GENERAL ASSEMBLY

INTRO TO DATA SCIENCE

SUPPORT VECTOR MACHINES

I. SUPPORT VECTOR MACHINES
II. MAXIMUM MARGIN HYPERPLANES
III. SLACK VARIABLES
IV. NONLINEAR CLASSIFICATION

EXERCISE:
V. SVM IN SCIKIT-LEARN

I. SUPPORT VECTOR MACHINES

Q: What is a support vector machine?

Q: What is a support vector machine?

A: A binary linear classifier whose decision boundary is *explicitly* constructed to minimize generalization error.

Q: What is a support vector machine?

A: A binary linear classifier whose decision boundary is *explicitly* constructed to minimize generalization error.

recall:

binary classifier – solves two-class problem

linear classifier – creates linear decision boundary (in 2d)

Q: How is the decision boundary derived?

Q: How is the decision boundary derived?

A: Using *geometric reasoning* (as opposed to the algebraic reasoning we've used to derive other classifiers).

Q: How is the decision boundary derived?

A: Using *geometric reasoning* (as opposed to the algebraic reasoning we've used to derive other classifiers).

NOTE

These are two different ways of looking at the same problem.

Familiarity with both leads to deeper understanding!

Q: How is the decision boundary derived?
A: Using *geometric reasoning* (as opposed to the algebraic reasoning we've used to derive other classifiers).

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

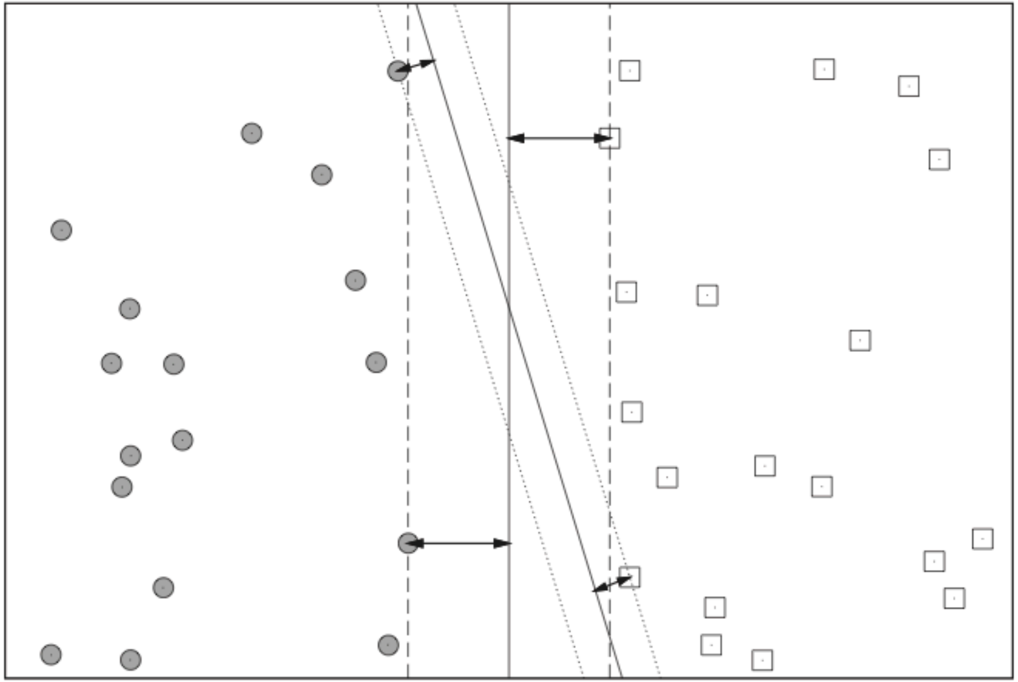


FIGURE 18-4. Two decision boundaries and their margins. Note that the vertical decision boundary has a wider margin than the other one. The arrows indicate the distance between the respective support vectors and the decision boundary.

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

Q: How is the decision boundary derived?
A: Using *geometric reasoning* (as opposed to the algebraic reasoning we've used to derive other classifiers).

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

Q: How is the decision boundary derived?

A: Using *geometric reasoning* (as opposed to the algebraic we've used to derive other classifiers).

NOTE

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

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

Q: If SVM is a linear classifier, how can you use it for nonlinear classification?

Q: If SVM is a linear classifier, how can you use it for nonlinear classification?

A: Using a clever maneuver called the **kernel trick**.

Nonlinear applications of SVM rely on an implicit (nonlinear) mapping Φ that sends vectors from the original feature space K into a higher-dimensional feature space K'.

Nonlinear applications of SVM rely on an implicit (nonlinear) mapping Φ that sends vectors from the original feature space K into a higher-dimensional feature space K'.

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

Nonlinear applications of SVM rely on an implicit (nonlinear) mapping Φ that sends vectors from the original feature space K into a higher-dimensional feature space K'.

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

In practice, this involves no computations in the higher dimensional space!

II. MAXIMUM MARGIN HYPERPLANES

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

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

A: By the **discriminant function**,

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

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

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

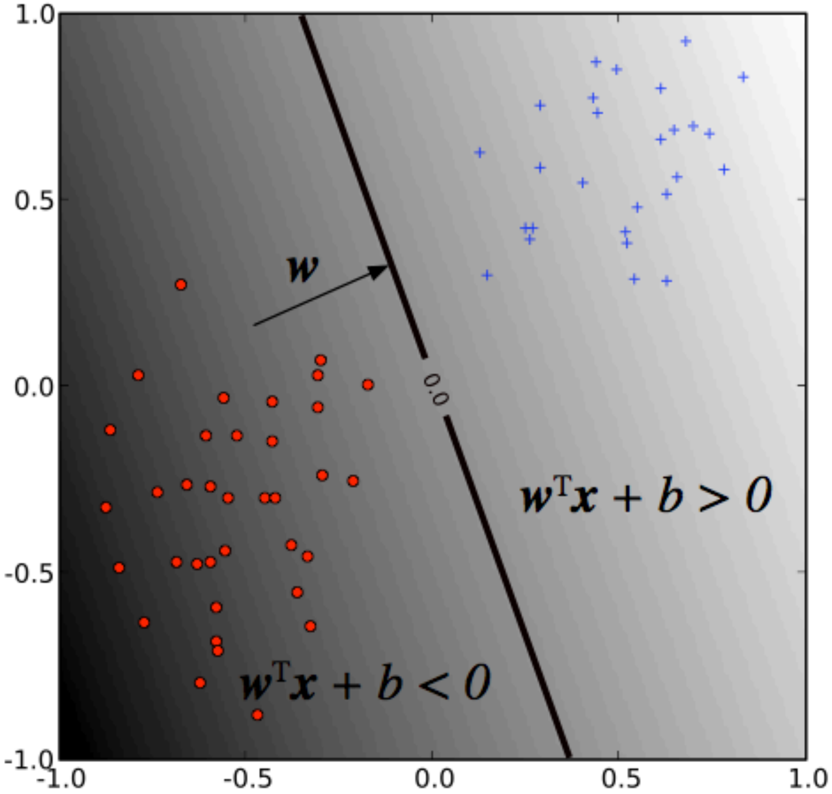
A: By the **discriminant function**,

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

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.

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?

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.

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

NOTE

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

Q: Why are these the same thing?

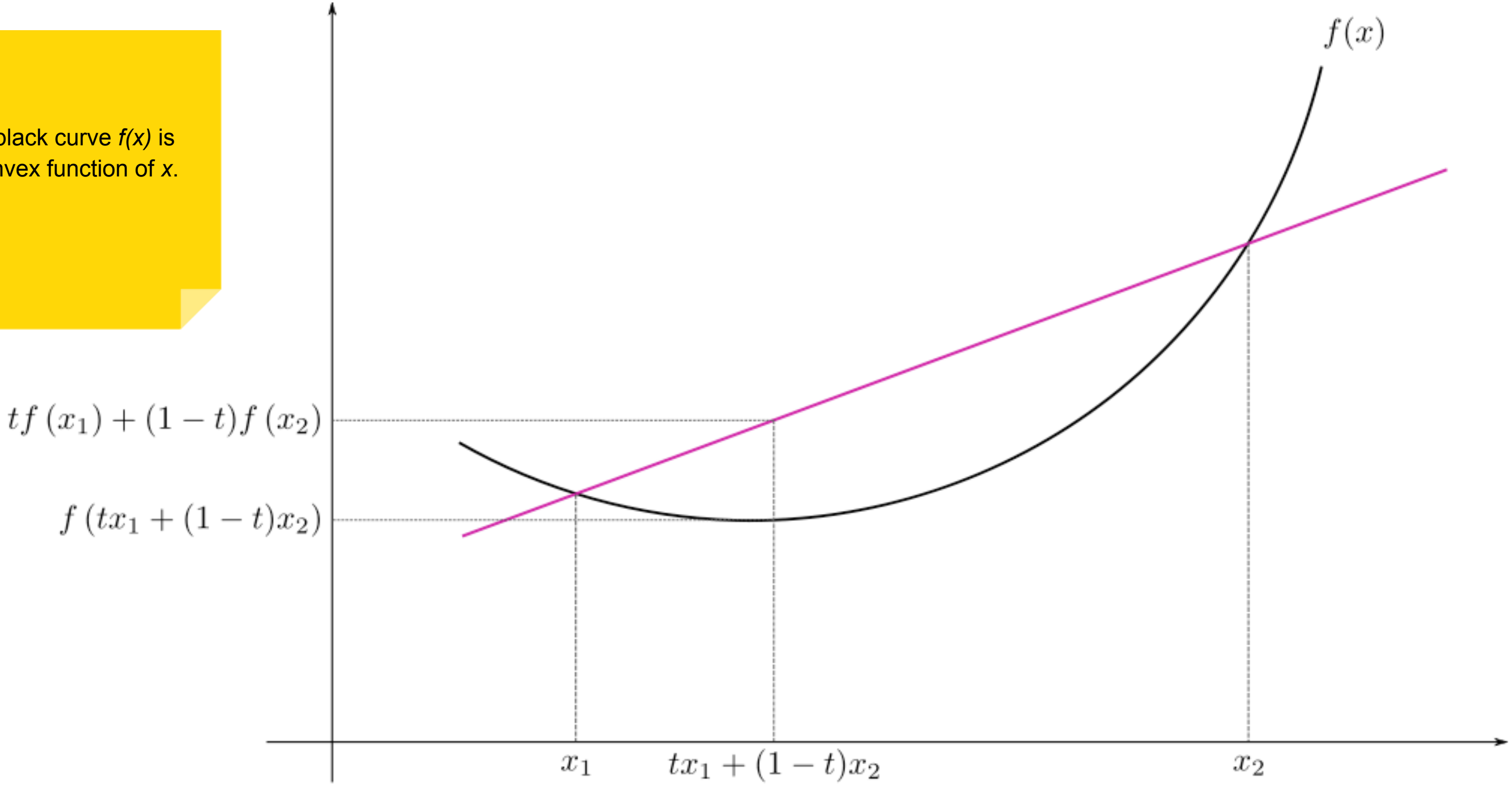
A: Because using the mmh as the decision boundary minimizes the probability that a small perturbation in the position of a point produces a classification error.

Selecting the mmh is an exercise in analytic geometry.

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

NOTE

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



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

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.

This is nice because convex optimization problems are guaranteed to give **global optima** (and they're easy to solve numerically too).

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 **con** objective function.

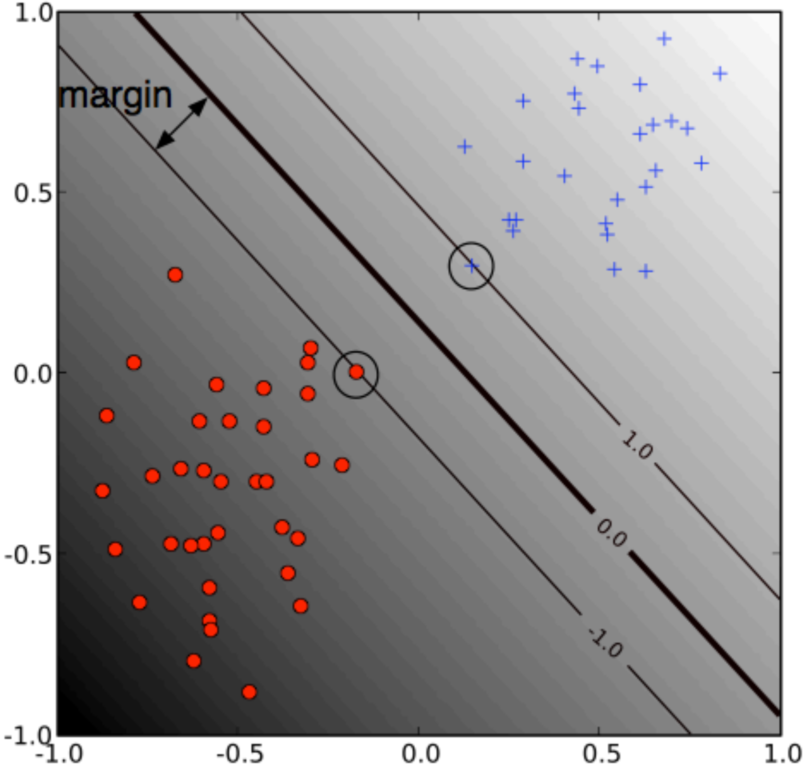
NOTE

The heuristic techniques we've discussed (eg greedy algorithms) are not necessary with convex optimization!

This is nice because convex optimization problems are guaranteed to give **global optima** (and they're easy to solve numerically too).

Notice that the margin depends only on a *subset* of the training data; namely, those points that are nearest to the decision boundary.

MAXIMUM MARGIN HYPERPLANES



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

Notice that the margin depends only on a *subset* of the training data; namely, those points that are nearest to the decision boundary.

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

Notice that the margin depends only on a *subset* of the training data; namely, those points that are nearest to the decision boundary.

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

The other points (far from the decision boundary) don't affect the construction of the mmh at all!

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

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

The optimization problem that this SVM solves is:

$$\underset{\mathbf{w},b}{\text{minimize}} \quad \frac{1}{2}||\mathbf{w}||^2$$
$$\text{subject to:} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \quad i = 1, \ldots, n\,.$$

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

The optimization problem that this SVM solves is:

$$\underset{\mathbf{w},b}{\text{minimize}} \quad \frac{1}{2}||\mathbf{w}||^2$$
$$\text{subject to:} \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \quad i = 1, \ldots, n\,.$$

NOTE

This type of optimization problem can be solved with *quadratic programming*.

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

III. SLACK VARABLES

Recall that in building the hard margin classifier, we assumed that our data was **linearly separable** (eg, that we could perfectly classify each record with a linear decision boundary).

Recall that in building the hard margin classifier, we assumed that our data was **linearly separable** (eg, that we could perfectly classify each record with a linear decision boundary).

Suppose that this was not true, or suppose that we wanted to use a larger margin at the expense of incurring some training error.

Recall that in building the hard margin classifier, we assumed that our data was **linearly separable** (eg, that we could perfectly classify each record with a linear decision boundary).

Suppose that this was not true, or suppose that we wanted to use a larger margin at the expense of incurring some training error.

This can be done using by introducing **slack variables.**

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

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:

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

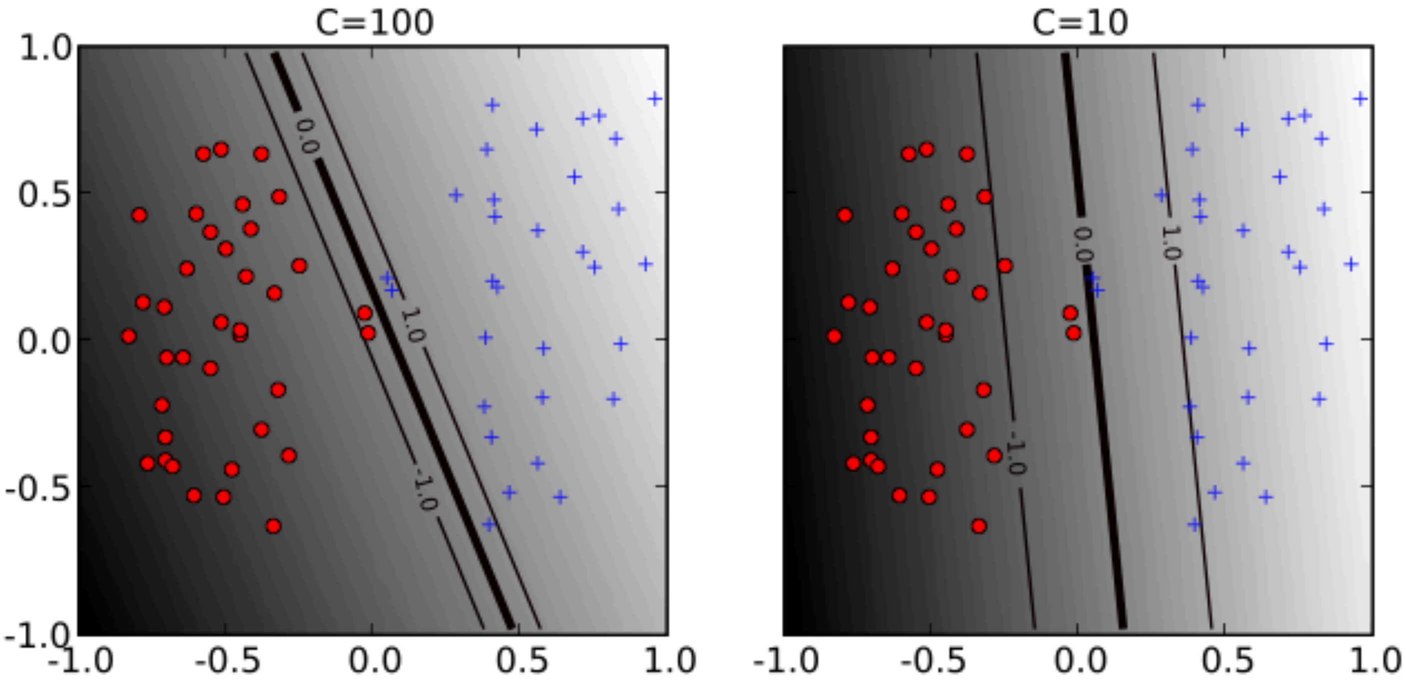
Slack variables ξ_i generalize the optimization problem to permit some misclassified training records (which come at a cost C).

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

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

This an example of *bias-variance tradeoff*.

SLACK VARIABLES – SOFT MARGIN CONSTANT



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

The soft-margin optimization problem can be rewritten as:

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

The soft-margin optimization problem can be rewritten as:

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

NOTE

This is called the *dual formulation* of the optimization problem.

(reached via Lagrange multipliers)

The soft-margin optimization problem can be rewritten as:

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

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

The fact that we 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.

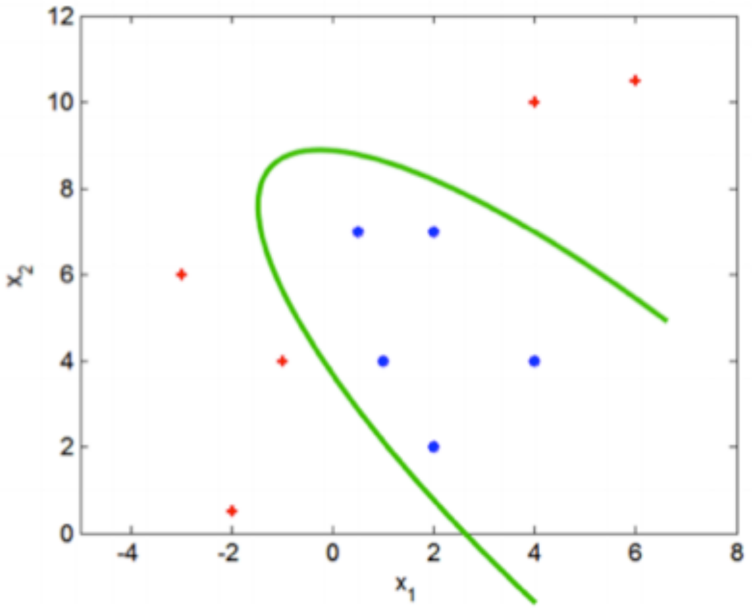
The inner product is an operation that takes two vectors and returns a real number.

The fact that we 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.

In particular, we can easily change K to be some other space K'.

IV. NONLINEAR CLASSIFICATION

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



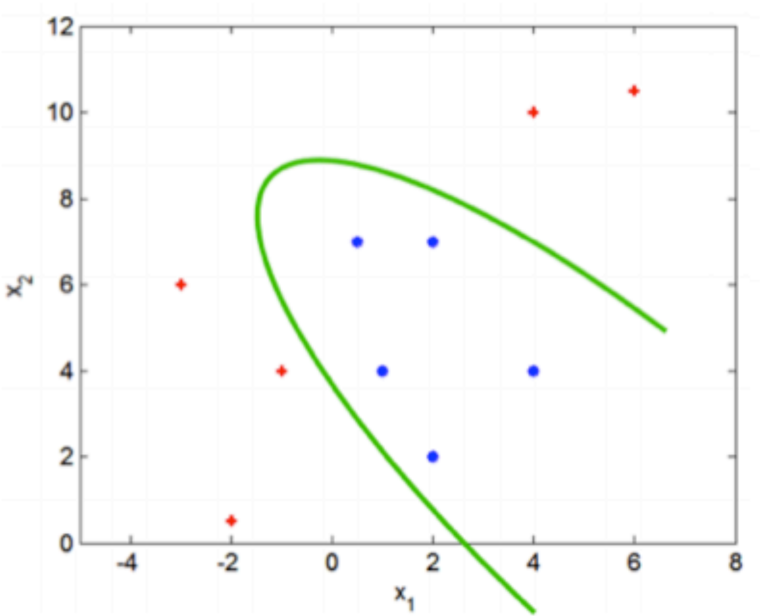
Suppose we need a more complex classifier than a linear decision boundary allows.

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.

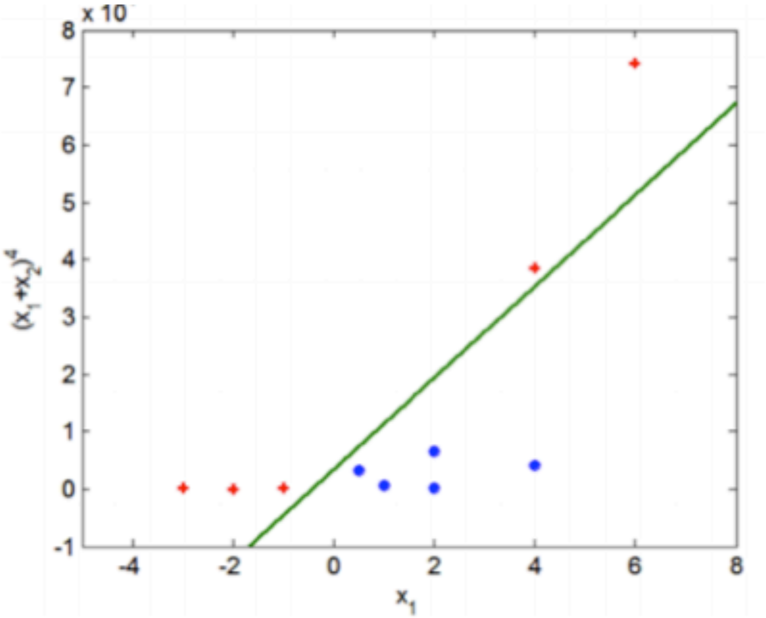
Suppose we need a more complex classifier than a linear decision boundary allows.

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.



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 $x^T x$:

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

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

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

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 $\mathbb{R}$.

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 $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}' + 1)^d$

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

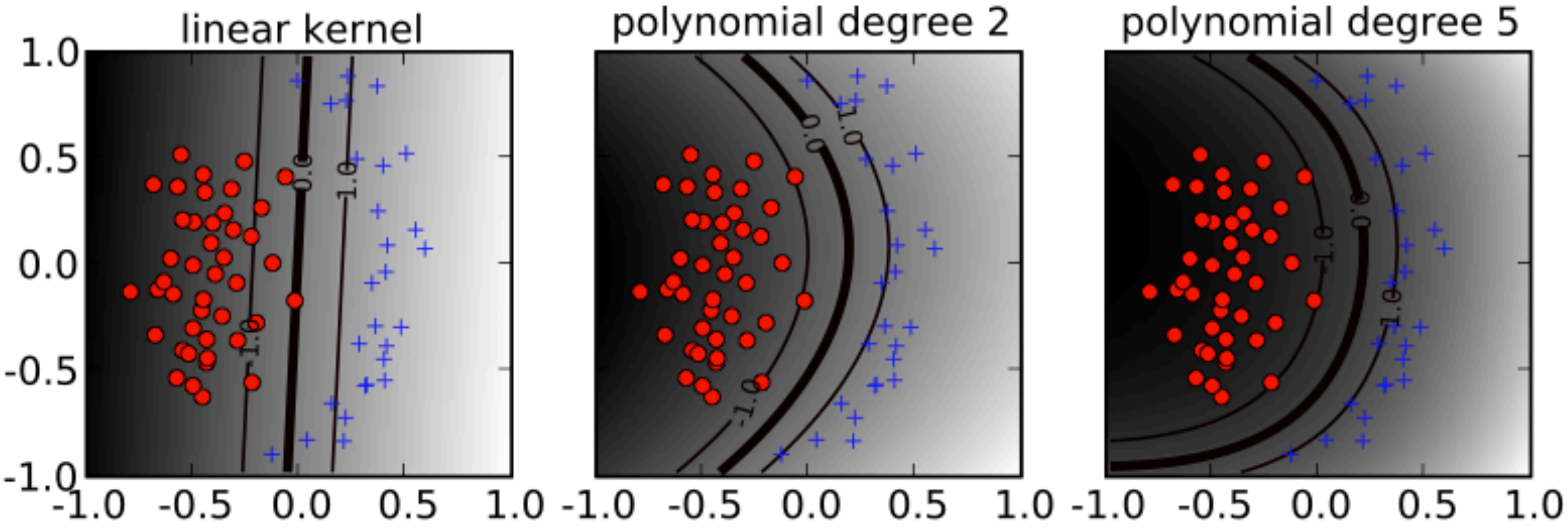
some popular kernels:

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

polynomial kernel $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}' + 1)^d$

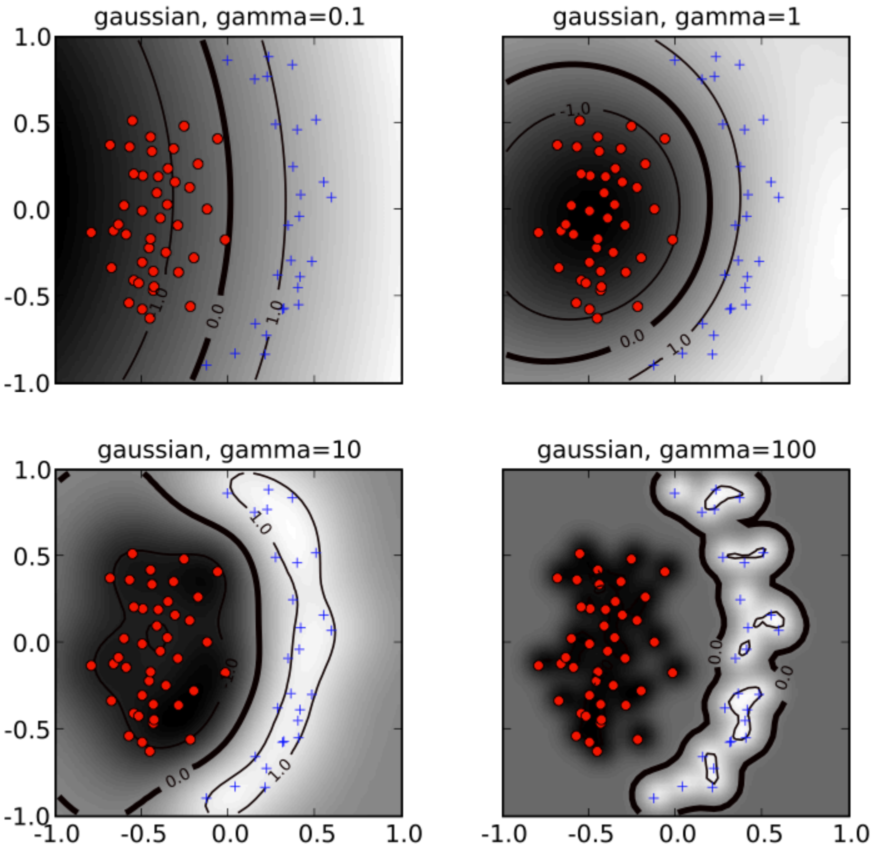
Gaussian kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma ||\mathbf{x} - \mathbf{x}'||^2)$

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



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

NONLINEAR CLASSIFICATION – GAUSSIAN KERNEL



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

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!

EX: SVM IN SCIKIT-LEARN