Support Vector Machines

INFO-4604, Applied Machine Learning

University of Colorado Boulder

September 28, 2017

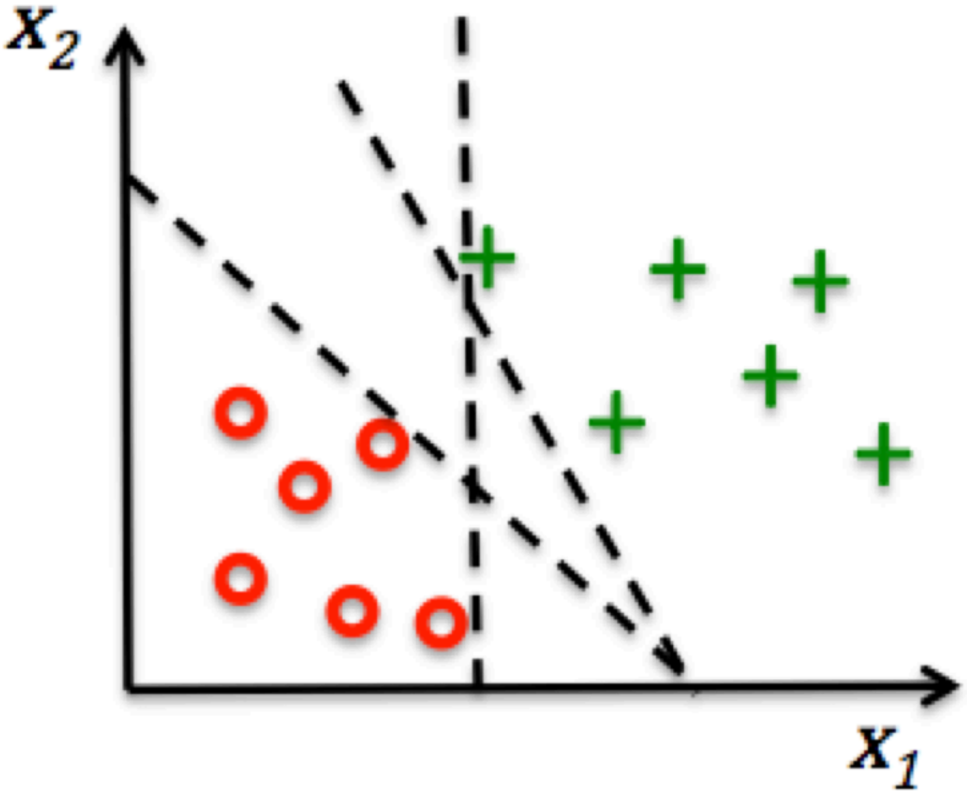
Prof. Michael Paul

Today

Two important concepts:

- Margins
- Kernels

Large Margin Classification



Which hyperplane?

Linear Predictions

Perceptron:

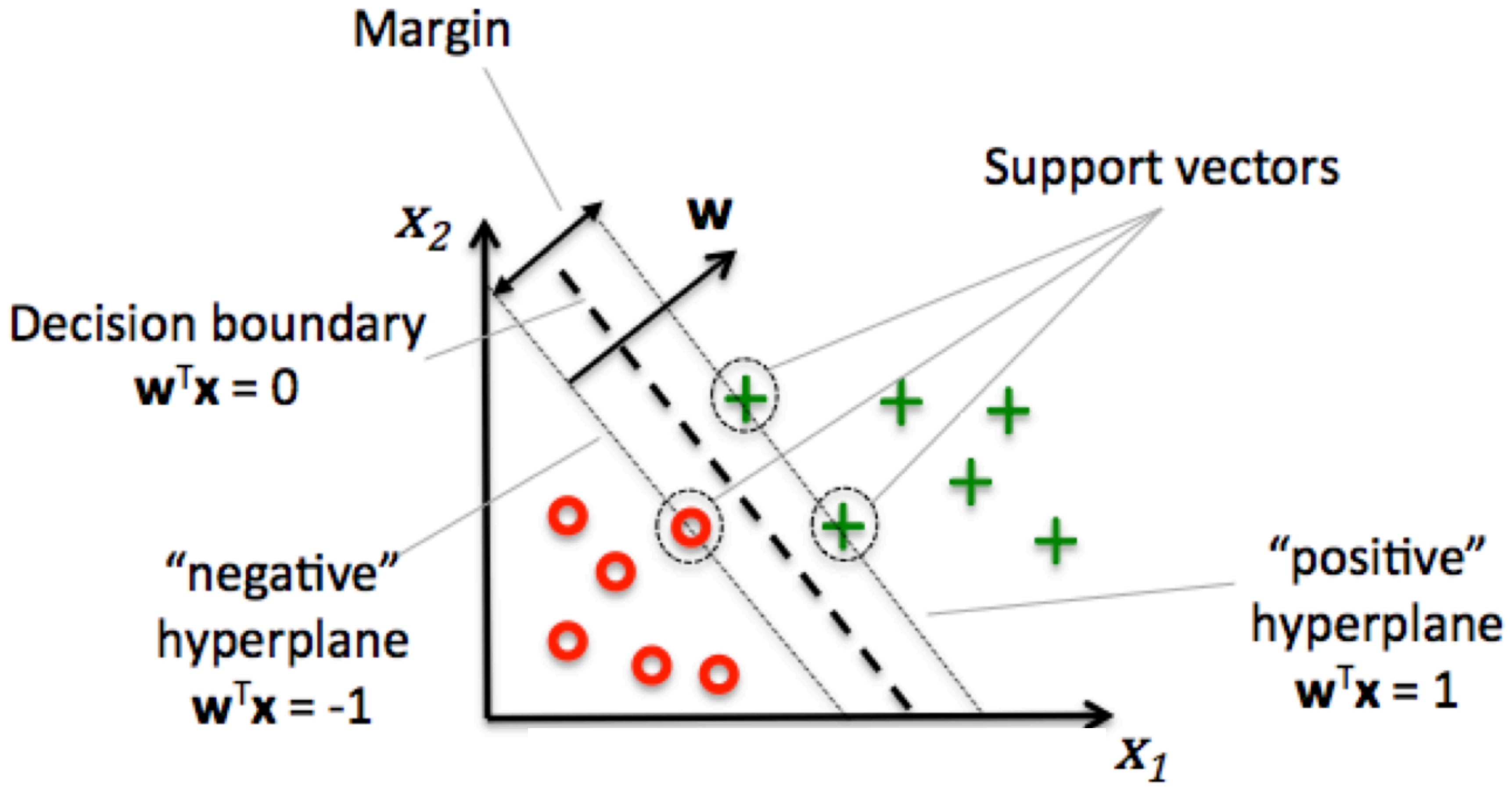
$$f(\mathbf{x}) = \begin{cases} 1, & \mathbf{w}^T\mathbf{x} \geq 0 \\ -1, & \mathbf{w}^T\mathbf{x} < 0 \end{cases}$$

SVM:

$$f(\mathbf{x}) = \begin{cases} 1, & \mathbf{w}^T\mathbf{x} \geq 1 \\ -1, & \mathbf{w}^T\mathbf{x} \leq -1 \end{cases}$$

Two different boundaries for positive vs negative

Large Margin Classification



Large Margin Classification

The **margin** is the distance between the two boundaries.

The **support vectors** are the instances at the boundaries (when $\mathbf{w}^T\mathbf{x} = 1$ or -1)

- Or within the boundaries, if not linearly separable

The goal of SVMs is to learn the boundaries to make the margin as large as possible (while still correctly classifying the instances)

- *maximum margin* classification

Large Margin Classification

The size of the margin is: 2 / ll**w**ll

- Recall: ll**w**ll is the *L2 norm* of the weight vector
- Smaller weights → larger margin

Learning goal:

- Maximize 2 / ll**w**ll, subject to the constraints that all instances are correctly classified
- Turn it into minimization problem by taking the inverse: ½ ll**w**ll
- Can also square the L2 norm (makes the calculus easier), just like with L2 regularization: ½ ll**w**ll^2

Large Margin Classification

The size of the margin is: 2 / ll**w**ll

- Recall: ll**w**ll is the *L2 norm* of the weight vector
- Smaller weights → larger margin

Learning goal:

- Minimize: $\frac{1}{2}\|\boldsymbol{w}\|^2$
- Subject to the constraints:

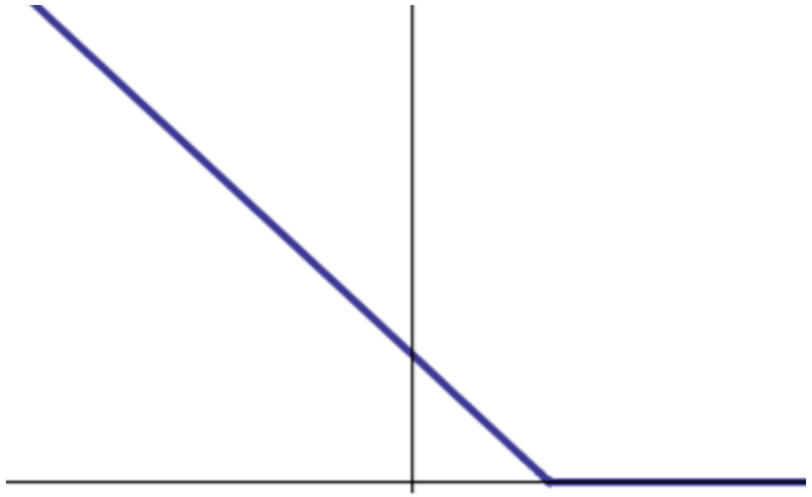
$$y^{(i)}\left(w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right) \geq 1 \ \forall_i$$

Only possible to satisfy these constraints if the instances are linearly separable!

Large Margin Classification

In the general case, SVM uses this loss function:

$$L_i(\mathbf{w}; \mathbf{x_i}) = \begin{cases} 0, & y_i\,(\mathbf{w}^T\mathbf{x_i}) \geq 1 \\ -\,y_i\,(\mathbf{w}^T\mathbf{x_i}), & \text{otherwise} \end{cases}$$



Same as perceptron, but $y_i\,(\mathbf{w}^T\mathbf{x_i}) \geq 1$ instead of $y_i\,(\mathbf{w}^T\mathbf{x_i}) \geq 0$

Large Margin Classification

In the general case, SVM uses this loss function:

$$L_i(\mathbf{w}; \mathbf{x_i}) = \begin{cases} 0, & y_i\,(\mathbf{w}^T\mathbf{x_i}) \geq 1 \\ -\,y_i\,(\mathbf{w}^T\mathbf{x_i}), & \text{otherwise} \end{cases}$$

The learning goal of SVMs when the data are not linearly separable is to minimize:

$$\underbrace{\tfrac{1}{2}\,||\mathbf{w}||^2}_{\text{inverse margin}} + C\,\underbrace{L(\mathbf{w})}_{\text{training loss}}$$

SVMs also use L2 regularization
- *C* is like λ from before, but larger *C* → lower loss

Large Margin Classification

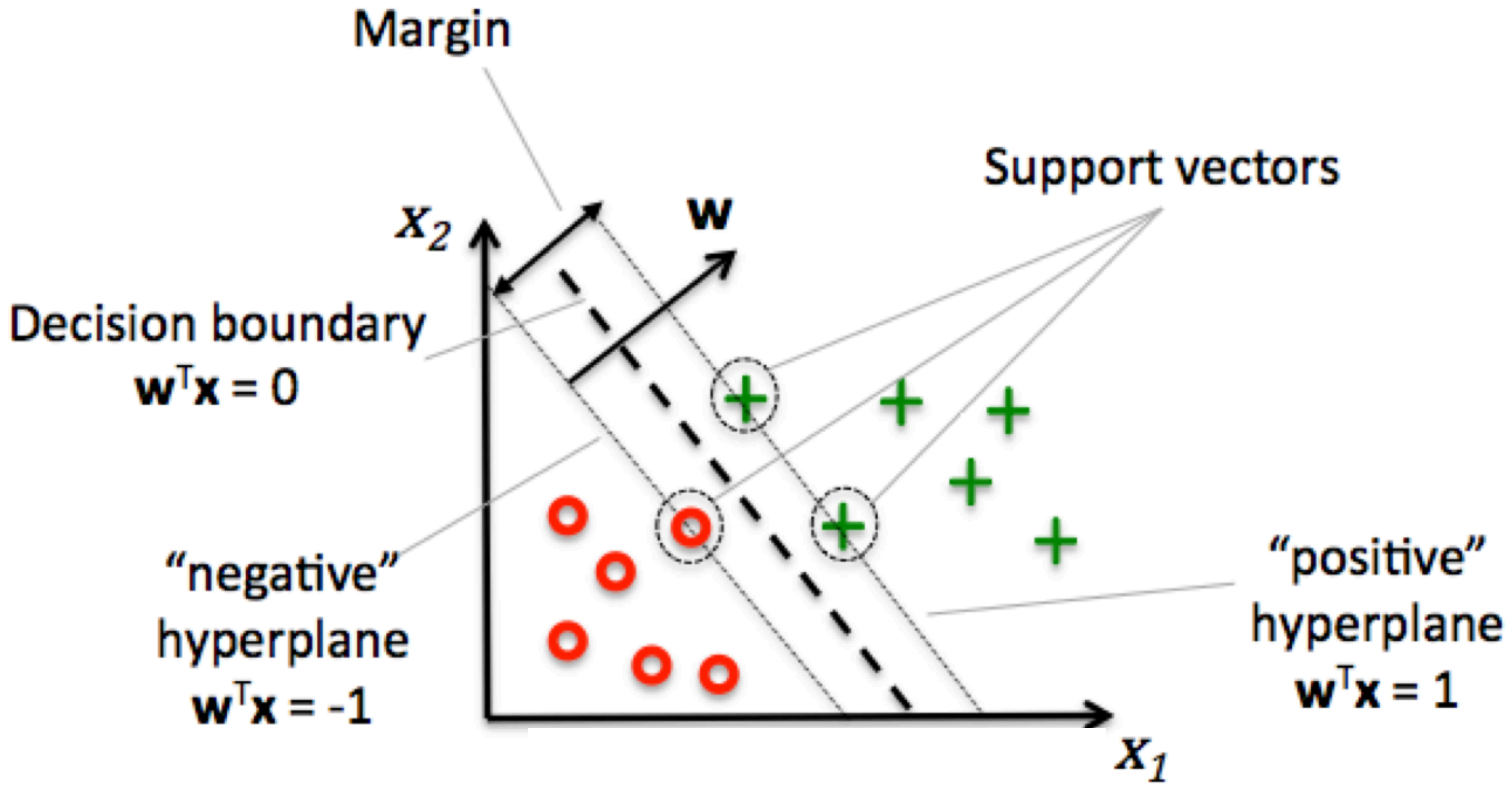
Like perceptron, the SVM function can be minimized using stochastic (sub)gradient descent

- With sklearn’s `SGDClassifier` class, SVM can be implemented by setting `loss='hinge’`

Other implementations (usually using different optimization algorithms than SGD)

- Liblinear and LIBSVM (both used by sklearn)
- SVM-light

Large Margins: Summary



Large Margins: Summary

Classifiers with large margins are more likely to have better generalization, less overfitting

- Hyperparameter *C* controls the tradeoff between margin size and classification error

The large margin principle is another justification for L2 regularization that you saw earlier

- Since the size of the margin is inversely proportional to the L2 norm of the weight vector

Kernel Trick

It turns out that the optimal solution for **w** is equivalent to:

$\sum_i \alpha_i \mathbf{x_i}$

Combination of each training instance's feature vector, weighted by α

So in the loss function and prediction functions, we can replace $\mathbf{w}^T\mathbf{x}$ with $\sum_i \alpha_i \mathbf{x_i}^T\mathbf{x}$

α_i is only nonzero for support vectors

- This summation can therefore skip over all other instances, making this calculation more efficient.

Kernel Trick

In the loss function and prediction functions, we can replace $\mathbf{w}^T\mathbf{x}$ with $\sum_i \alpha_i \mathbf{x_i}^T\mathbf{x}$

Now this looks similar to weighted nearest neighbor classification, where the “similarity” between an instance $\mathbf{x}$ and another instance $\mathbf{x_i}$ is $\mathbf{x_i}^T\mathbf{x}$ and this is additionally weighted by α_i

Learning goal is now to learn $\boldsymbol{\alpha}$ instead of $\mathbf{w}$

- How? More complex than before…

Kernel Functions

Loosely, a kernel function is a similarity function between two instances

General kernel trick:
Replace $\mathbf{w}^T\mathbf{x}$ with $\sum_i \alpha_i \, k(\mathbf{x}_i, \mathbf{x})$

The **linear kernel** function for an SVM is:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$$

Kernel Functions

What happens if we define the kernel function in some other way?

Then it won't be true that $\sum_i \alpha_i k(\mathbf{x_i}, \mathbf{x}) = \mathbf{w}^T\mathbf{x}$

But: kernels can be defined so that,

$$\sum_i \alpha_i k(\mathbf{x_i}, \mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}),$$

where $\phi(\mathbf{x})$ is some other feature representation.

Kernel Functions: Polynomial

A **polynomial kernel** function is defined as:

$$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i}^T\mathbf{x_j} + c)^d$$

If d=2 (quadratic kernel), then it turns out that

$$\sum_i \alpha_i k(\mathbf{x_i}, \mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

where

$$\varphi(x) = \langle x_n^2, \ldots, x_1^2, \sqrt{2}x_n x_{n-1}, \ldots, \sqrt{2}x_n x_1, \sqrt{2}x_{n-1}x_{n-2}, \ldots, \sqrt{2}x_{n-1}x_1, \ldots, \sqrt{2}x_2 x_1, \sqrt{2c}x_n, \ldots, \sqrt{2c}x_1, c\rangle$$

Kernel Functions: Polynomial

In other words, using a quadratic kernel is equivalent to using a standard SVM where you've expanded the feature vectors to include:

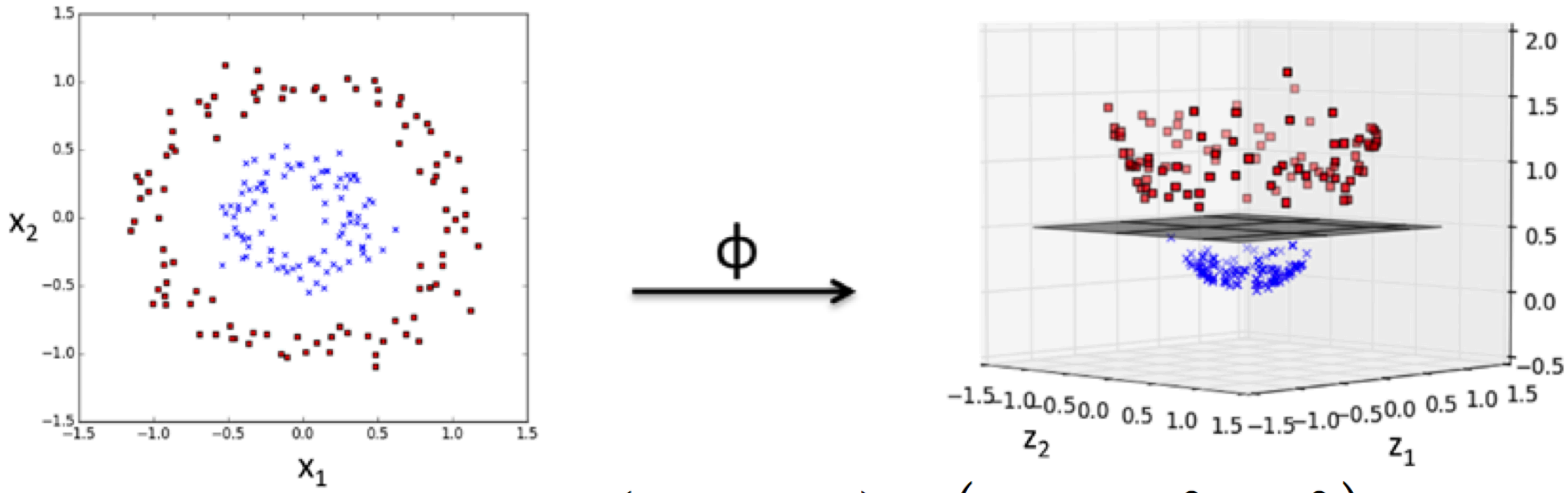
- Each original feature value (times a constant)
- Each feature value squared
- The product of each pair of feature values (times a constant)
    - This can be especially useful, since it can capture *interactions* between features

Without the kernel trick, this large feature set would be computationally expensive to work with.

Kernel Functions

In general, the kernel trick can create new features as nonlinear combinations of the old features

- Data that are not linearly separable in the original feature space might be separable in the new space



$$\phi(x_1, x_2) = (z_1, z_2, z_3) = \left(x_1, x_2, x_1^2 + x_2^2\right)$$

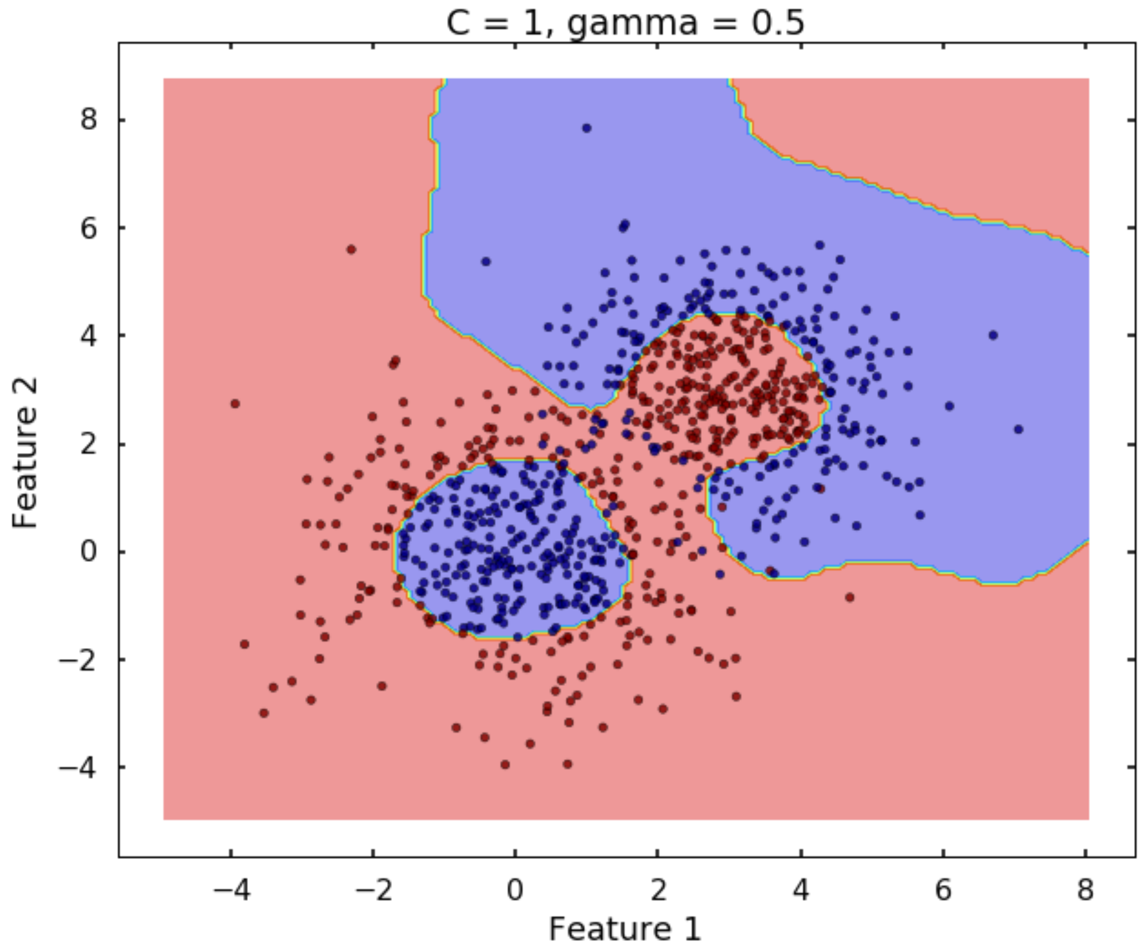
Kernel Functions: RBF

The **radial basis function** (RBF kernel) is:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \underbrace{||\mathbf{x}_i - \mathbf{x}_j||^2}_{\text{squared Euclidean distance}})$$

- One of the most popular SVM kernels
- Related to the Gaussian/normal distribution
- Interpretation as expanded feature vector?
  - It actually maps to a feature vector with infinitely many features… so technically equivalent, but impossible to implement without using the kernel trick.

Kernel Functions: RBF



From: http://qingkaikong.blogspot.com/2016/12/machine-learning-8-support-vector.html

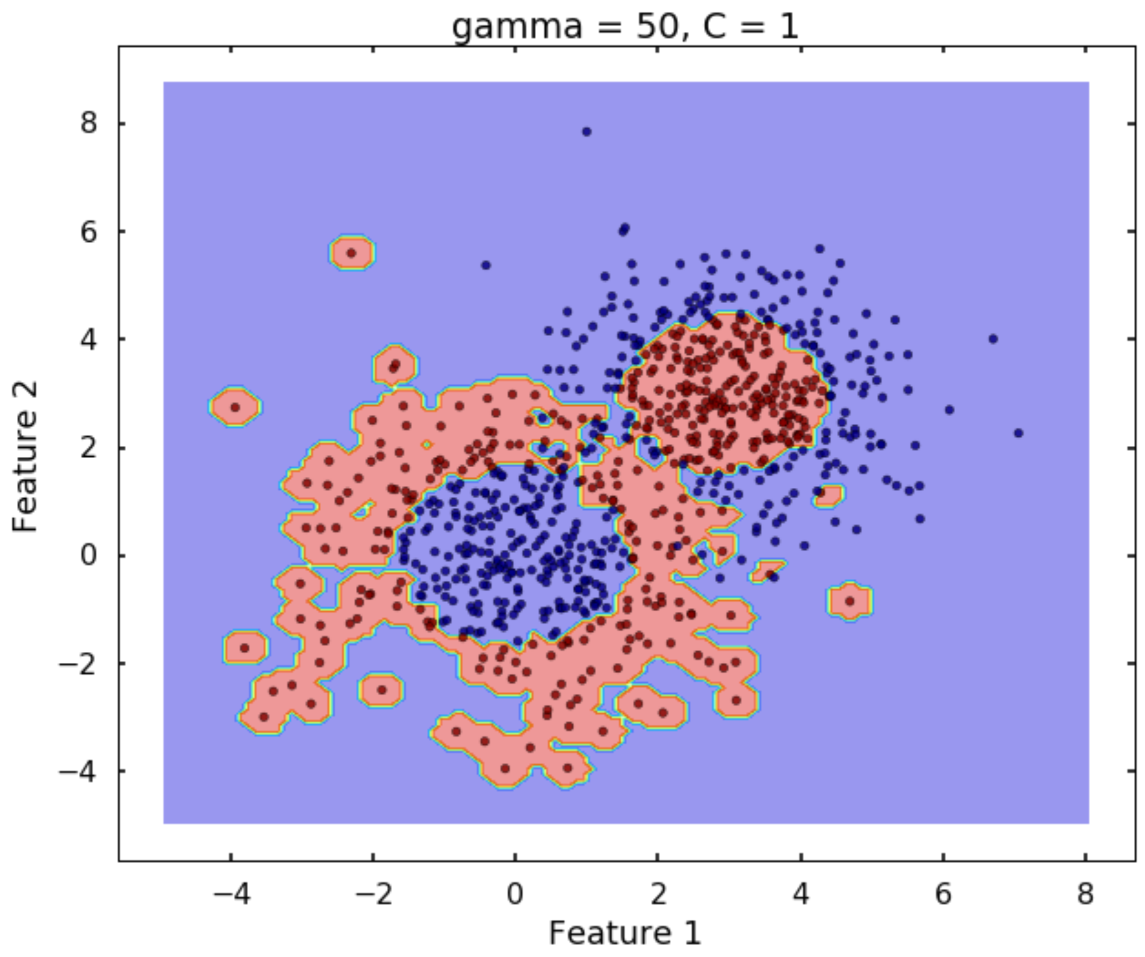
Kernel Functions: RBF

The **radial basis function** (RBF kernel) is:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \, ||\mathbf{x}_i - \mathbf{x}_j||^2)$$

- In addition to *C*, γ also affects overfitting
- Large $\gamma \rightarrow$ small differences in distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ are magnified
  - This will cause the classifier to fit the training data better, but may do worse on future data

Kernel Functions: RBF



From: http://qingkaikong.blogspot.com/2016/12/machine-learning-8-support-vector.html

Kernel Methods: Summary (1)

- Kernel SVM is a reformulation of SVM that uses similarity between instances
    - To make a prediction for a new instance, need to calculate kernel function for the new instance and all training instances that are the support vectors
- Kernel SVM is equivalent to an SVM with a expanded feature set
    - Sometimes there is an intuitive interpretation of what the “new” features mean; sometimes not
- Kernel SVM with a linear kernel is equivalent to a standard SVM

Kernel Methods: Summary (2)

- Kernels can be useful when your data has a small number of features and/or when the dataset is not linearly separable
- Some kernels are prone to overfitting
  - High degree polynomial; RBF with high scaling parameter
- Kernel SVM has additional hyperparameters you have to choose
  - Type of kernel
  - Parameters of kernel (e.g., d in polynomial, γ in RBF)

Kernel Methods: Summary (3)

Also be aware that:

- Kernel methods not unique to SVM (invented long before, for perceptron), but popularized by it.
- Lots of other kernel functions not shown here, but these are the most common.
    - Specialized kernels exist for certain types of data (e.g., biological sequences, syntax trees)