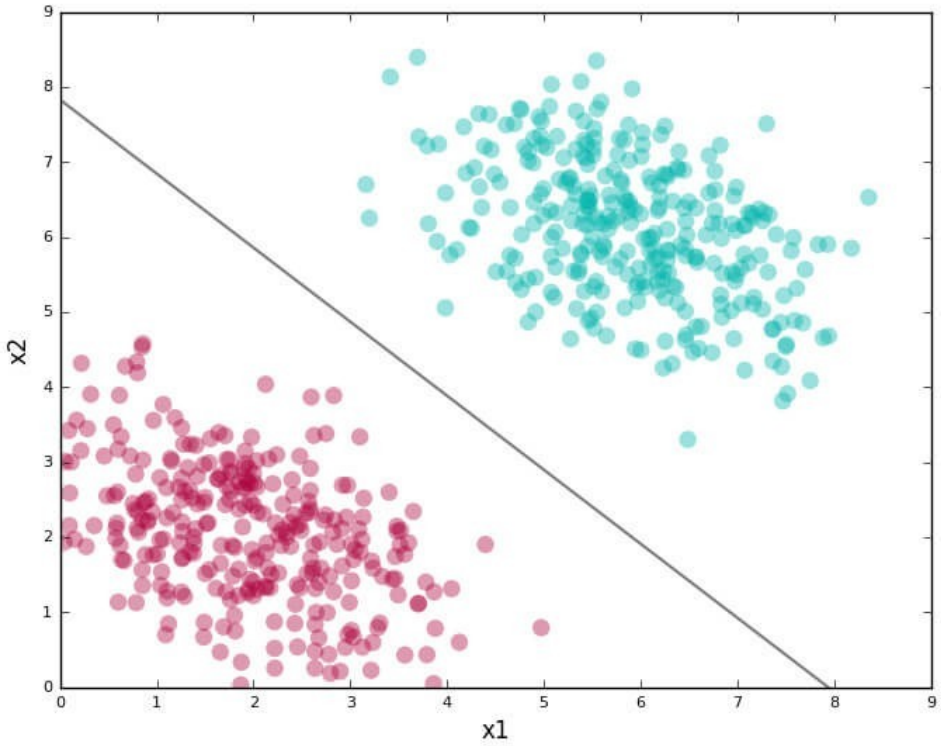
support vector machines

support vector machines are perhaps the most widely used machine learning technique

they are simple, yet effective methods for classification

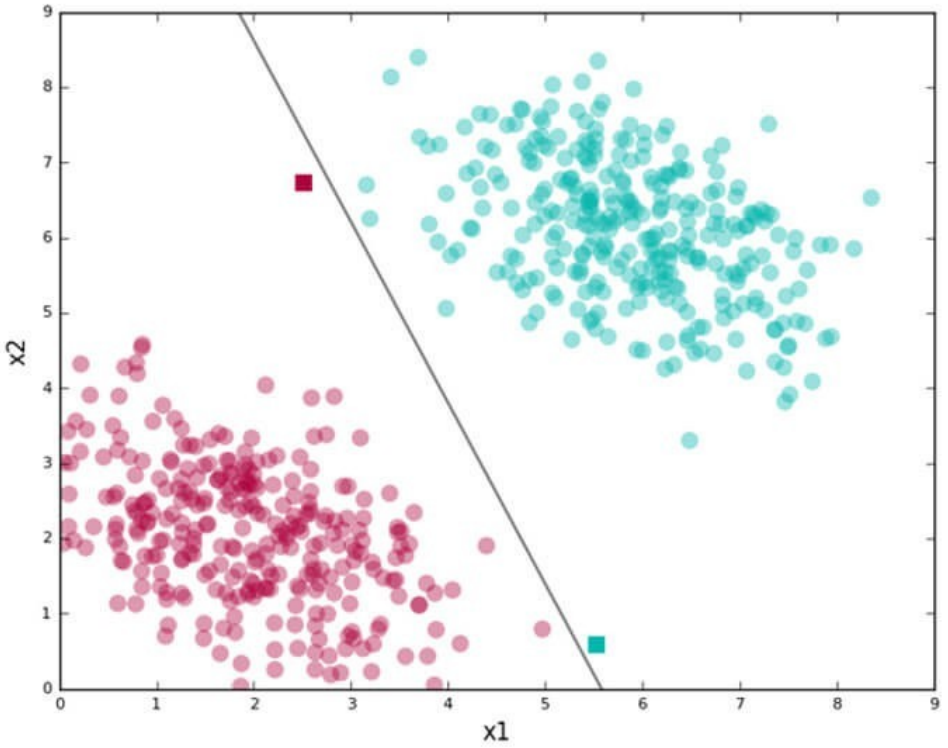
consider two linear separating boundaries

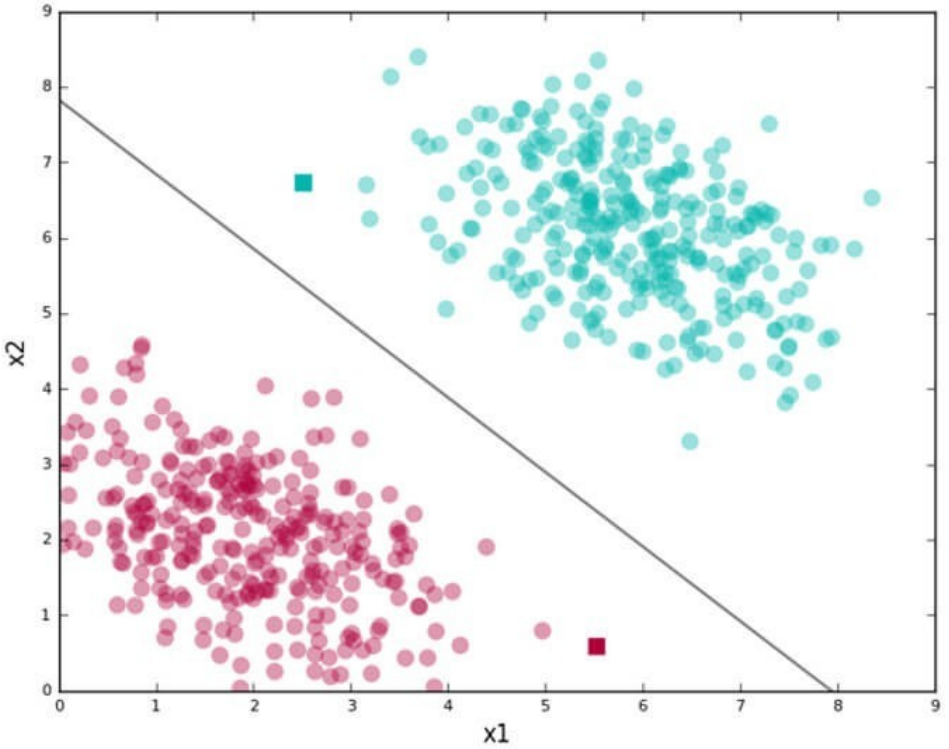




both have perfect classification,
is one any better than the other?

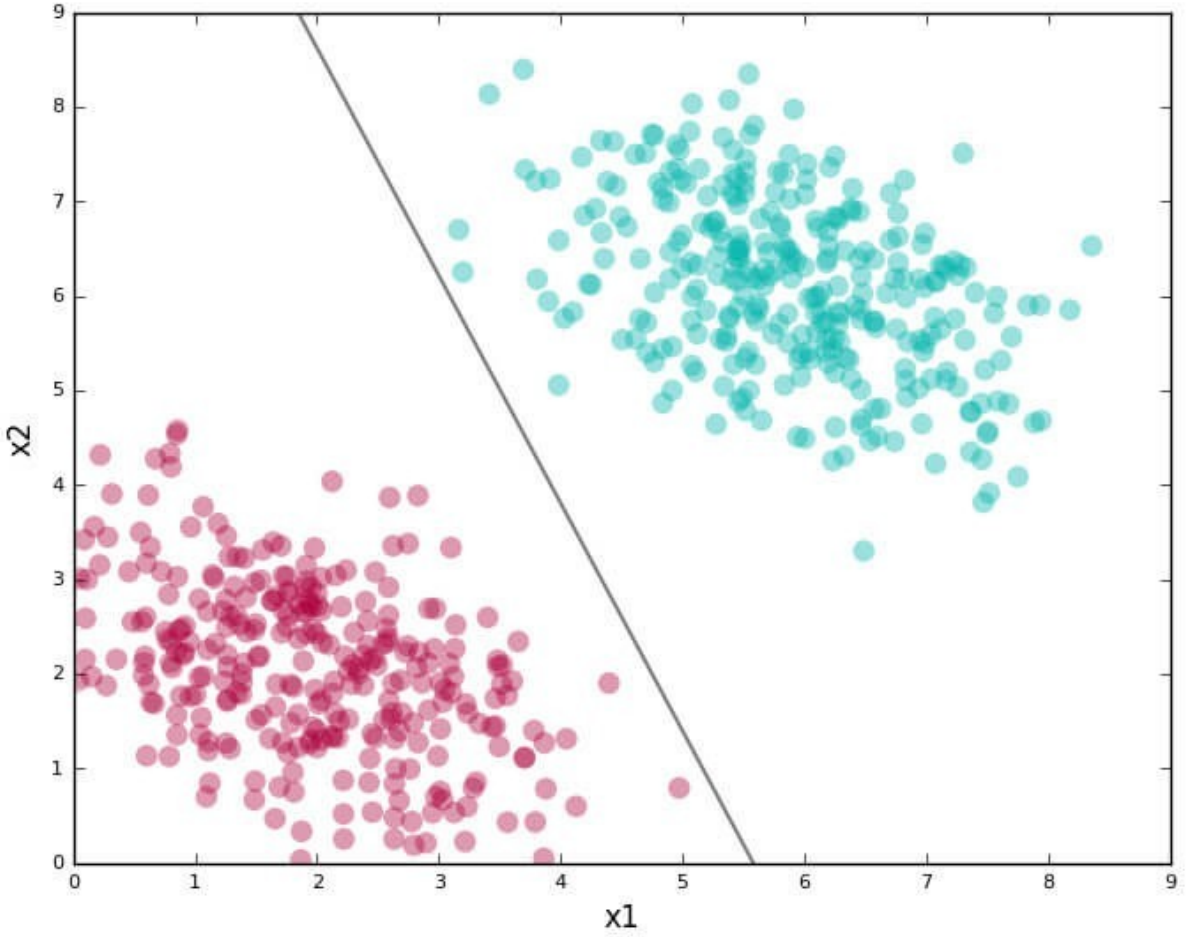
what if we consider points where they would disagree





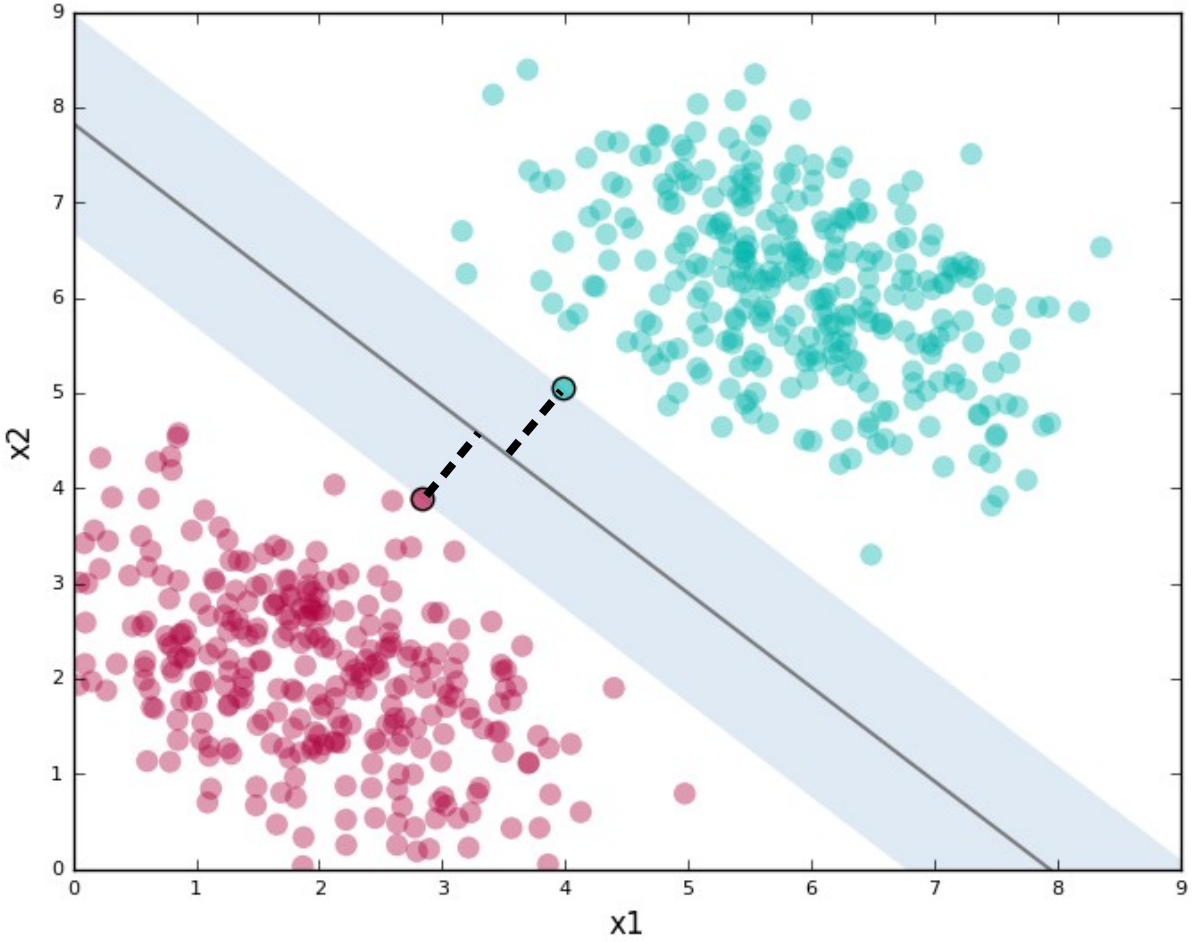
which new classification makes more sense?

despite perfect classification of the current data points, this was a poor choice of a boundary because there exist new potential points that are very similar (i.e. nearby) to points in one class that would be classified as the other

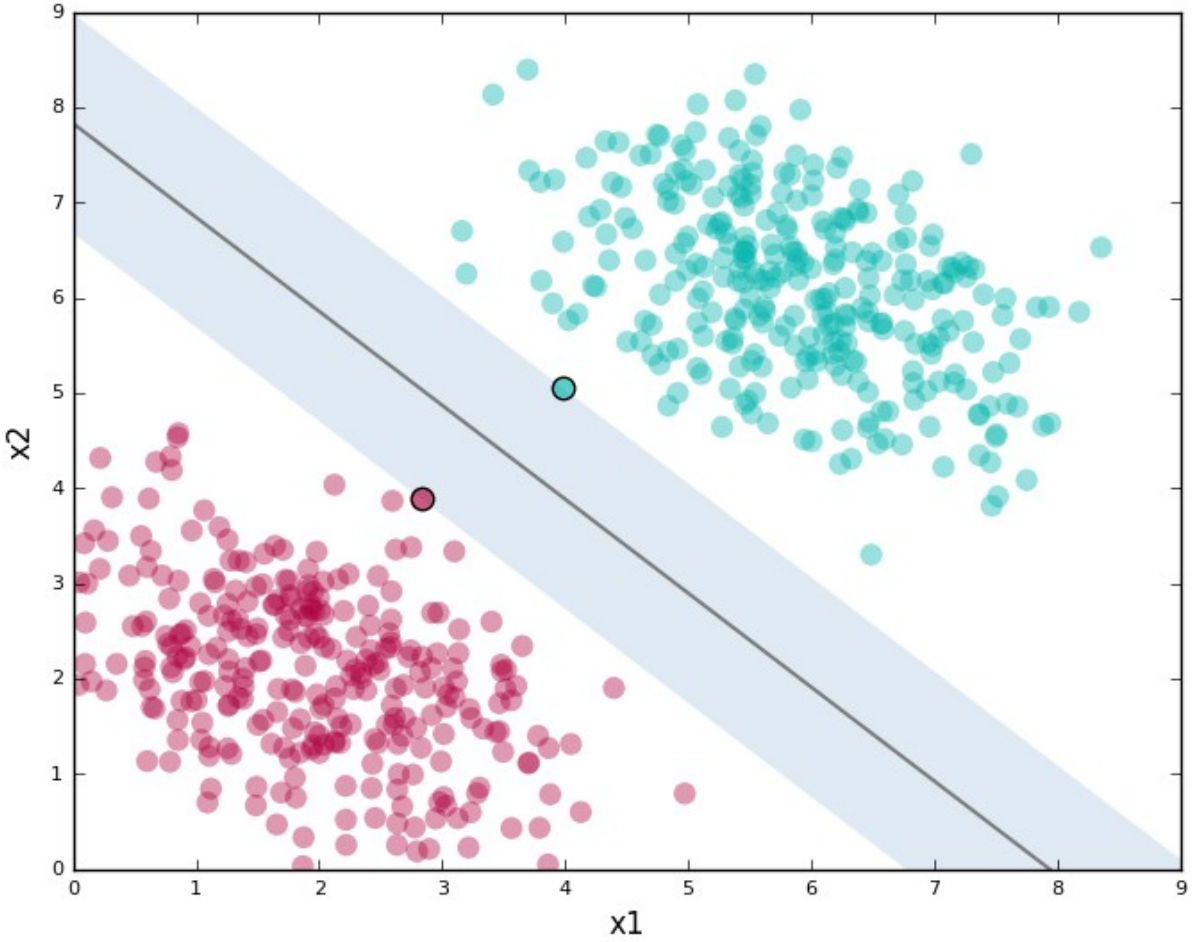


this means our solution would not generalize well!

instead we want a linear separator that will maximize the "margin" between the boundary and the closest example

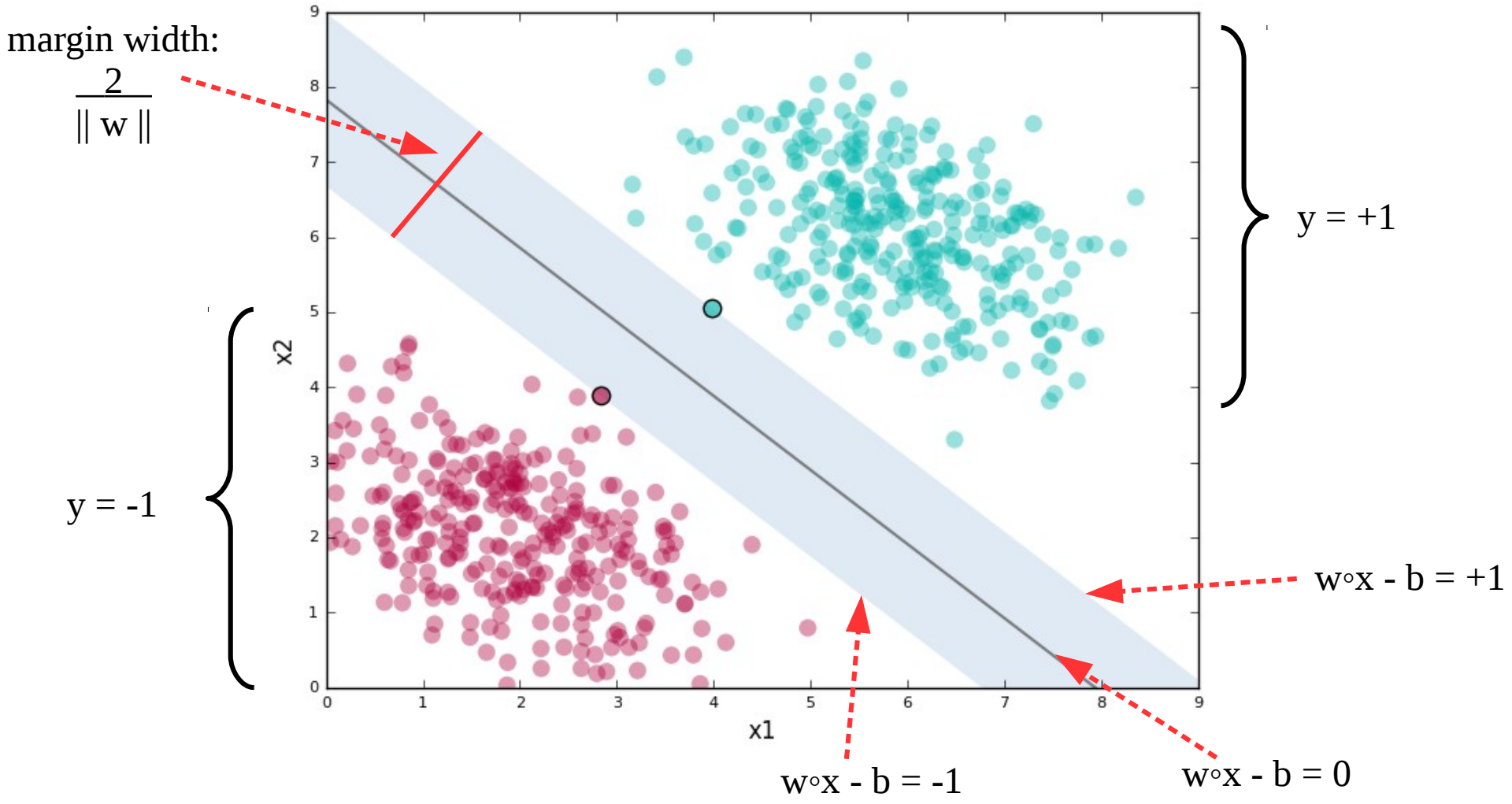


while the linear classification considers all points in making the decision boundary, to maximize the margins we only need to consider the two bold points below (called “support vectors”)



support vector machines (SVMs) incorporate the advantages of both parametric and non-parametric methods

they do this by fitting a model to the data, but only using a small number of points local to that decision boundary



minimize ||w|| subject to the constraint that $y(w \circ x - b) \geq 1$, for all (x, y)
"get as wide a margin as you can while still classifying all the points correctly"

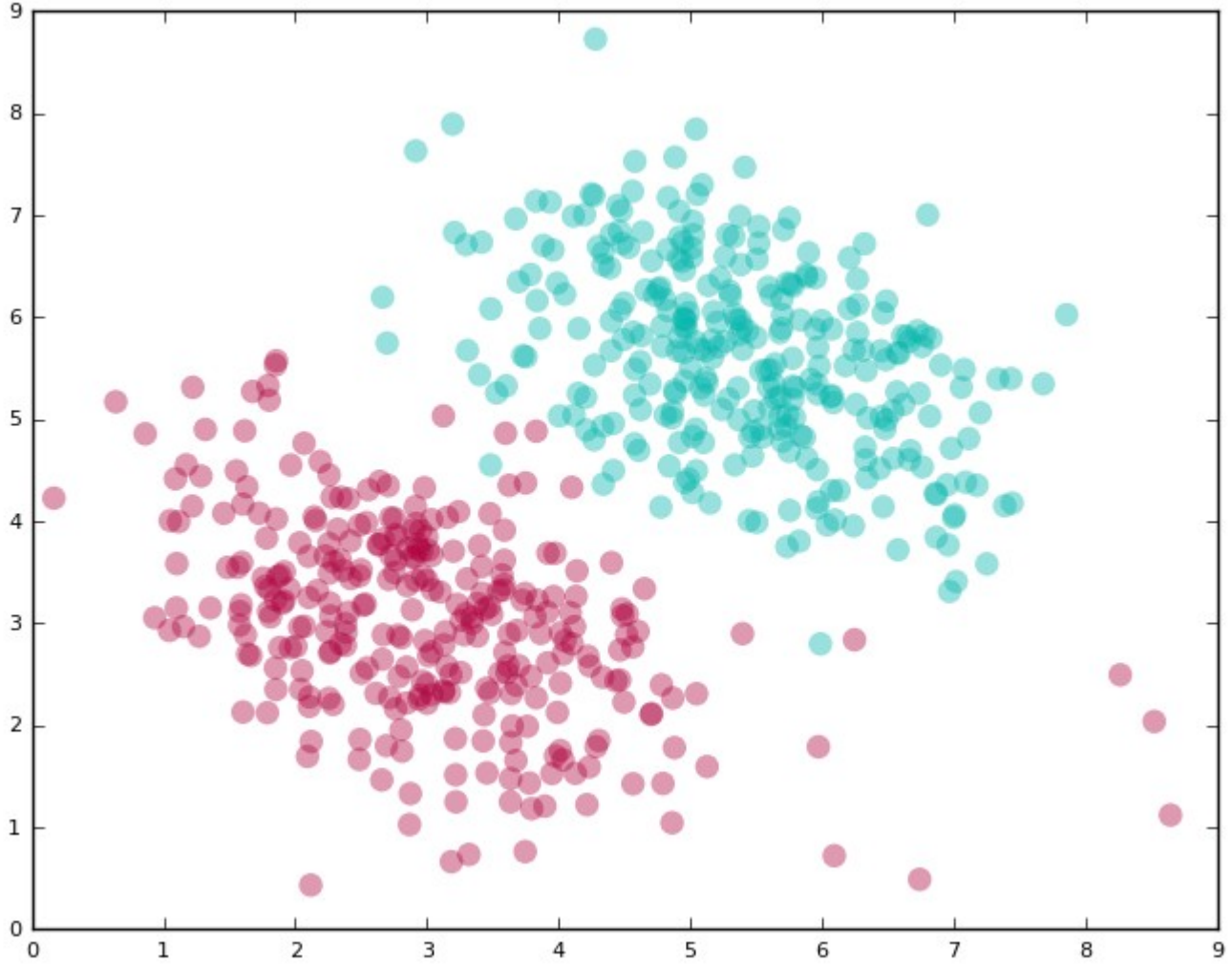
this system can be solved easily and quickly with quadratic programming
(we could use iterative optimization if we wanted, but it would take longer)

we said that SVMs were one of the most widely used method in machine learning techniques

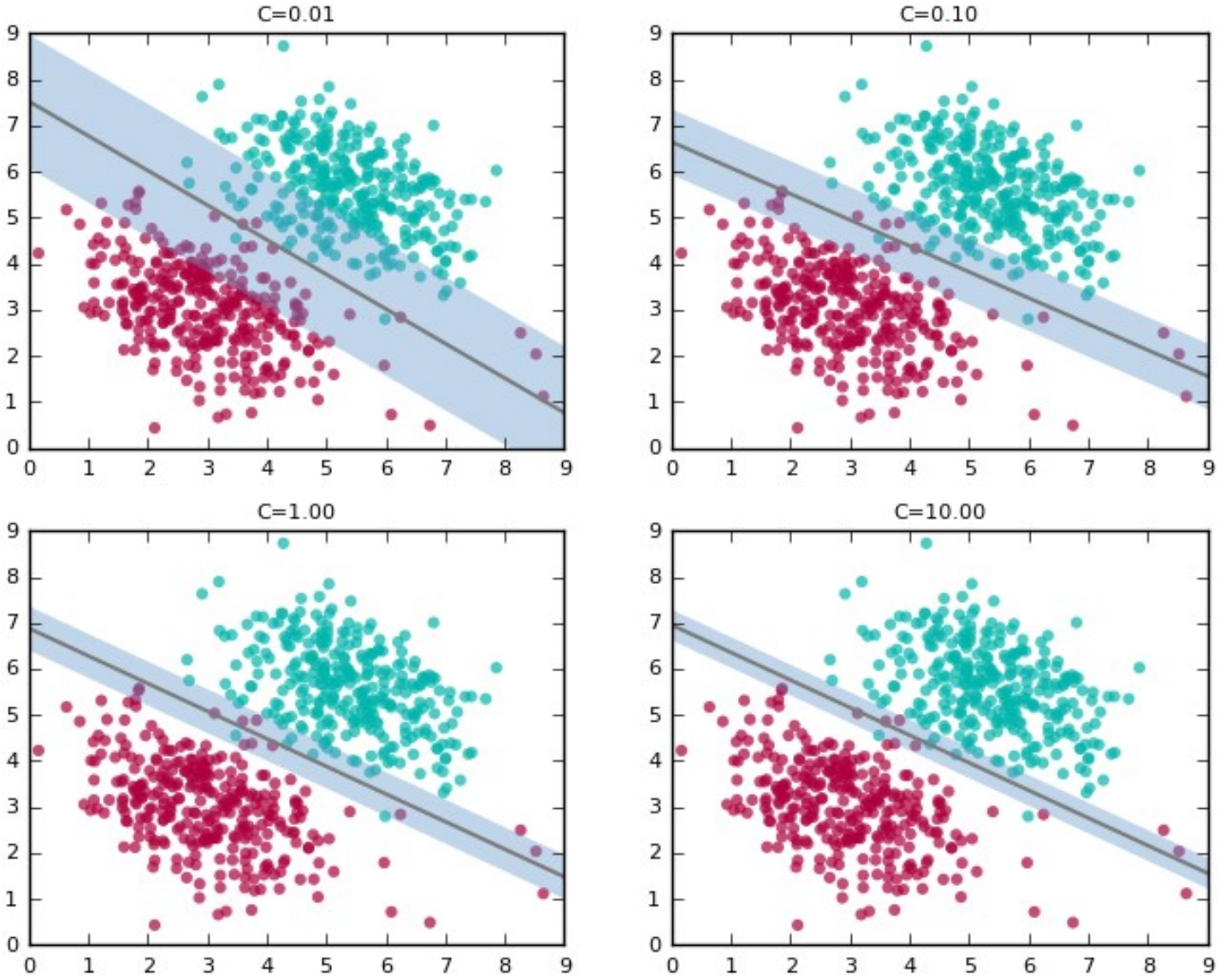
but everything we've seen to far is built on the assumption of simple linearly separable classes…

what if it's not separable?

for data that is noisy and not perfectly separable,
we can allow data points to enter or cross the margin,
incurring a cost (C or λ) for these events (“soft margin”)

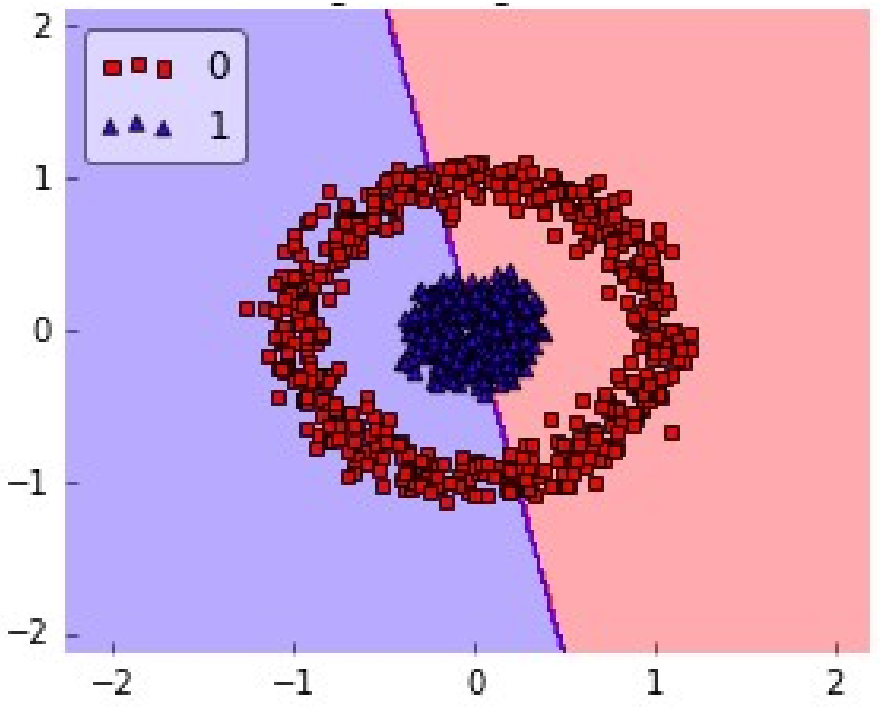


the severity of this cost dictates how much the SVM will try to avoid allowing points into the margin (so a large cost will lead to a small margin)

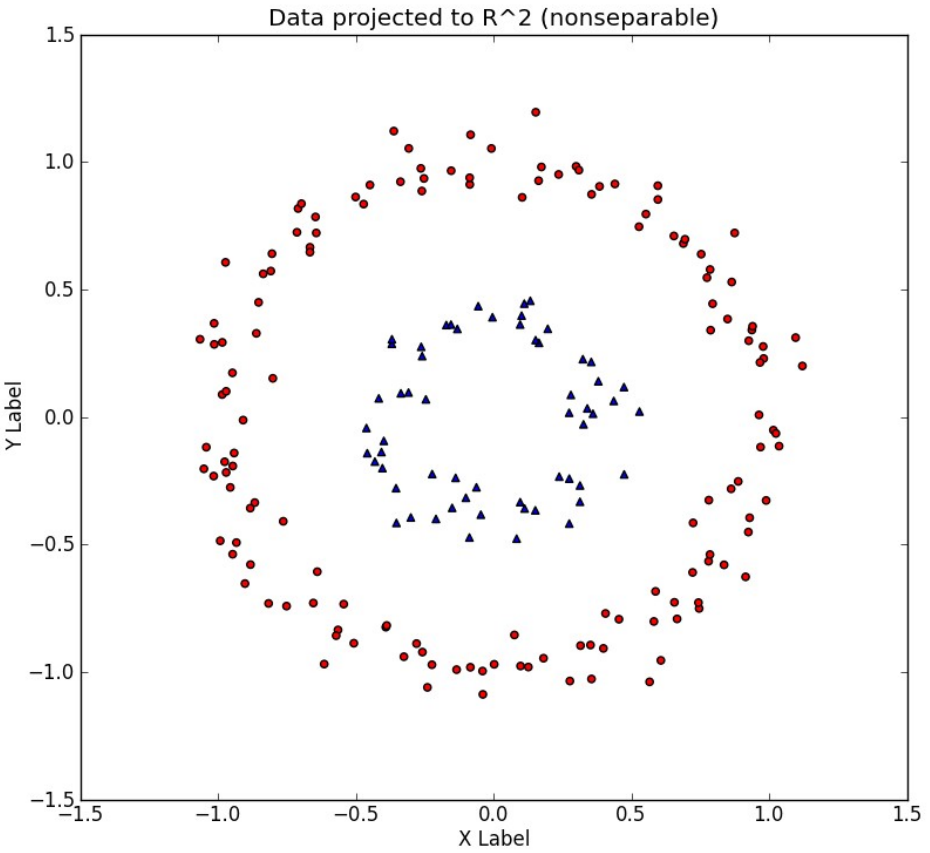


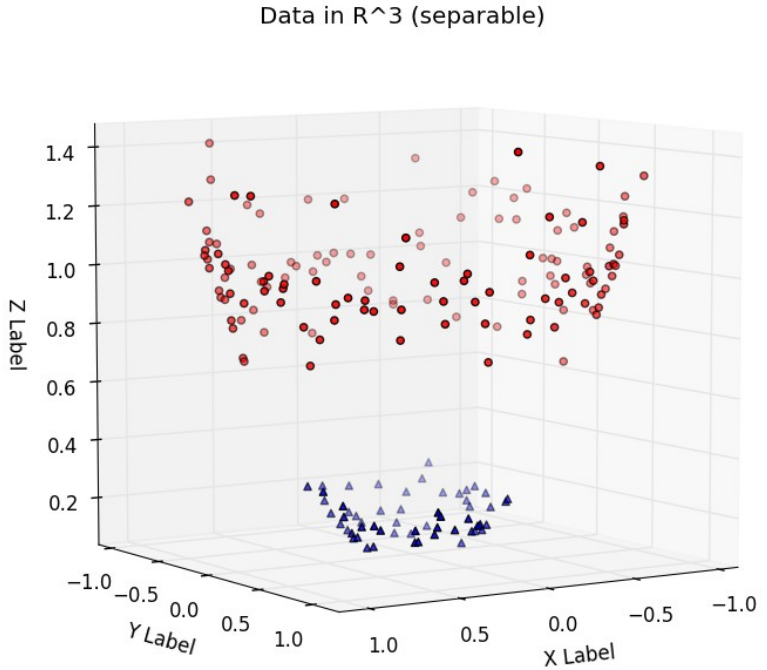
what if it's not a linear relationship?

the solvers for SVMs work very well for linear systems, but how can we solve a classification problem like this with a linear separation boundary?



let's project it into an extra dimension, so now there's a linearly separating plane!

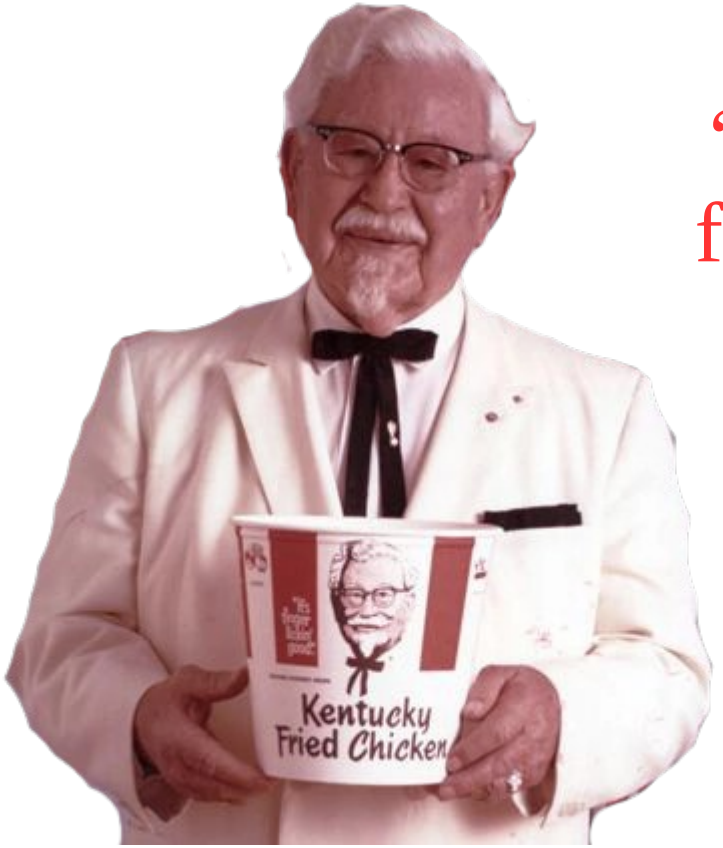




this is called the “kernel trick”

this new function of the existing dimensions

e.g. $\Phi(x_1, x_2)$, $(\mathbf{R}^2 \rightarrow \mathbf{R}^3)$

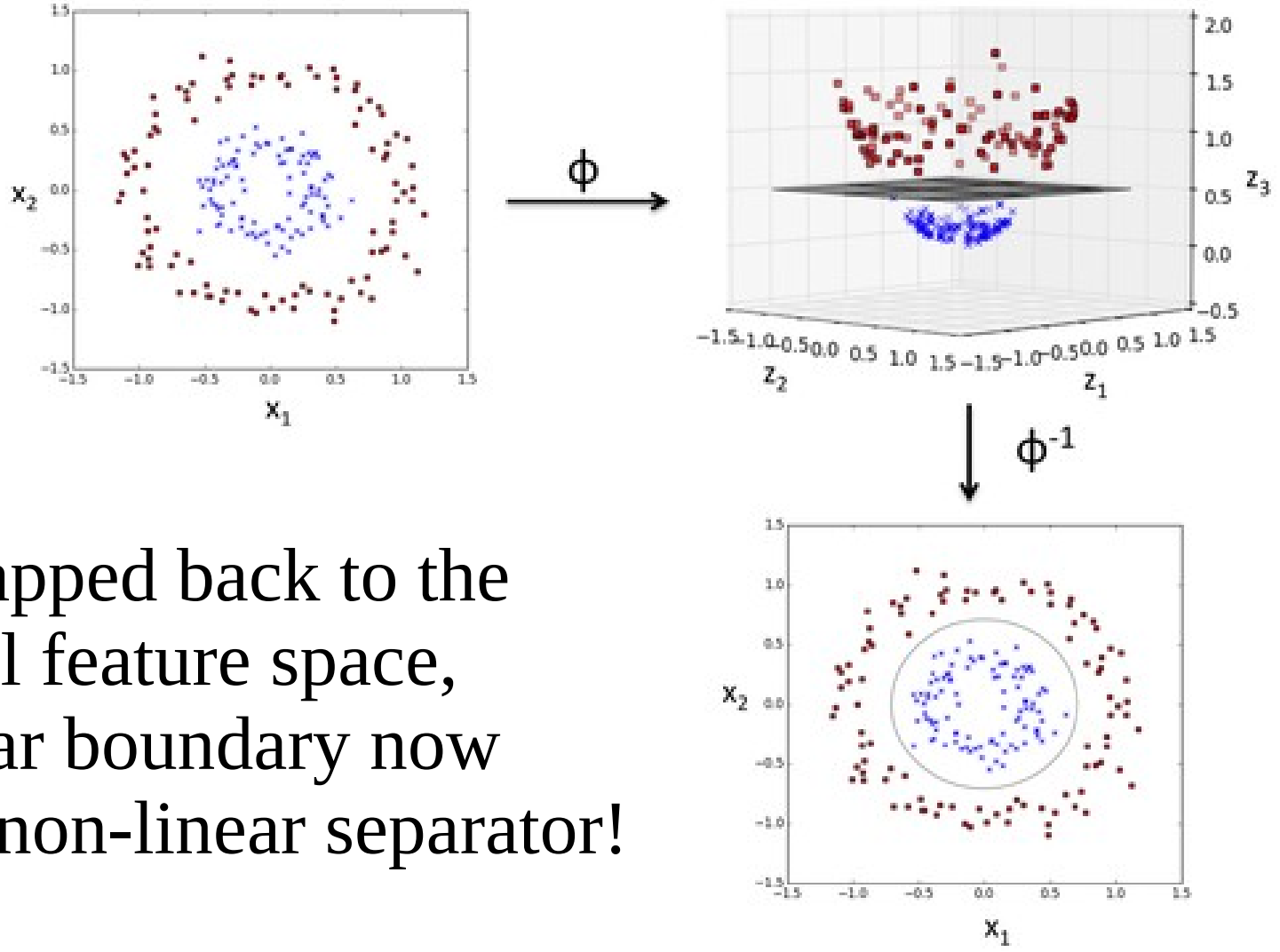


“kernel function”

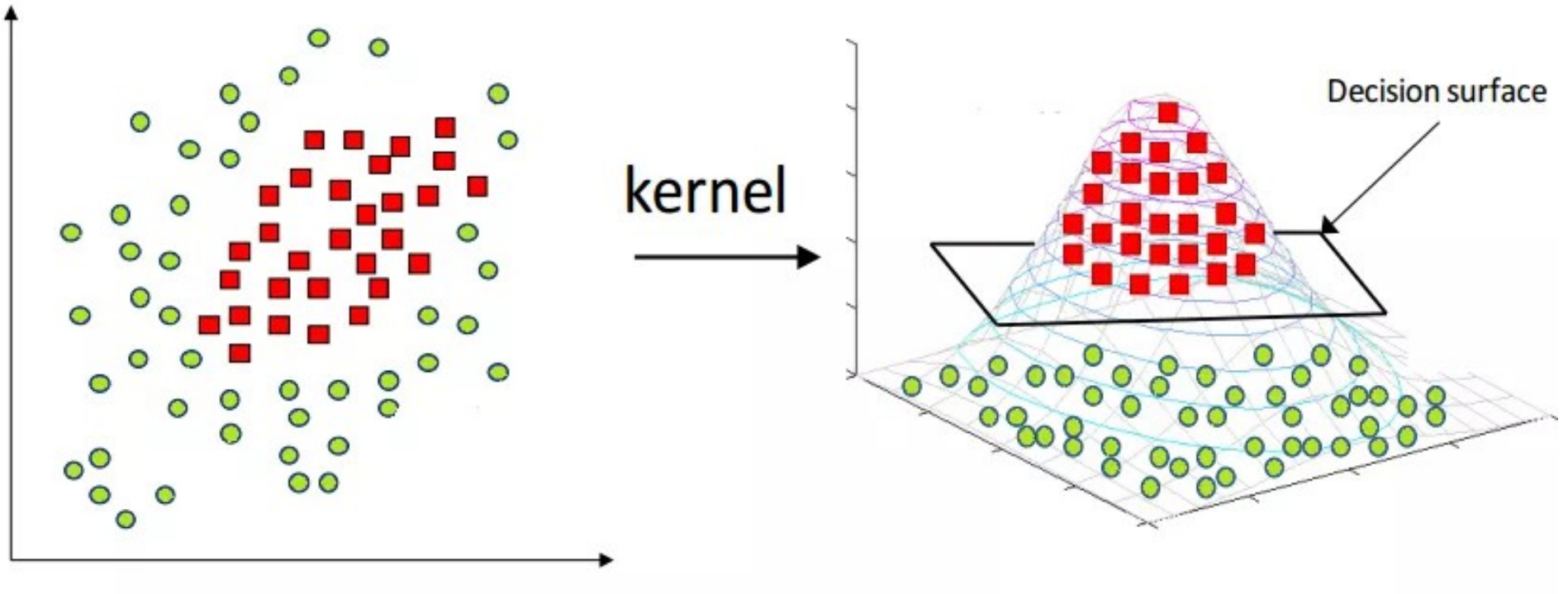
$$f_1 = x_1$$
$$f_2 = x_2$$
$$f_3 = \sqrt{(2)}\, x_1 x_2$$

our quadratic programming solver only ever sees the dot product of two data points

$$f_3(x_j) \circ f_3(x_k) = (x_j \circ x_k)^2$$



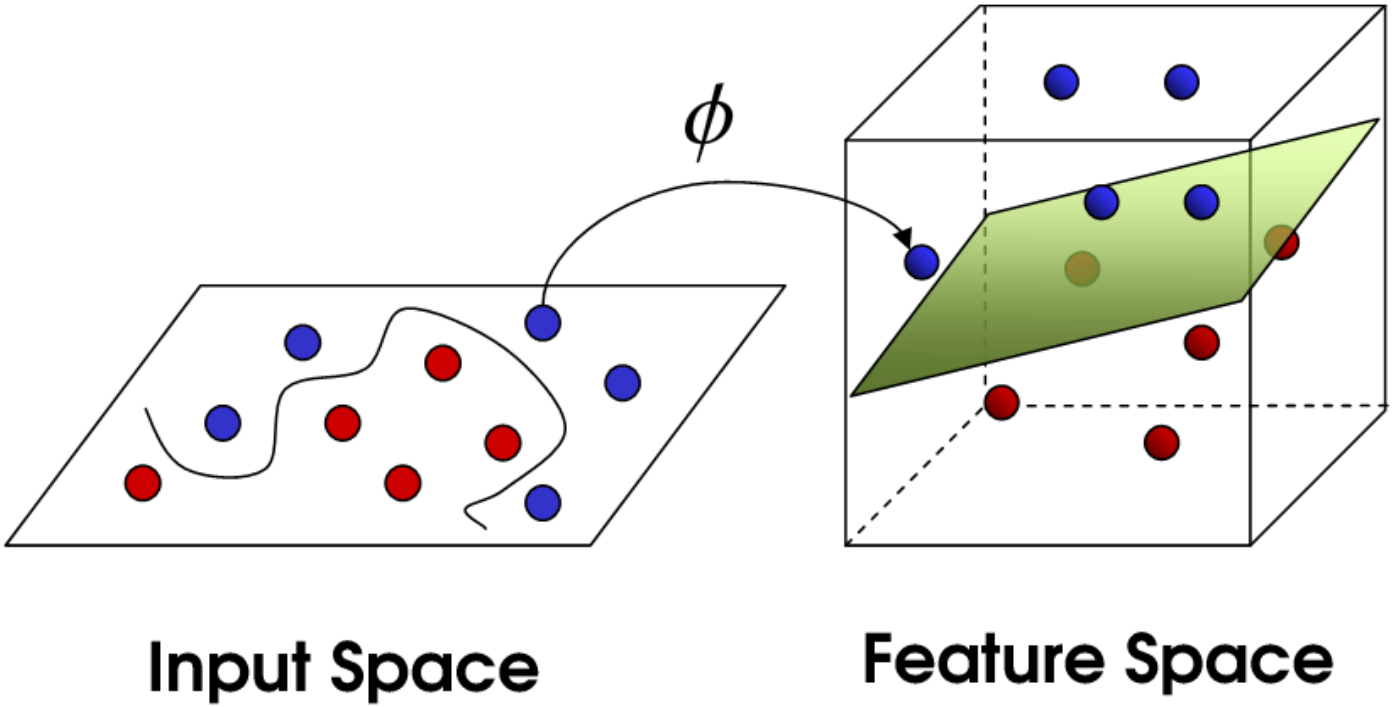
when mapped back to the original feature space, this linear boundary now represents a non-linear separator!


kernel
Decision surface

there are many kernel functions that may be helpful features to add...

Some common kernels include:

- Polynomial (homogeneous): $k(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j})^d$
- Polynomial (inhomogeneous): $k(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j} + 1)^d$
- Gaussian radial basis function: $k(\vec{x_i}, \vec{x_j}) = \exp(-\gamma \|\vec{x_i} - \vec{x_j}\|^2)$, for $\gamma > 0$. Sometimes parametrized using $\gamma = 1/2\sigma^2$
- Hyperbolic tangent: $k(\vec{x_i}, \vec{x_j}) = \tanh(\kappa \vec{x_i} \cdot \vec{x_j} + c)$, for some (not every) $\kappa > 0$ and $c < 0$



(almost) every system of N data points
will be linearly separable in N-1 dimensions

and often in many fewer dimensions than that in practice
(especially problems with strong classes/signals)

using domain knowledge to pick kernels can be helpful

or use cross-validation to test a large number of kernels
(careful of overfitting it testing lots of them!)

this “kernel trick” now lets us solve
a non-linear problem with a linear solver

it can also be used on problems that don't require
a linear system (e.g. KNN, logistic classification)