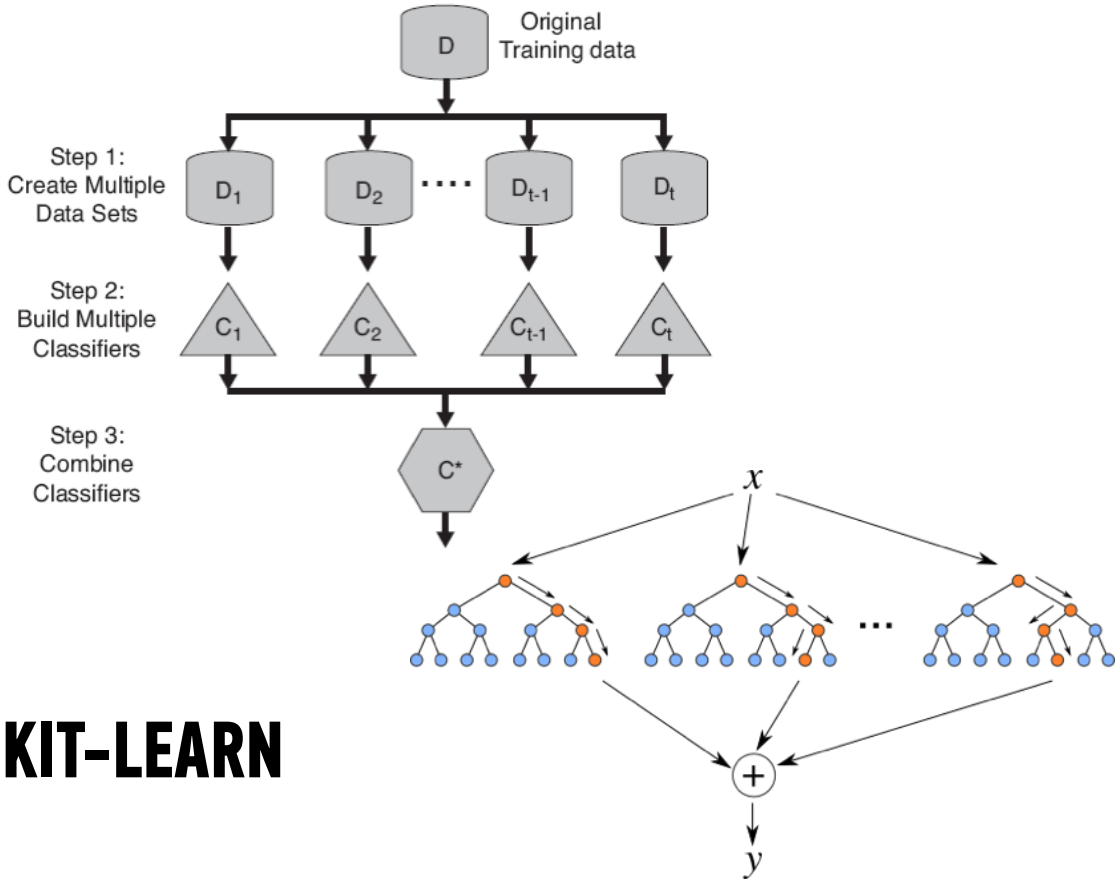
GENERAL ASSEMBLY

INTRO TO DATA SCIENCE

SUPPORT VECTOR MACHINES

I. ENSEMBLE TECHNIQUES
II. BOOTSTRAP & BAGGING
III. RANDOM FORESTS

LAB:
IV. RANDOM FOREST IN SCIKIT-LEARN



QUESTIONS?

WHAT WAS THE MOST INTERESTING THING YOU LEARNED?

WHAT WAS THE HARDEST TO GRASP?

AGENDA

I. SUPPORT VECTOR MACHINES (SVM)
II. SLACK VARIABLES
III. NONLINEAR CLASSIFICATION WITH KERNELS

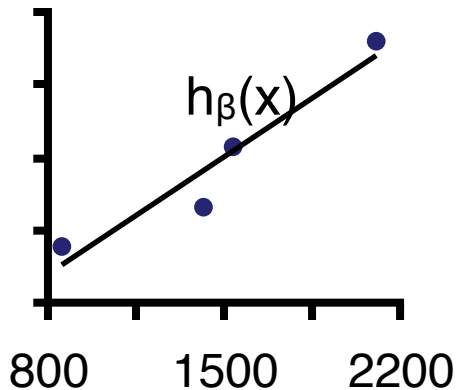
IV. LAB ON SVM'S IN PYTHON

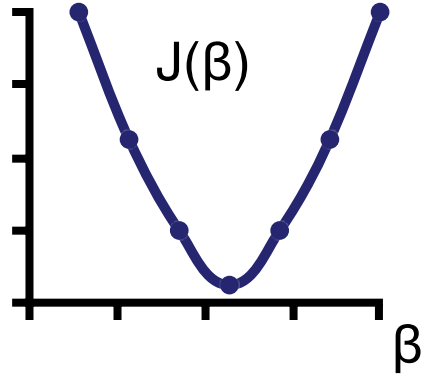
SUPPORT VECTOR MACHINES

Remember linear regression:

Training set : (x, y)

Hypothesis Function : $h_\beta(x)$



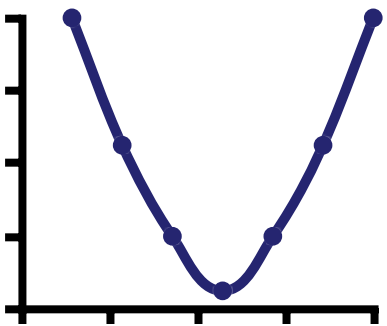


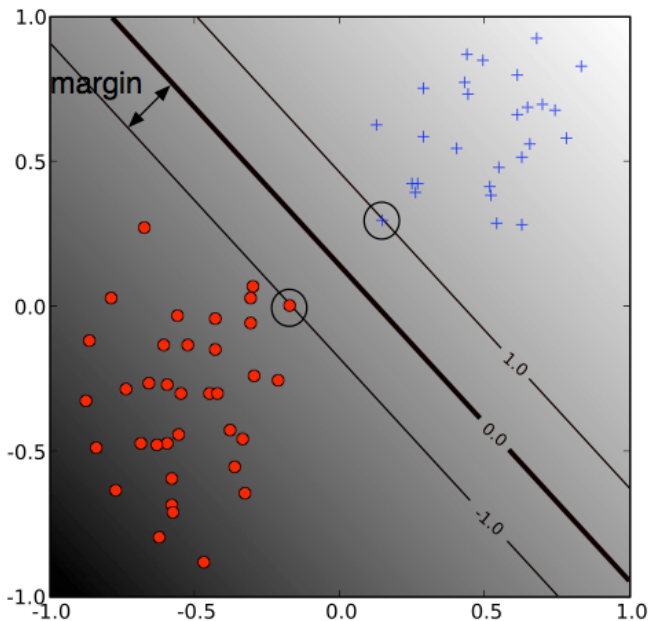
Cost Function: J(β)

GOAL

Find the values for the parameters β that minimize the cost function over the set of training data

Support vector machines are binary linear classifier whose decision boundary is explicitly constructed to minimize generalization error.





Decision boundary is derived using geometric reasoning

(as opposed to the algebraic reasoning we've used to derive other classifiers).

generalization error *is equated with the geometric concept of* **margin**, *which is the region along the decision boundary that is free of data points.*

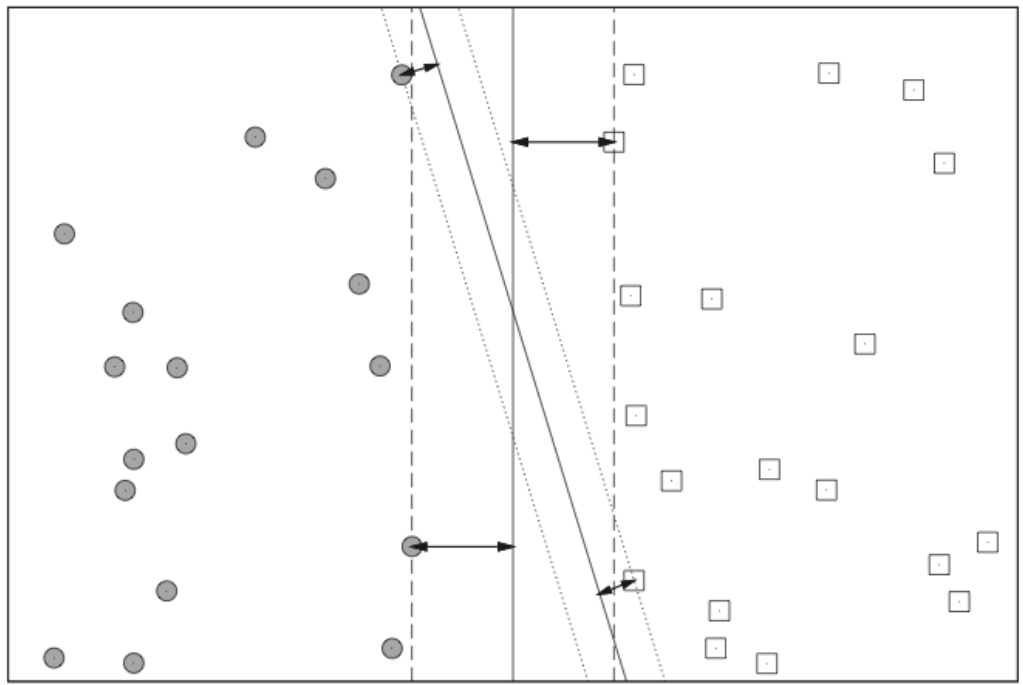
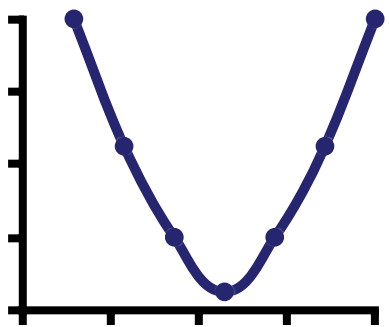
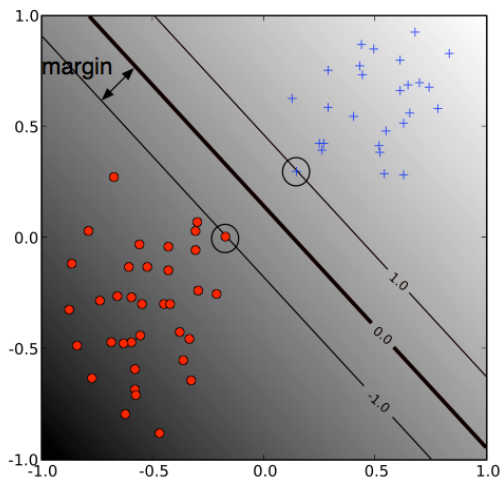


FIGURE 18-4. Two decision boundaries and their margins. Note that the vertical decision boundary has a wider margin than the other one. The arrows indicate the distance between the respective support vectors and the decision boundary.

source: Data Analysis with Open Source Tools, by Philipp K. Janert. O'Reilly Media, 2011.

The goal of an SVM is to create the linear decision boundary with the ***largest margin****. This is commonly called the* **maximum margin hyperplane (MMH)**.





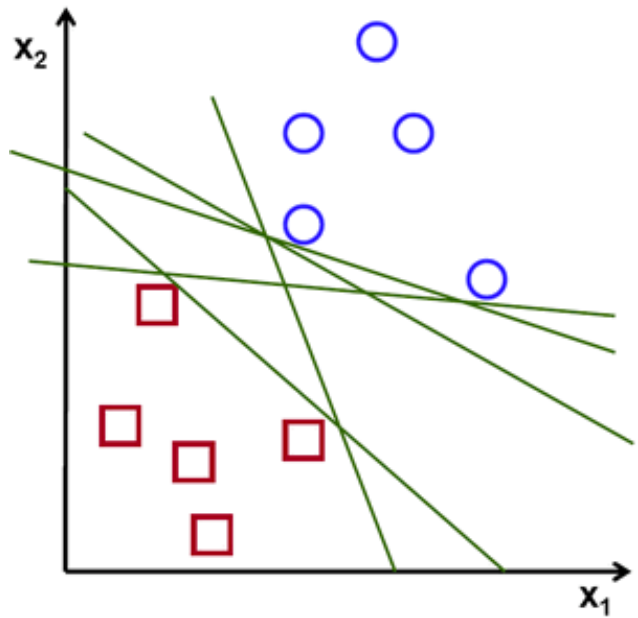
NOTE

A *hyperplane* is just a high-dimensional generalization of a line.

MAXIMUM MARGIN HYPERPLANES (MMH)

Q: How is the decision boundary (MMH) derived?

so many possible hyperplanes, which one to choose?



Q: How is the decision boundary (MMH) derived?

A: By the **discriminant function**,

$$f(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{x} + b.$$

such that w is the weight vector and b is the bias.

Q: How is the decision boundary (MMH) derived?

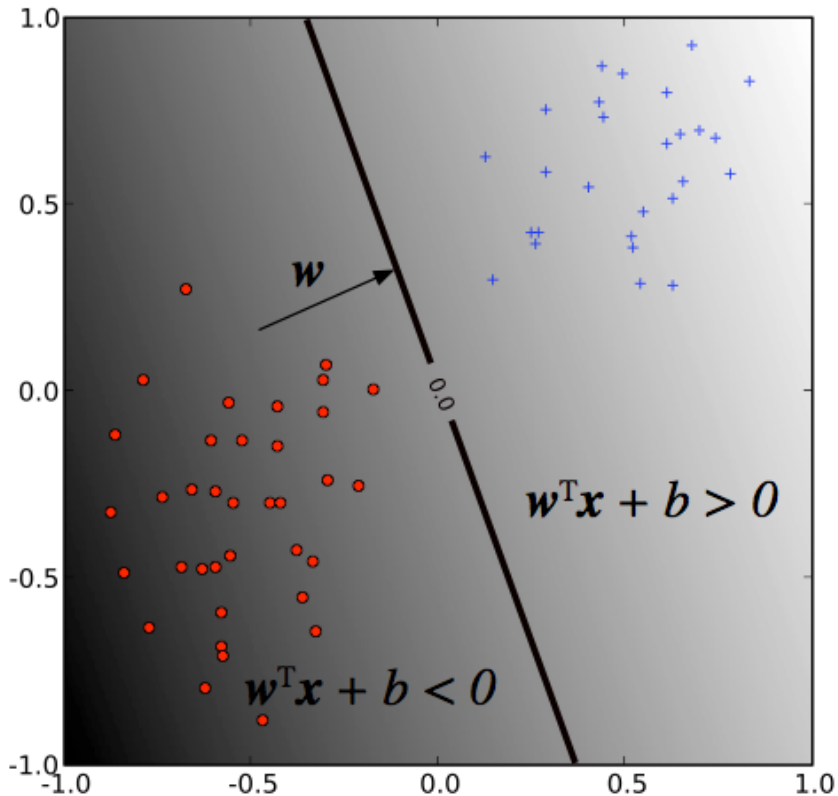
A: By the **discriminant function**,

$$f(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{x} + b.$$

such that w is the weight vector and b is the bias.

The sign of f(x) determines the (binary) class label of a record x.

MAXIMUM MARGIN HYPERPLANES



source: http://pyml.sourceforge.net/doc/howto.pdf

NOTE

The weight vector determines the *orientation* of the decision boundary.

The bias determines its *translation* from the origin.

MAXIMUM MARGIN HYPERPLANES

As we said before, SVM solves for the decision boundary that minimizes generalization error, or equivalently, that has the maximum margin.

Q: Why are these the same thing?

MAXIMUM MARGIN HYPERPLANES

As we said before, SVM solves for the decision boundary that minimizes generalization error, or equivalently, that has the maximum margin.

Q: Why are these the same thing?
A: Because using the MMH as the decision boundary minimizes the probability that a small perturbation in the position of a point produces a classification error.

MAXIMUM MARGIN HYPERPLANES

As we said before, SVM solves for the decision boundary that minimizes generalization error, or equivalently, that has the maximum margin.

Q: Why are these the same thing?

A: Because using the MMH as the decision boundary minimizes the probability that a small perturbation in the position of a point produces a classification error.

NOTE

Intuitively, the wider the margin, the clearer the distinction between classes.

Selecting the MMH is a straightforward exercise in analytic geometry (we won't go through the details here).

In particular, this task reduces to the optimization of a **convex** *objective function.*

Selecting the MMH is a straightforward exercise in analytic geometry (we won't go through the details here).

In particular, this task reduces to the optimization of a **convex** *objective function.*

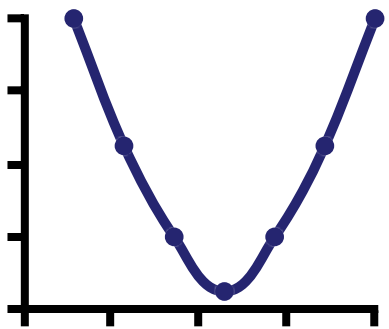
Remember what convex means in this context...?

MAXIMUM MARGIN HYPERPLANES

Selecting the MMH is a straightforward exercise in analytic geometry (we won't go through the details here).

In particular, this task reduces to the optimization of a **convex** *objective function.*

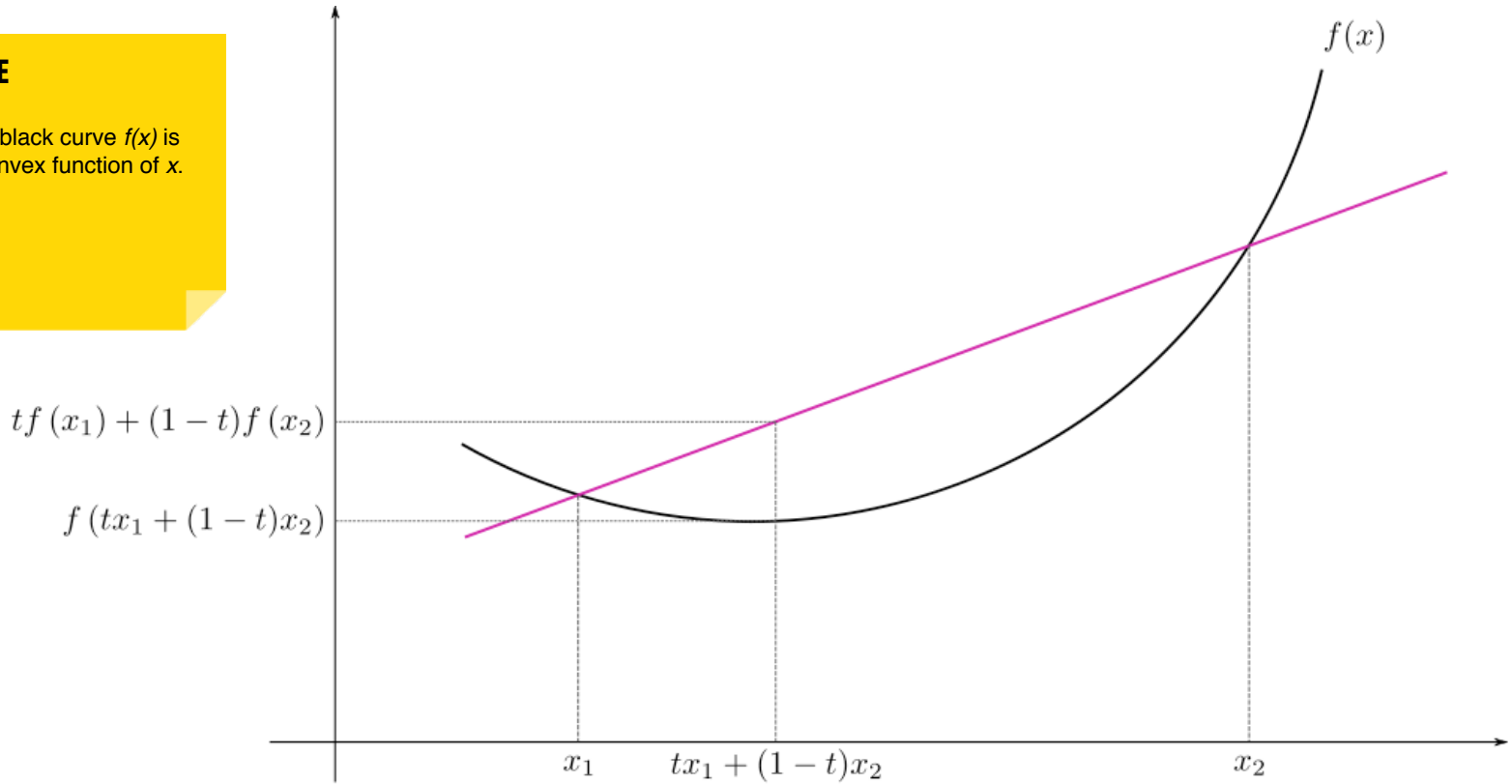
Nice! Convex optimization problems are guaranteed to give **global optima** *(and they're easy to solve numerically too).*



CONVEX FUNCTIONS

NOTE

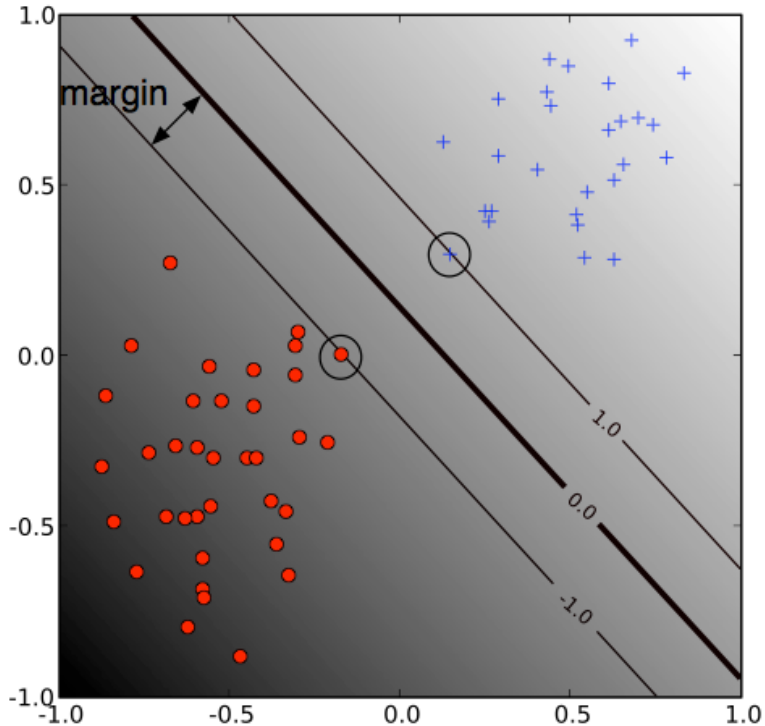
The black curve *f(x)* is a convex function of *x*.



source: http://en.wikipedia.org/wiki/File:ConvexFunction.svg

Notice that the margin depends only on the points that are nearest to the decision boundary.

These points are called the **support vectors**.



MAXIMUM MARGIN HYPERPLANES

All of the decision boundaries we've seen so far have split the data perfectly; eg, the data are (perfectly) **linearly separable**, *and therefore the training error is 0.*

The optimization problem that this SVM solves is:

$$\begin{aligned} \underset{\mathbf{w},b}{\text{minimize}} \quad & \tfrac{1}{2}||\mathbf{w}||^2 \\ \text{subject to:} \quad & y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) \geq 1 \quad i = 1, \ldots, n\,. \end{aligned}$$

Note: Those double vertical lines mean the magnitude of the vector:

$$\|\boldsymbol{x}\| := \sqrt{x_1^2 + \cdots + x_n^2}.$$

https://en.wikipedia.org/wiki/Norm_(mathematics)

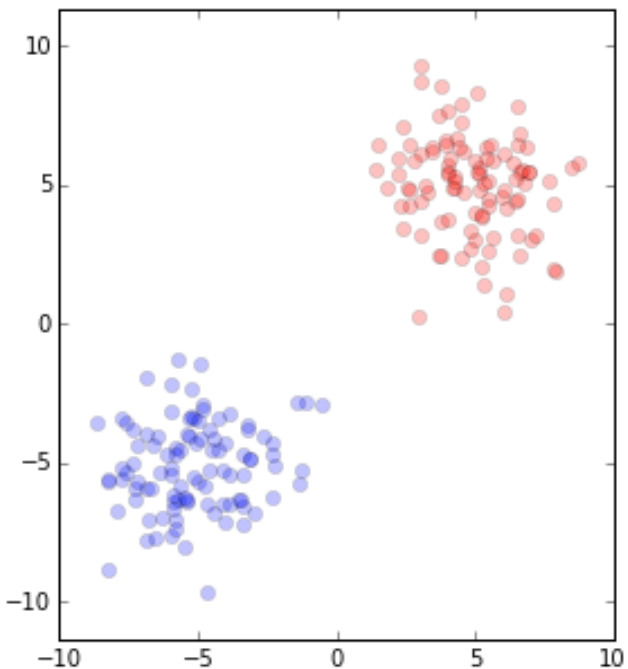
NOTE

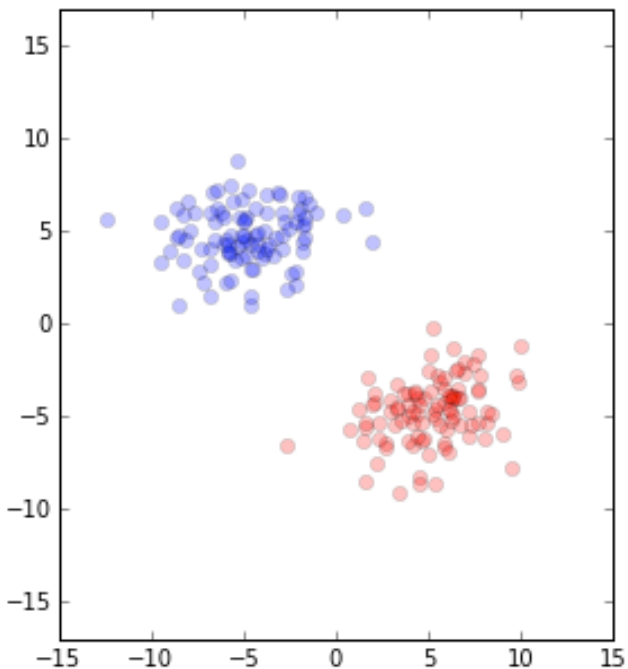
This type of optimization problem is called a *quadratic program.*

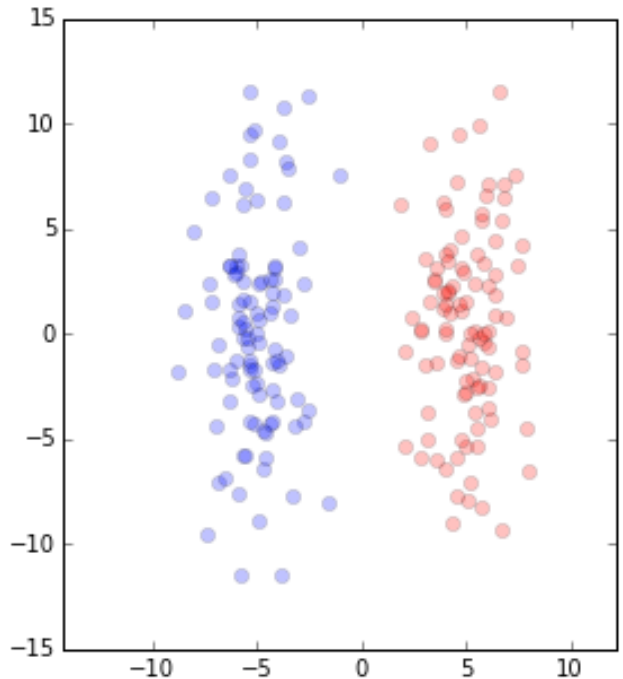
The result of this qp is the *hard margin classifier* we've been discussing.

MAXIMUM MARGIN HYPERPLANES

You do it: find the MMH

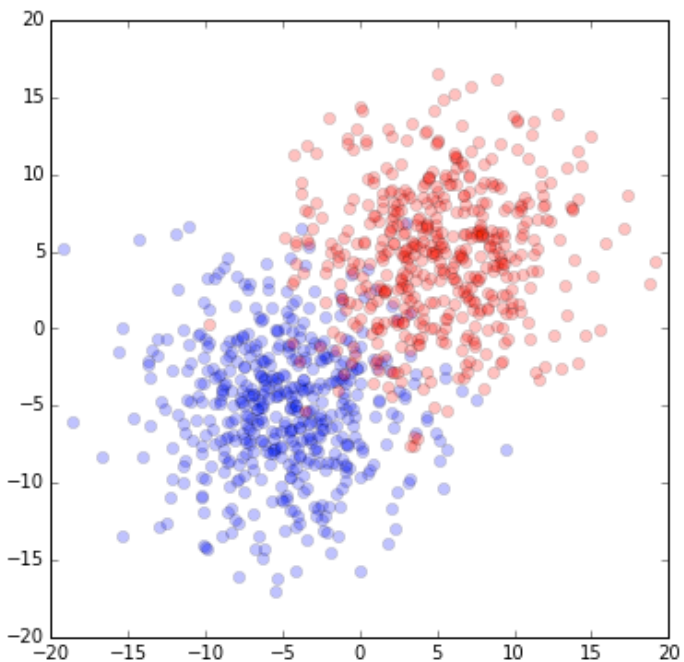






MAXIMUM MARGIN HYPERPLANES

What if data is not perfectly separable with a linear boundary?

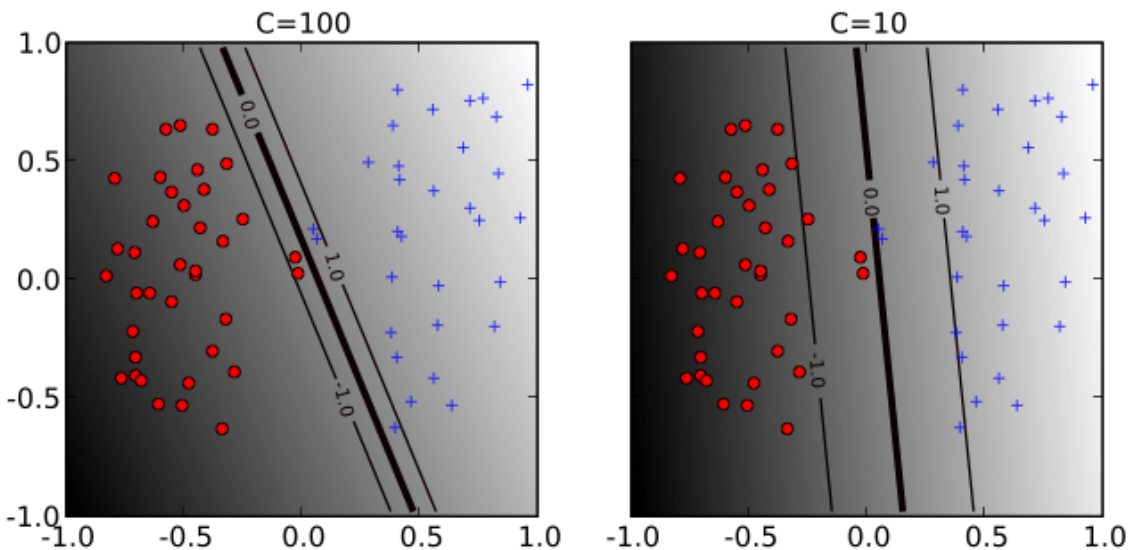


?

SLACK VARIABLES

SLACK VARIABLES

We can trade some training error in order to get a larger margin (remember the bias-variance trade-off)



source: http://pyml.sourceforge.net/doc/howto.pdf

Slack variables ξ_i generalize the optimization problem to permit some misclassified training records (which come at a cost C).

The resulting **soft margin classifier** *is given by:*

$$\begin{aligned} \underset{\mathbf{w},b}{\text{minimize}} \quad & \tfrac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to:} \quad & y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0. \end{aligned}$$

(This is another example of bias-variance tradeoff)

The soft-margin optimization problem can be rewritten as:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \textstyle\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_j \\ \text{subject to:} \quad & \textstyle\sum_{i=1}^{n} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C. \end{aligned}$$

The soft-margin optimization problem can be rewritten as:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_j \\ \text{subject to:} \quad & \sum_{i=1}^{n} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C. \end{aligned}$$

Notice that this expression depends on the features x_i only via the **inner product**

$$\langle x_i, x_j \rangle = x_i^T x_j$$

The inner product is an operation that takes two vectors and returns a real number.

The fact that we can rewrite the optimization problem in terms of the inner product means that we don't actually have to do any calculations in the feature space K.

The inner product is an operation that takes two vectors and returns a real number.

The fact that we can rewrite the optimization problem in terms of the inner product means that we don't actually have to do any calculations in the feature space K.

This will have a very powerful consequence as we shall shortly see.

Quick recall exercise. Calculate these dot products:

$$v = \begin{bmatrix} 1 & 3 & 9 & 2 \end{bmatrix} \qquad v.w = ?$$

$$u = \begin{bmatrix} 2 & 4 & 6 & 7 \end{bmatrix} \qquad v.u = ?$$

$$w = \begin{bmatrix} 2 & 3 & 6 & 5 \end{bmatrix} \qquad w.u = ?$$

Quick recall exercise. Calculate these dot products:

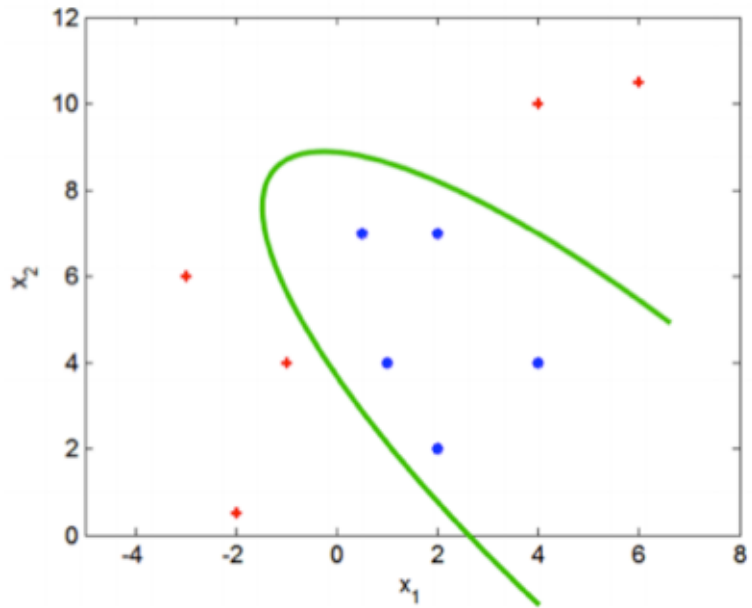
$$v = \begin{bmatrix} 1 & 3 & 9 & 2 \end{bmatrix} \qquad v.w = 75$$

$$u = \begin{bmatrix} 2 & 4 & 6 & 7 \end{bmatrix} \qquad v.u = 82$$

$$w = \begin{bmatrix} 2 & 3 & 6 & 5 \end{bmatrix} \qquad w.u = 87$$

NONLINEAR CLASSIFICATION

Suppose we need a more complex classifier than a linear decision boundary allows.



Suppose we need a more complex classifier than a linear decision boundary allows.

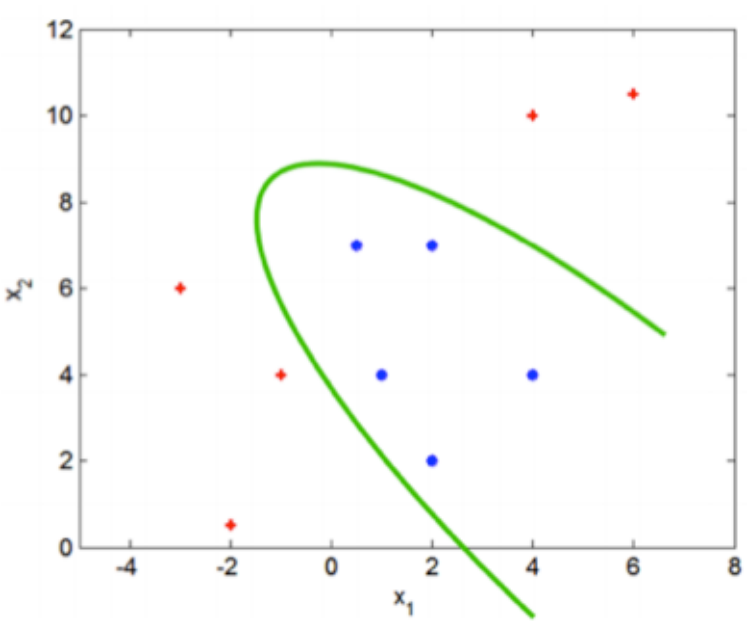
One possibility is to add nonlinear combinations of features to the data, and then to create a linear decision boundary in the enhanced (higher-dimensional) feature space.

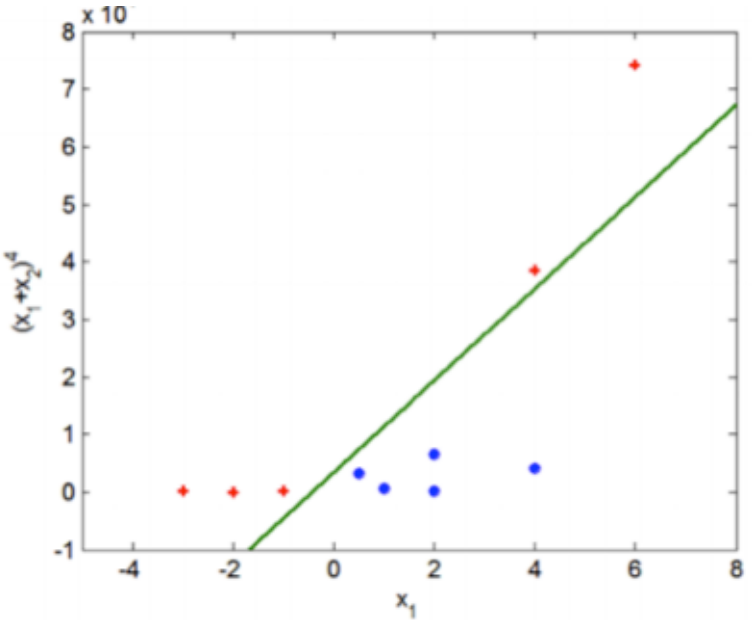
Suppose we need a more complex classifier than a linear decision boundary allows.

One possibility is to add nonlinear combinations of features to the data, and then to create a linear decision boundary in the enhanced (higher-dimensional) feature space.

This linear decision boundary will be mapped to a nonlinear decision boundary in the original feature space.

NONLINEAR CLASSIFICATION





original feature space *K*

higher-dim feature space *K'*

The logic of this approach is sound, but there are a few problems with this version.

The logic of this approach is sound, but there are a few problems with this version.

In particular, this will not scale well, since it requires many high-dimensional calculations.

The logic of this approach is sound, but there are a few problems with this version.

In particular, this will not scale well, since it requires many high-dimensional calculations.

It will likely lead to more complexity (both modeling complexity and computational complexity) than we want.

Let's hang on to the logic of the previous example, namely:

Let's hang on to the logic of the previous example, namely:

- remap the feature vectors x_i into a higher-dimensional space K'

- create a linear decision boundary in K'

- back out the nonlinear decision boundary in K from the result

Let's hang on to the logic of the previous example, namely:

- remap the feature vectors x_i into a higher-dimensional space K'

- create a linear decision boundary in K'

- back out the nonlinear decision boundary in K from the result

But we want to save ourselves the trouble of doing a lot of additional high-dimensional calculations. How can we do this?

Recall that our optimization problem depends on the features only through the inner product $x^T x$:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_j \\ \text{subject to:} \quad & \sum_{i=1}^{n} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C. \end{aligned}$$

Recall that our optimization problem depends on the features only through the inner product $x^T x$:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_j \\ \text{subject to:} \quad & \sum_{i=1}^{n} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C. \end{aligned}$$

We can replace this inner product with a more general function that has the same type of output as the inner product.

Formally, we can think of the inner product as a map that sends two vectors in the feature space K into the real line $\mathbb{R}$

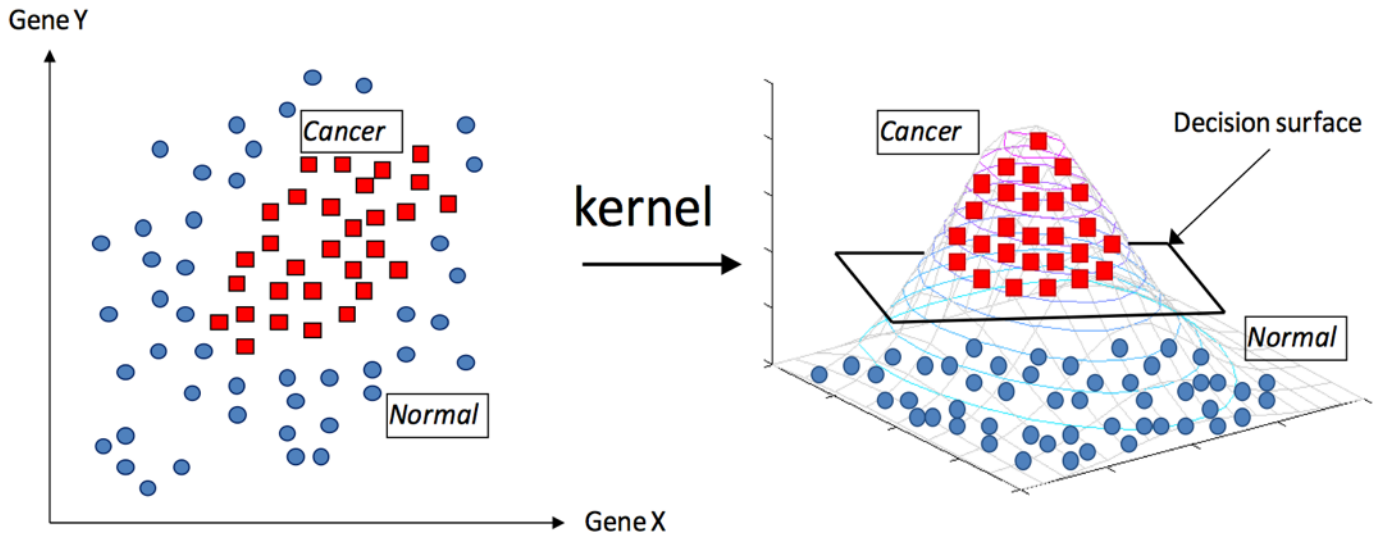
Formally, we can think of the inner product as a map that sends two vectors in the feature space K into the real line $\mathbb{R}$

We can replace this with a generalization of the inner product called a **kernel function** *that maps two vectors in a higher-dimensional feature space K' into* $\mathbb{R}$

Nonlinear *applications of SVM rely on an implicit mapping* Φ *that sends vectors from the original feature space* K *into a higher-dimensional feature space* K'*. This mapping is called a* ***kernel.***

Nonlinear classification in K *is then obtained by creating a linear decision boundary in* K'*.*

THE KERNEL TRICK



If a linear decision boundary cannot be found in the original space, we can map into a higher dimensional space and find the separating surface.

The upshot is that we can use a kernel function to implicitly train our model in a higher-dimensional feature space, without incurring additional computational complexity!

The upshot is that we can use a kernel function to implicitly train our model in a higher-dimensional feature space, without incurring additional computational complexity!

As long as the kernel function satisfies certain conditions, our conclusions above regarding the MMH continue to hold.

The upshot is that we can use a kernel function to implicitly train our model in a higher-dimensional feature space, without incurring additional computational complexity!

As long as the kernel function satisfies certain conditions, our conclusions above regarding the MMH continue to hold.

NOTE

These conditions are contained in a result called *Mercer's theorem*.

The upshot is that we can use a kernel function to implicitly train our model in a higher-dimensional feature space, without incurring additional computational complexity!

As long as the kernel function satisfies certain conditions, our conclusions above regarding the MMH continue to hold.

In other words, no algorithmic changes are necessary, and all the benefits of a linear SVM are maintained.

some popular kernels:

linear kernel $k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$

polynomial kernel $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\mathsf{T}\mathbf{x}' + 1)^d$

Gaussian kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma||\mathbf{x}-\mathbf{x}'||^2)$

some popular kernels:

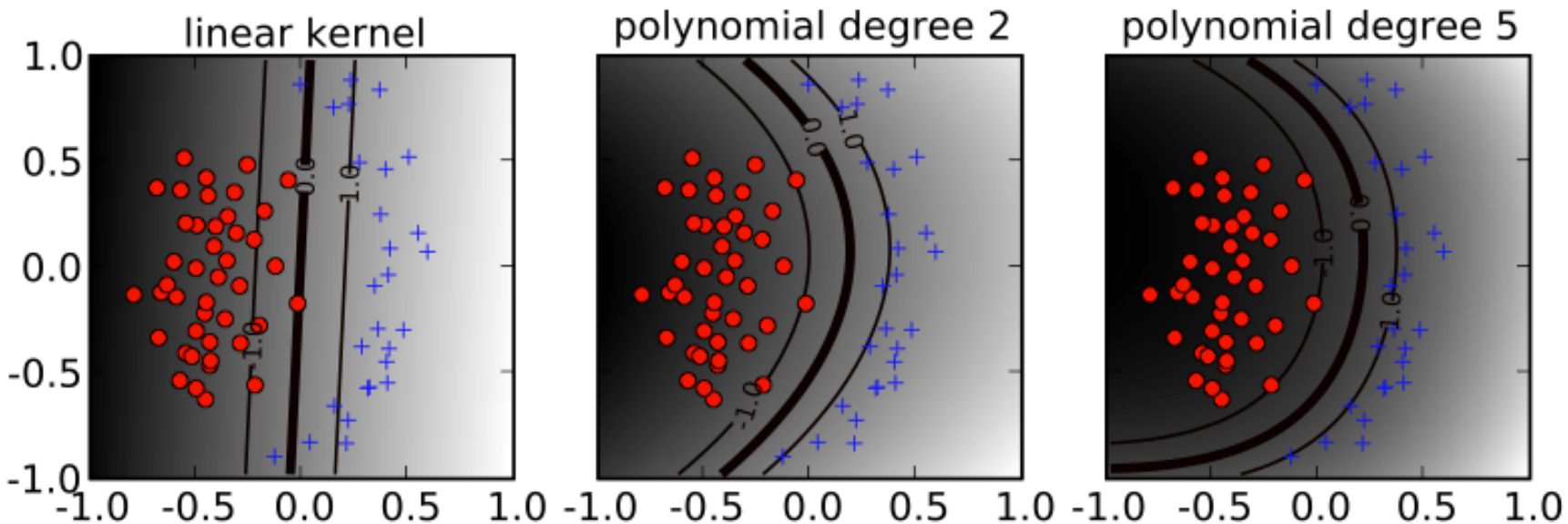
linear kernel $k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$

polynomial kernel $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\mathsf{T}\mathbf{x}' + 1)^d$

Gaussian kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma||\mathbf{x}-\mathbf{x}'||^2)$

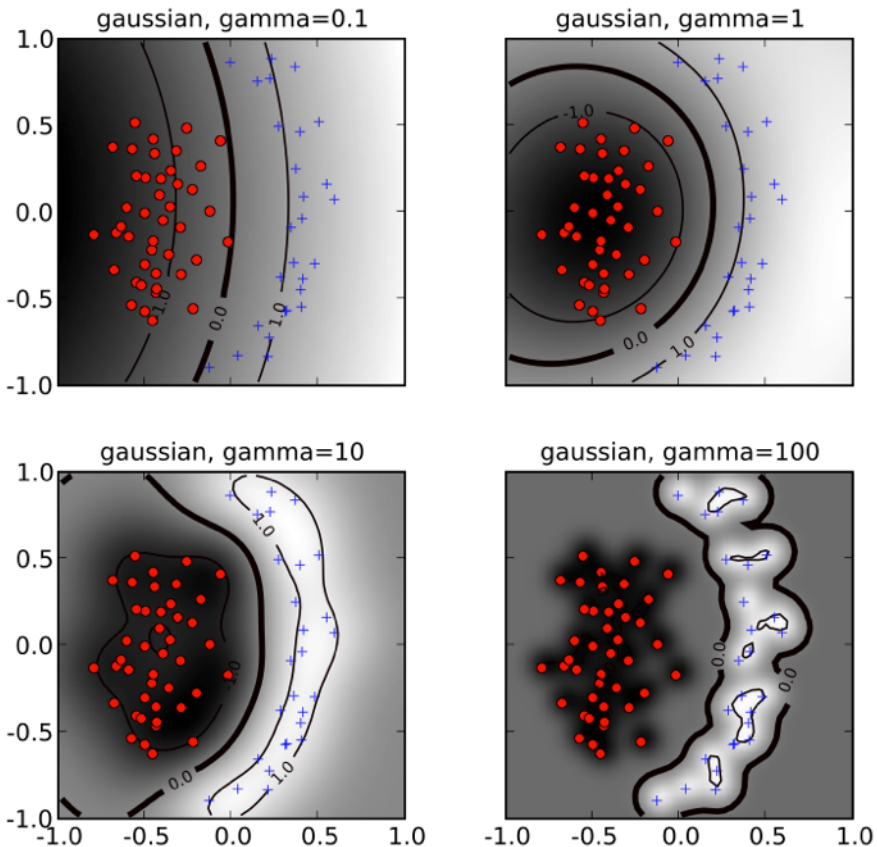
The **hyperparameters** d, γ *affect the flexibility of the decision boundary.*

NONLINEAR CLASSIFICATION – POLYNOMIAL KERNEL



source: http://pyml.sourceforge.net/doc/howto.pdf

NONLINEAR CLASSIFICATION – GAUSSIAN KERNEL



source: http://pyml.sourceforge.net/doc/howto.pdf

SVM STRENGTHS & WEAKNESSES

SVMs (and **kernel methods** *in general) are versatile, powerful, and popular techniques that can produce accurate results for a wide array of classification problems.*

The main disadvantage of SVMs is the lack of intuition they produce. These models are truly black boxes!

Quick exercise:

discuss with the person next to you what kernels are and how they can be used

LAB SVM