Lecture 8: More on Classification, Lines from Edges, Interest Points, Binary Operations

CAP 5415: Computer Vision

Fall 2009

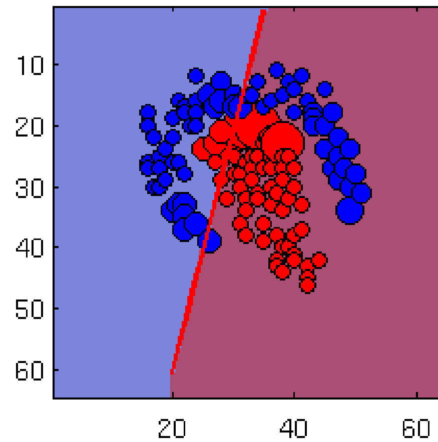
Non-Linear Classification

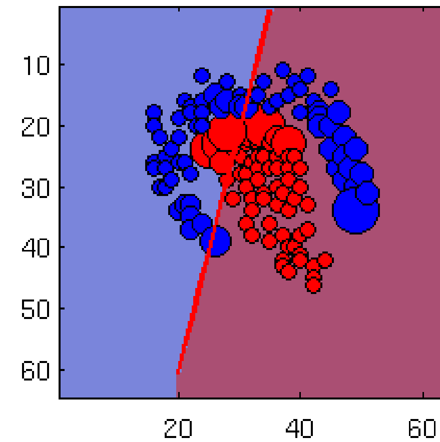
- We've focused on linear decision boundaries
- What if that's not good enough?

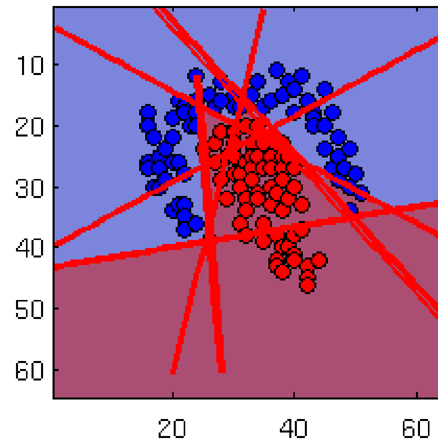
x

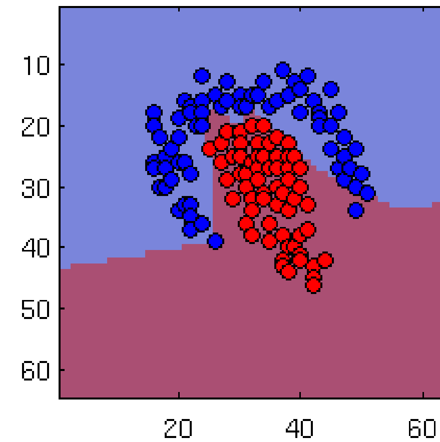
y

Example from last time





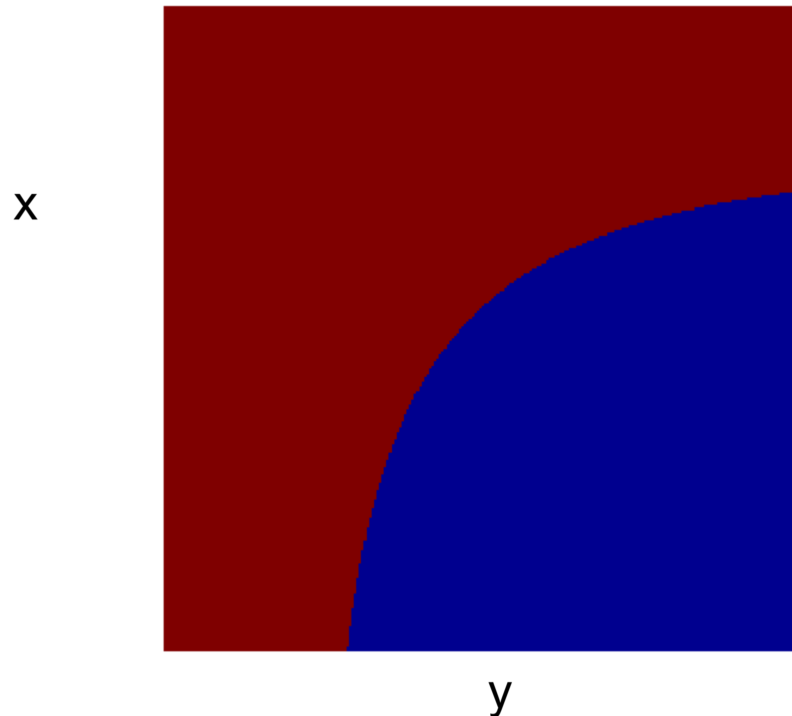




Non-Linear Classification

- What would a quadratic decision boundary look like?
- This is the decision boundary from

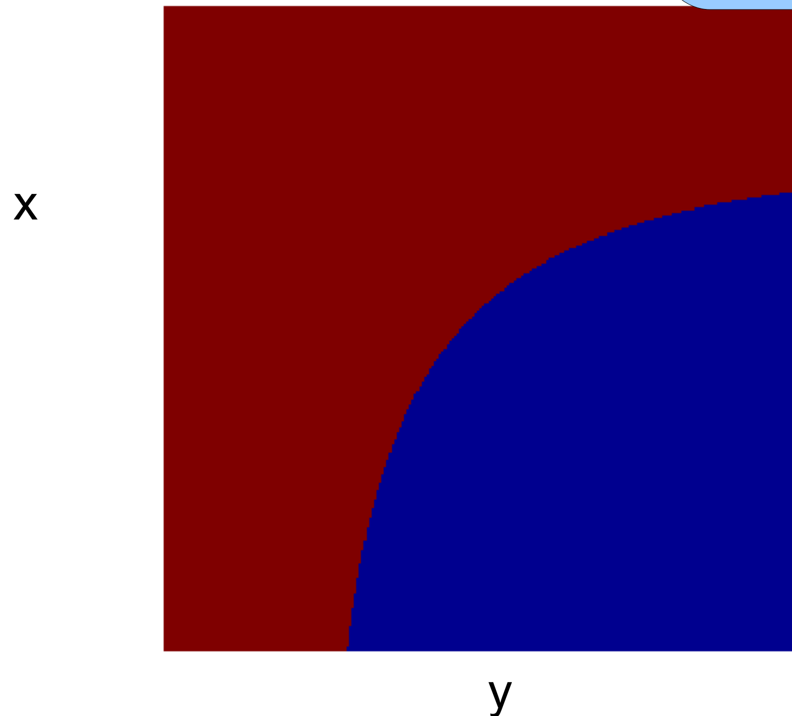
$$x^2 + 8xy + y^2 > 0$$



Non-Linear Classification

- We can generalize the form of the boundary
- $ax^2 + bxy + cy^2+d > 0$

Notice that this is linear in a,b,c, and d!

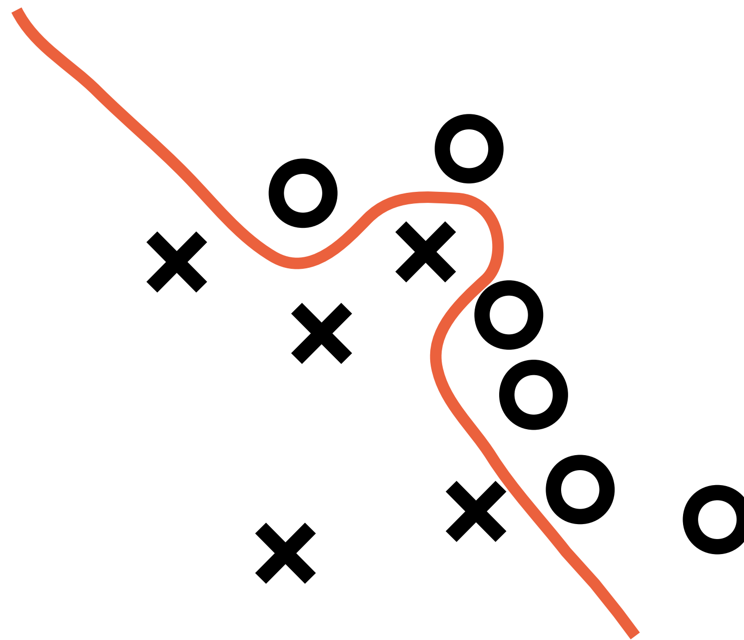


Easy Non-Linear Classification

- Take our original training example *(x,y)*, and make a new example
- $(x,y) \rightarrow (x^2, xy, y^2)$
- Now, plug these new vectors back into our linear classification machinery
- No new coding!

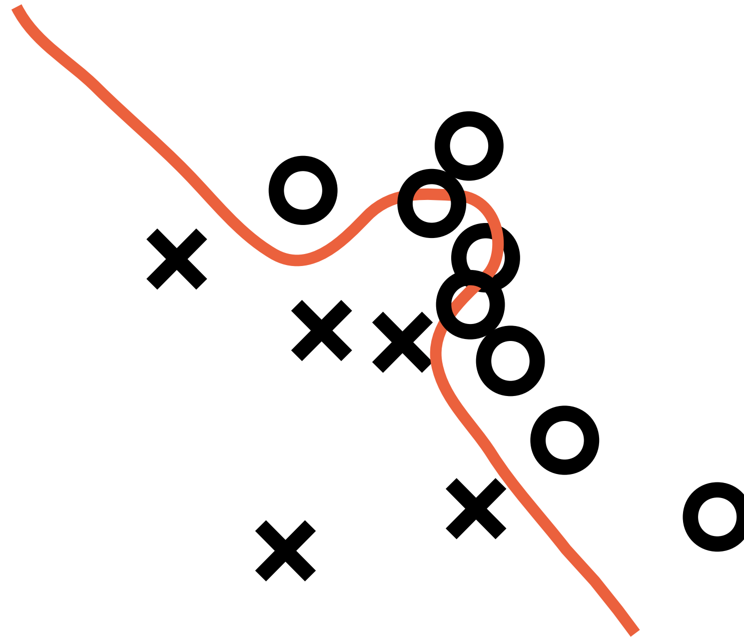
What's wrong with this decision boundary?

(Assume this is the training data)



What's wrong with this decision boundary?

- What if you tested on this data?
- The classifier has *over-fit* the data

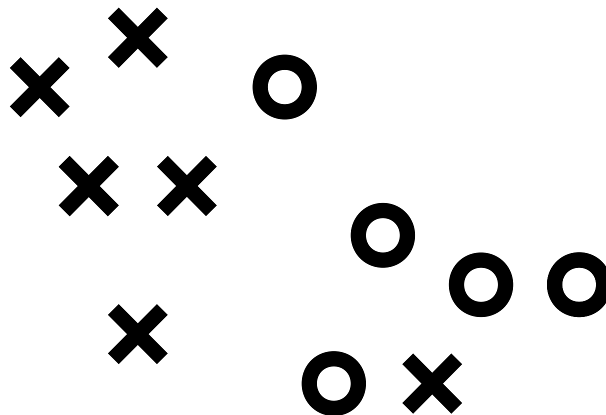


How to tell if your classifier is overfitting

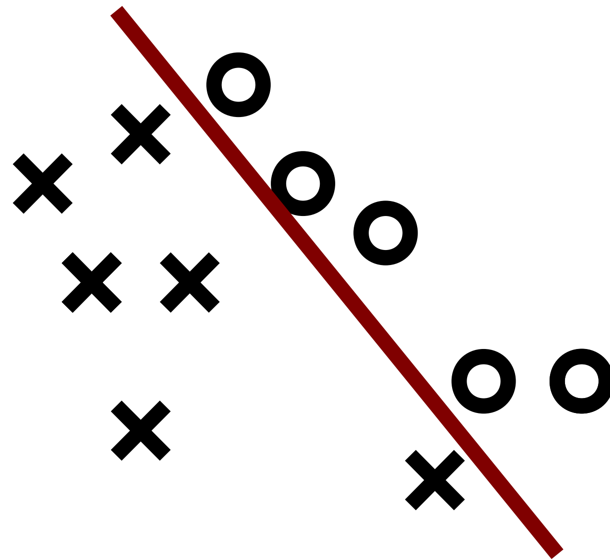
- Strategy #1:Hold out part of your data as a test set
- What if data is hard to come by?
- Strategy #2: *k*-fold cross-validation
  - Break the data set into *k* parts
  - For each part, hold a part out, then train the classifier and use the held out part as a test set
  - Slower than test-set method
  - More efficient use of limited data

The Support Vector Machinecd ..

- Boosted Classifiers and SVM's are probably the two most popular classifiers today
- I won't get into the math behind SVM's, if you are interested, you should take the pattern recognition course (highly recommended)

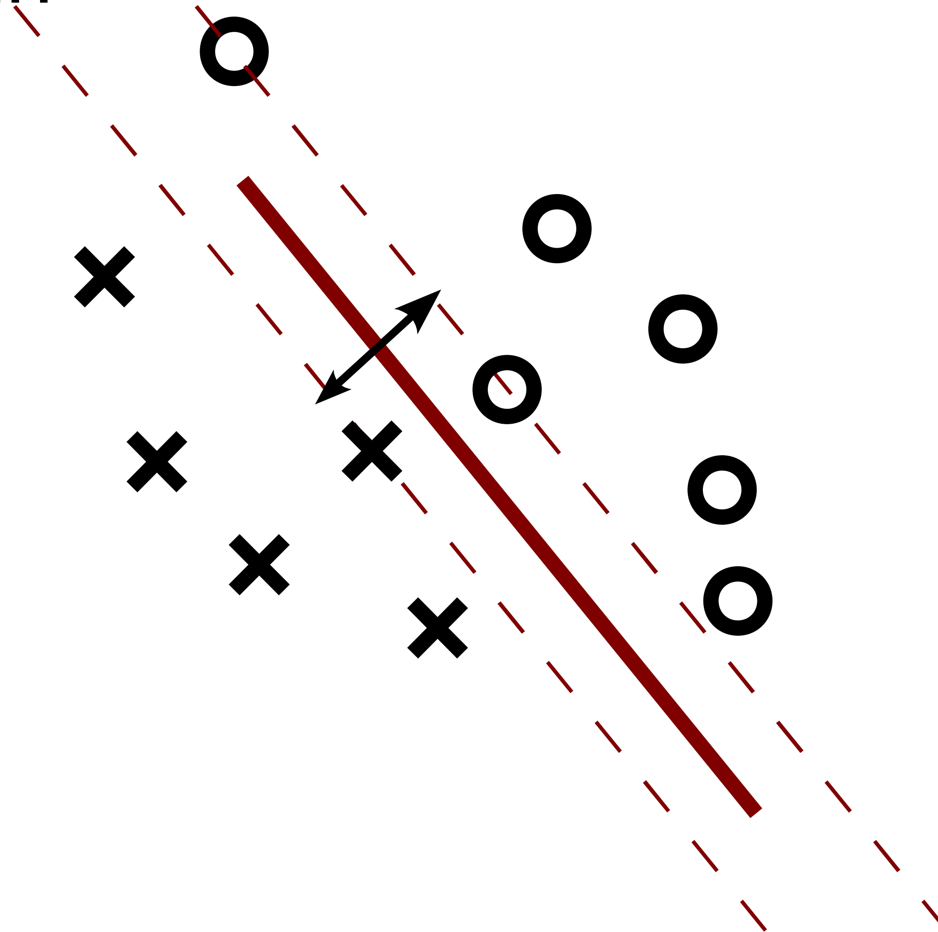


The Support Vector Machine



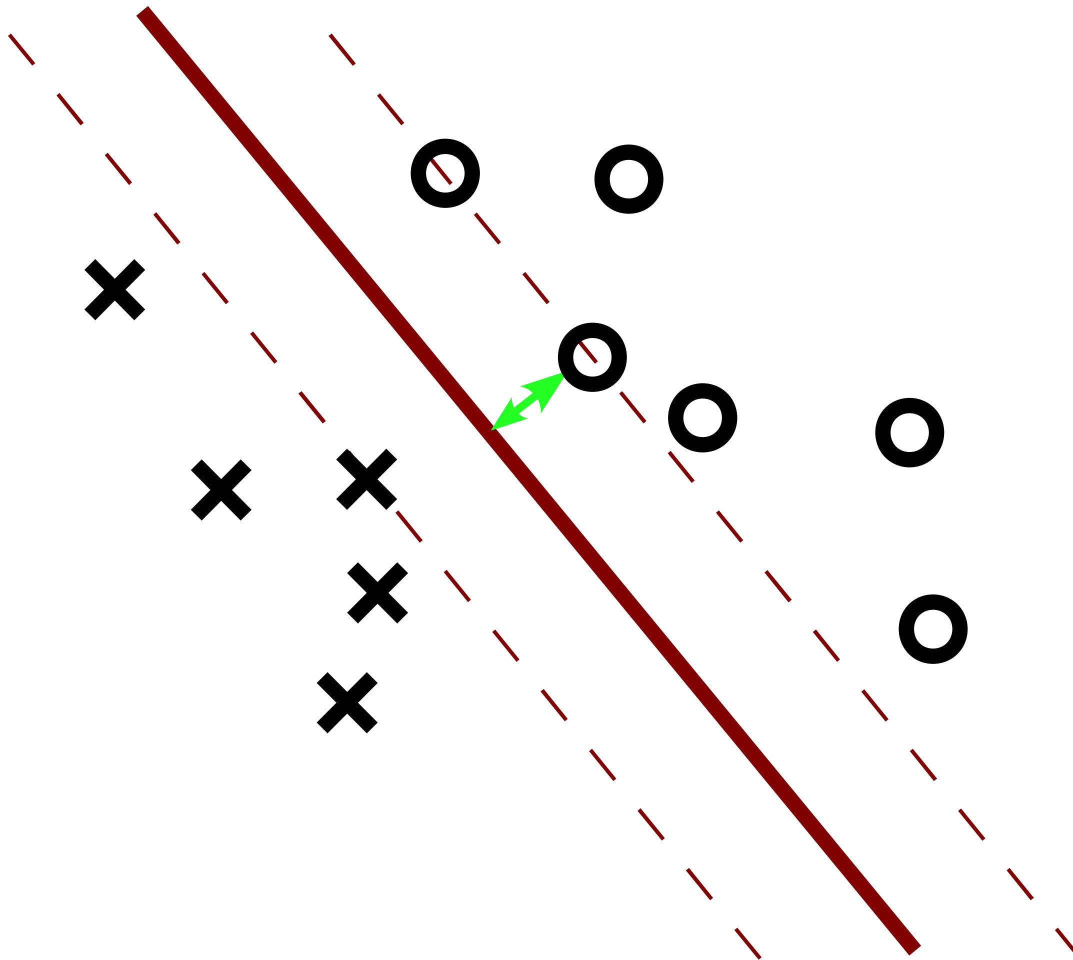
The Support Vector Machine

- Last time, we talked about different criteria for a good classifier
- Now, we will consider a criterion called the margin



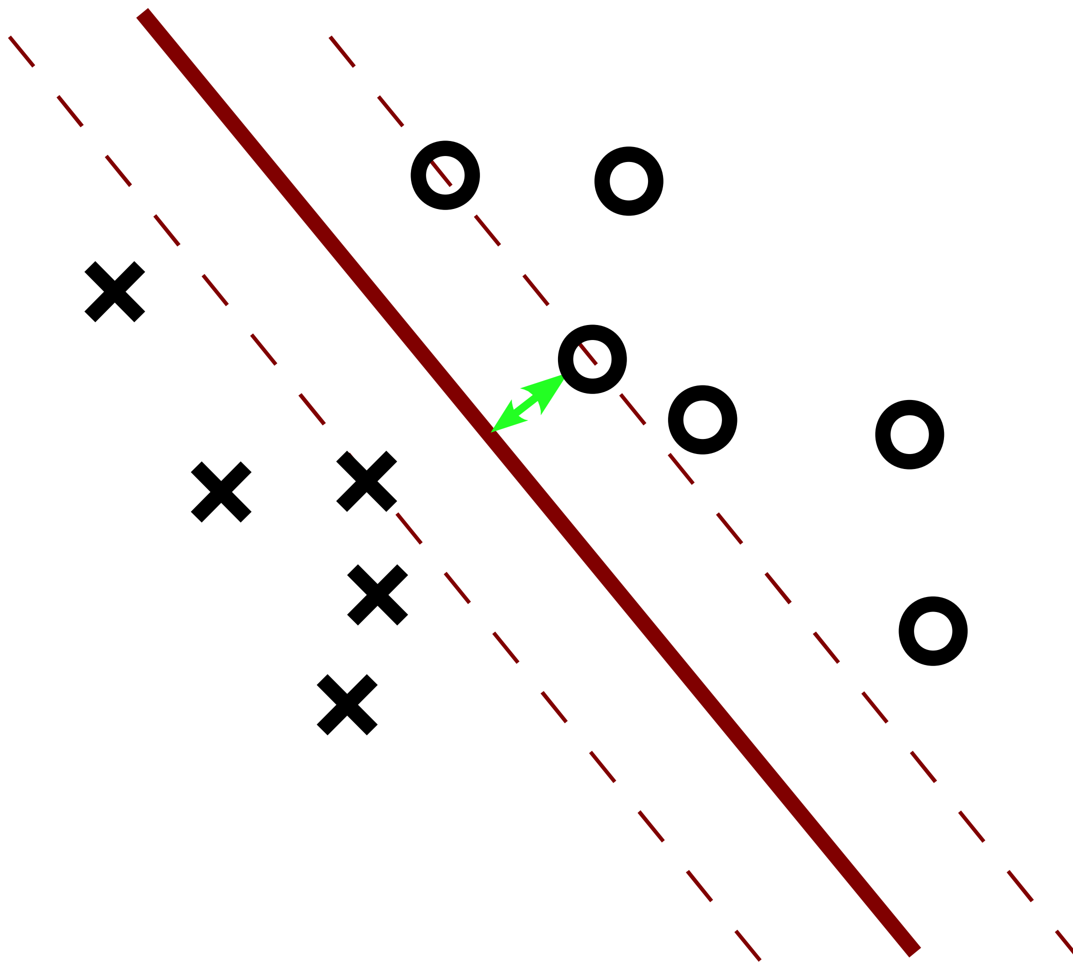
The Support Vector Machine

- Margin – minimum distance from a data point to the decision boundary



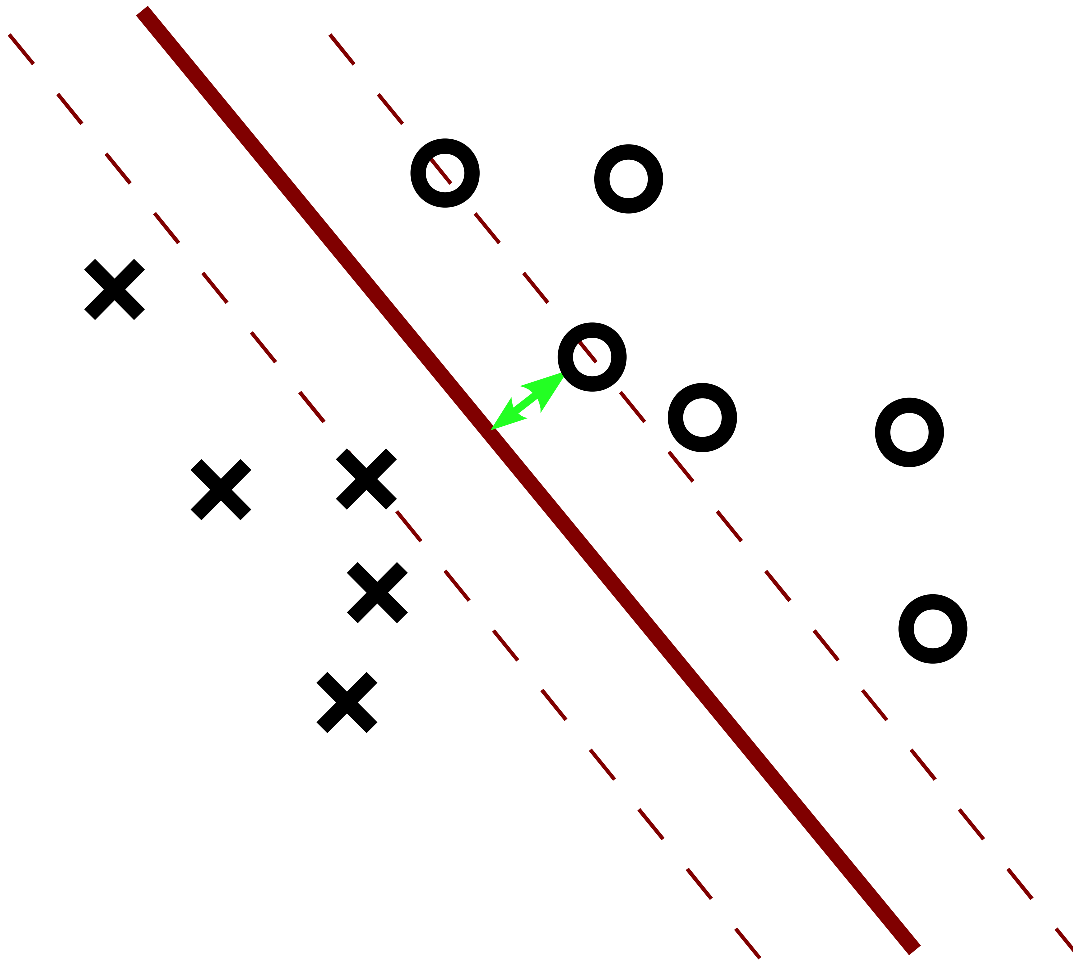
The Support Vector Machine

- The SVM finds the boundary that maximizes the margin



The Support Vector Machine

- Data points that are along the margin are called support vectors

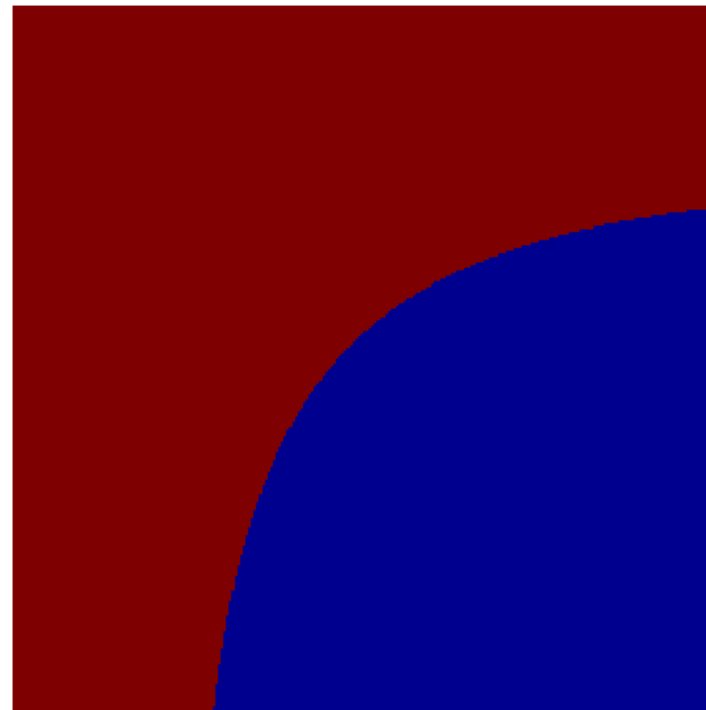


Non-Linear Classification in SVMs

- We will do the same trick as before

This is the decision boundary from
$x^2 + 8xy + y^2 > 0$

This is the same as making a new set of features, then doing linear classification



Non-Linear Classification in SVMs

The decision function can be expressed in terms of dot-products

$$f(x) = b + \sum_{i=1}^{N} \alpha_i y_i < x, x_i >$$

Each α will be zero unless the vector is a support vector

Non-Linear Classification in SVMs

- What if we wanted to do non-linear classification?
- We could transform the features and compute the dot product of the transformed features.
- But there may be an easier way!

The Kernel Trick

Let Φ(x) be a function that transforms x into a different space

$$\Phi : \mathbf{R}^d \mapsto \mathcal{H}.$$

A kernel function K is a function such that

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

Example (Burges 98)

If

$$\Phi(\mathbf{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}\, x_1 x_2 \\ x_2^2 \end{pmatrix}$$

Then

$$(\mathbf{x} \cdot \mathbf{y})^2 = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$$

This is called the polynomial kernel

Gaussian RBF Kernel

One of the most commonly used kernels

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2}$$

Equivalent to doing a dot-product in an *infinite* dimensional space

The Kernel Trick

So, with a kernel function *K*, the new classification rule is

$$f(\mathbf{x}) = \sum_{i=1}^{N_S} \alpha_i y_i \Phi(\mathbf{s}_i) \cdot \Phi(\mathbf{x}) + b = \sum_{i=1}^{N_S} \alpha_i y_i K(\mathbf{s}_i, \mathbf{x}) + b$$

Basic ideas:

- Computing the kernel function should be easier than computing a dot-product in the transformed space
- Other algorithms, like logistic regression can also be “kernelized”

So what if I want to use an SVM?

There are well-developed packages with Python and MATLAB interfaces

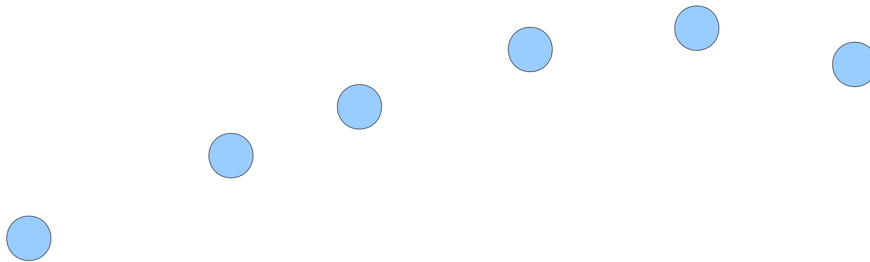
- libSVM
- SVMLight
- SVMTorch

From Edges to Lines

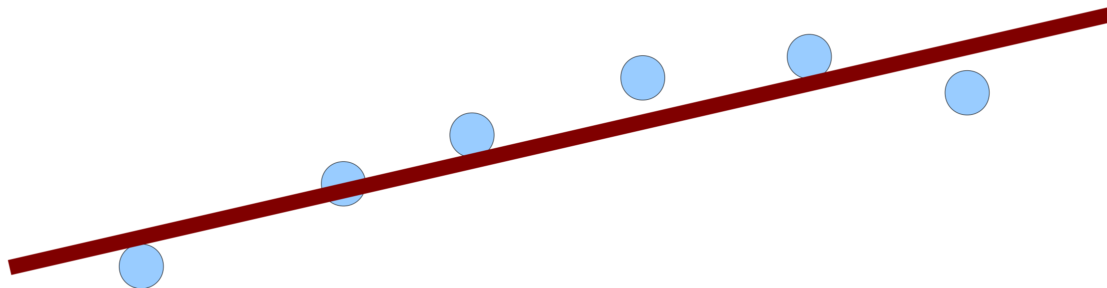
We’ve talked about detecting Edges, but how can we extract lines from those edges?

We'll start with a very basic, least squares approach

We have edge points



We need a line



Start with classic equation of line

$$y = mx + b$$

Remember this?

m – slope

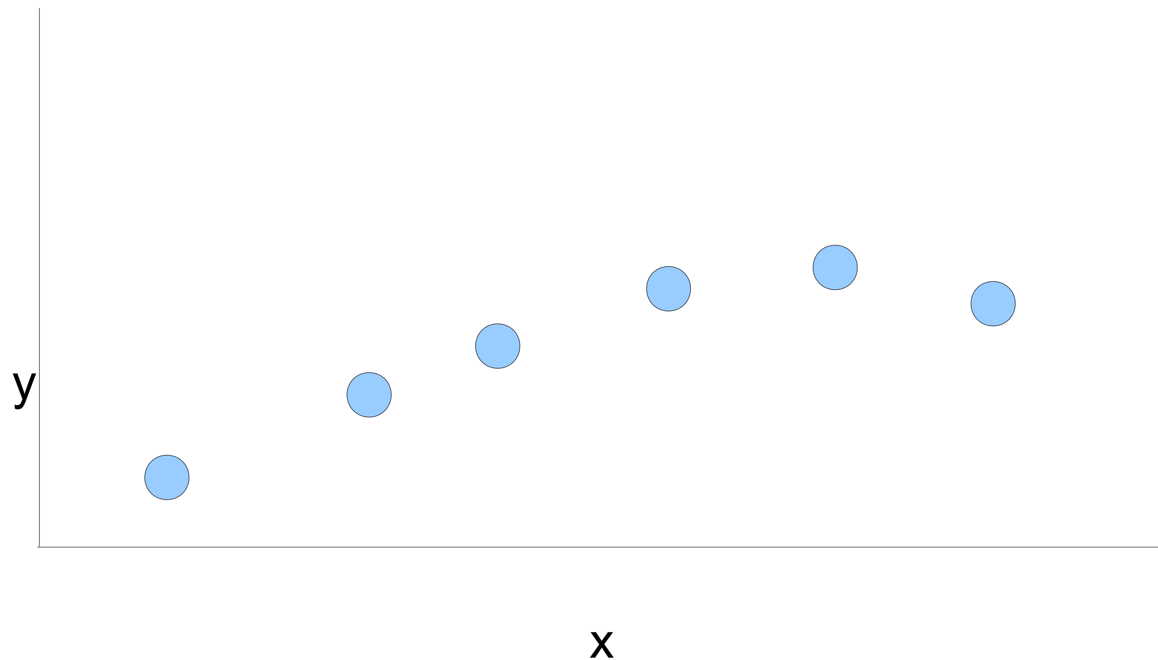
b - y-intercept

Fitting the line

In classification, we fit a separating boundary, or line, to maximize the probability of the data

Here, we will minimize error

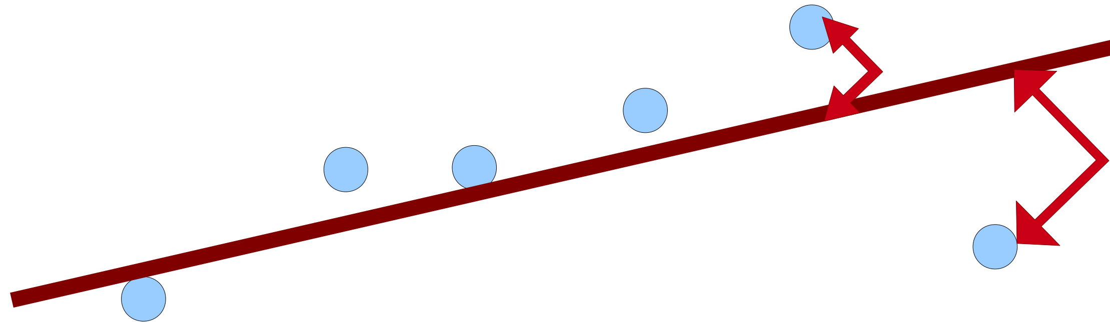
Convert this picture into numbers



Call every point (x_i, y_i)

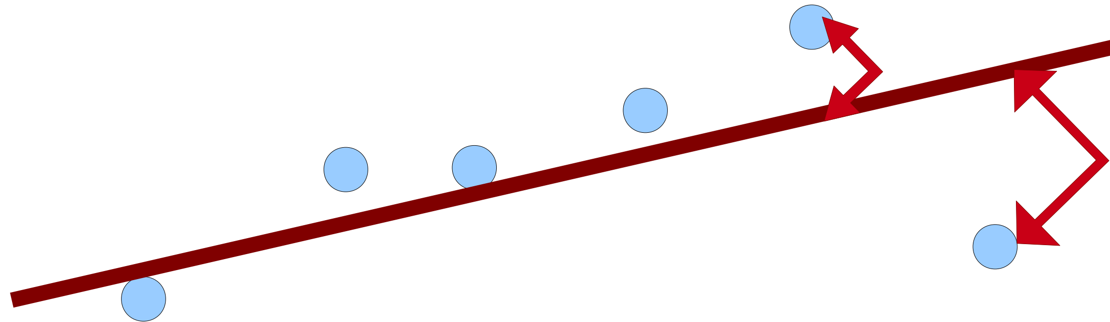
Our new criterion

We want to fit a line so that the squared difference between the point and the line's "prediction" is minimized



Mathematically

$$L(m,b) = \sum_{i=1}^{N} (mx_i + b - y_i)^2$$



See why it's called least squares?

A little calculus

$$\frac{\partial L(m,b)}{\partial m} = \sum_{i=1}^{N} 2\left(mx_i + b - y_i\right)x_i$$

$$\frac{\partial L(m,b)}{\partial b} = \sum_{i=1}^{N} 2\left(mx_i + b - y_i\right)$$

Set these to zero and solve

Let's take a matrix view of this

$$A = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ \vdots & \vdots \\ x_{N-1} & 1 \\ x_N & 1 \end{bmatrix} \qquad \mathbf{m} = \begin{bmatrix} m \\ b \end{bmatrix}$$

Let's take a matrix view of this

So, we can write the prediction at multiple points as

$$\hat{\mathbf{y}} = A\mathbf{m}$$

$$A = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ \vdots & \vdots \\ x_{N-1} & 1 \\ x_N & 1 \end{bmatrix} \qquad \mathbf{m} = \begin{bmatrix} m \\ b \end{bmatrix}$$

Going back

Notice that

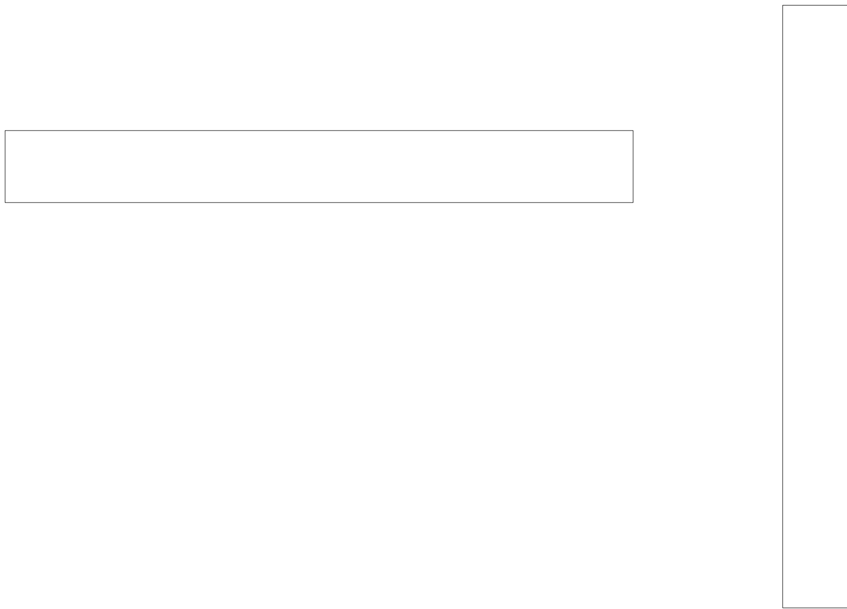
$$\frac{\partial L(m,b)}{\partial m} = \sum_{i=1}^{N} 2\left(mx_i + b - y_i\right)x_i$$

Can be rewritten as

$$A(:,1)^T(A\mathbf{m} - y)$$

(I'm using MATLAB notation for A(:,1))

Sketch it Out



$$A(:,1)^T(A\mathbf{m} - y)$$

$$\frac{\partial L(m,b)}{\partial m} = \sum_{i=1}^{N} 2\left(mx_i + b - y_i\right)x_i$$

Now we can do the same thing again

Notice that

$$\frac{\partial L(m,b)}{\partial b} = \sum_{i=1}^{N} 2\left(mx_i + b - y_i\right)$$

Can be rewritten as

$$A(:,2)^T(A\mathbf{m} - y)$$

(I'm using MATLAB notation for A(:,2))

Now, setting everything to zero

$$A^T(A\mathbf{m} - y) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$A^T A\mathbf{m} = A^T\mathbf{y}$$

$$\mathbf{m} = \left(A^T A\right)^{-1} A^T\mathbf{y}$$

Also called pseudo-inverse

You will often see this appear as

$$\mathbf{x} = \left(A^T A\right)^{-1} A^T \mathbf{h}$$

Why all the matrix hassle?

This will allow us to easily generalize to higher-dimensions