Last Updated: November 22, 2012

KERNEL METHODS

J. Elder

CSE 4404/5327 Introduction to Machine Learning and Pattern Recognition

Kernel Methods: Outline

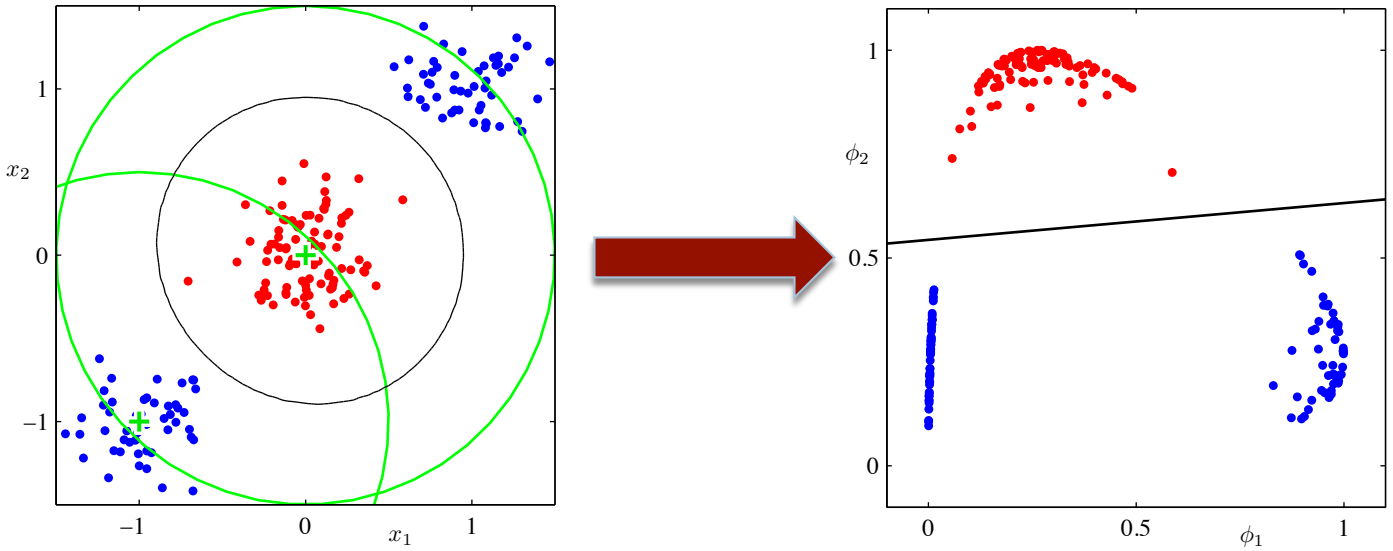
- Generalized Linear Models
- Radial Basis Function Networks
- Support Vector Machines
  - Separable classes
  - Non-separable classes
- The Kernel Trick

Kernel Methods: Outline

- **Generalized Linear Models**
- Radial Basis Function Networks
- Support Vector Machines
  - Separable classes
  - Non-separable classes
- The Kernel Trick

Generalizing Linear Classifiers

- One way of tackling problems that are not linearly separable is to transform the input in a nonlinear fashion prior to applying a linear classifier.
- The result is that decision boundaries that are linear in the resulting feature space may be highly nonlinear in the original input space.



Nonlinear Basis Function Models

- Generally

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})$$

- where $\phi_j(\mathbf{x})$ are known as *basis functions*.
- Typically, $\Phi_0(\mathbf{x}) = 1$, so that w_0 acts as a bias.

Nonlinear basis functions for classification

- In the context of classification, the discriminant function in the feature space becomes:

$$g\left(y(x)\right) = w_0 + \sum_{i=1}^{M} w_i y_i(x) = w_0 + \sum_{i=1}^{M} w_i \phi_i(x)$$

- This formulation can be thought of as an input space approximation of the true separating discriminant function g(x) using a set of interpolation functions $\phi_i(x)$.

Dimensionality

- The dimensionality M of the feature space may be less than, equal to, or greater than the dimensionality D of the original input space.
  - M < D: This may result in a factoring out of irrelevant dimensions, reduction in the number of model parameters, and resulting improvement in generalization (reduced overlearning).
  - M > D: Problems that are not linearly separable in the input space may become separable in the feature space, and the probability of linear separability generally increases with the dimensionality of the feature space. Thus choosing M >> D helps to make the problem linearly separable.

Cover's Theorem

- "A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated."

  — *Cover, T.M. , Geometrical and Statistical properties of systems of linear inequalities with applications in pattern recognition., 1965*

Example

Kernel Methods: Outline

- Generalized Linear Models
- **Radial Basis Function Networks**
- Support Vector Machines
  - Separable classes
  - Non-separable classes
- The Kernel Trick

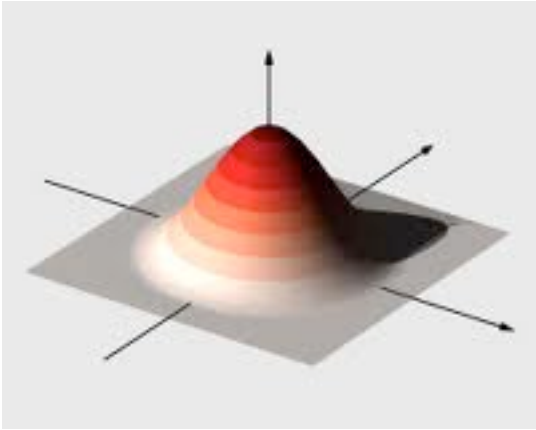
Radial Basis Functions

- Consider interpolation functions (kernels) of the form

$$\phi_i\left(\left\|x-\mu_i\right\|\right)$$

- In other words, the feature value depends only upon the Euclidean distance to a 'centre point' in the input space.
- A commonly used RBF is the isotropic Gaussian:

$$\phi_i(x)=\exp\left(-\frac{1}{2\sigma_i^2}\left\|x-\mu_i\right\|^2\right)$$



Relation to KDE

- We can use Gaussian RBFs to approximate the discriminant function g(x):

$$g\left(y(x)\right) = w_0 + \sum_{i=1}^{M} w_i y_i(x) = w_0 + \sum_{i=1}^{M} w_i \phi_i(x)$$

- where

$$\phi_i(x) = \exp\left(-\frac{1}{2\sigma_i^2}\left\|x - \mu_i\right\|^2\right)$$

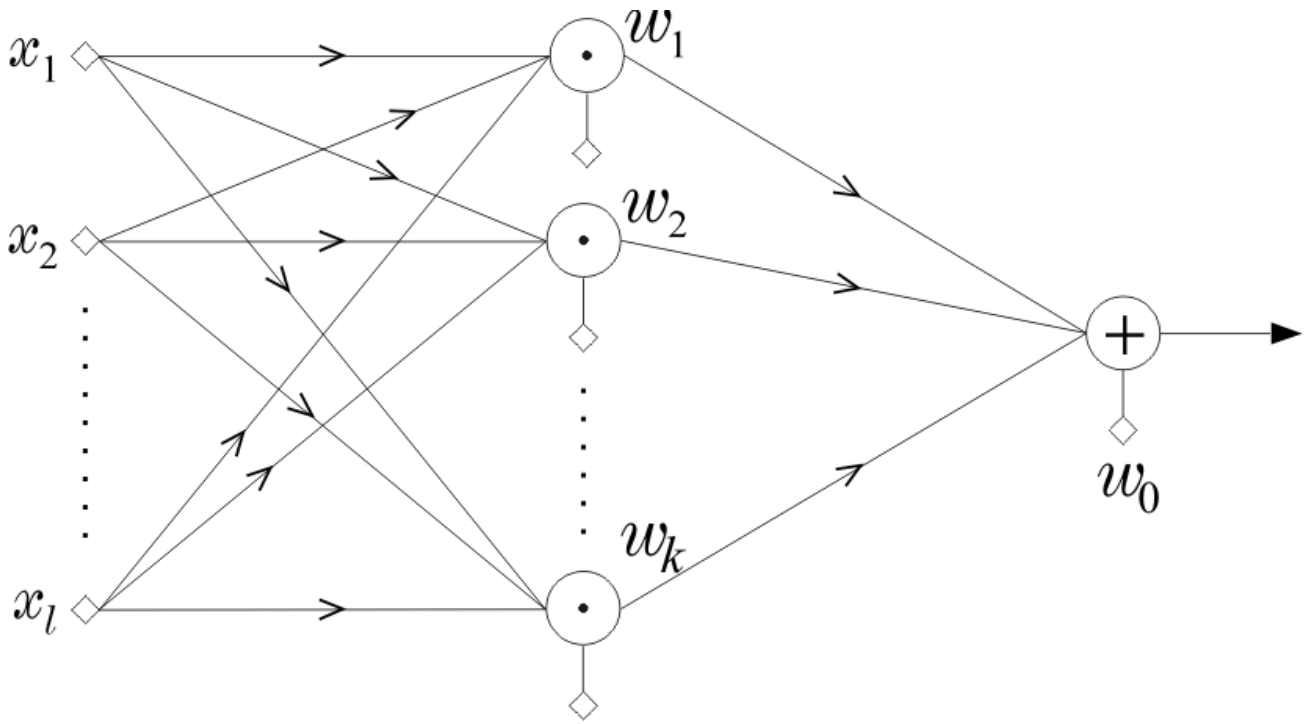
- This is reminiscent of kernel density estimation, where we approximated probability densities as a normalized sum of Gaussian kernels.

Relation to KDE

- For KDE we planted a kernel at each data point. Thus there were N kernels.
- For RBF networks, we generally use far fewer kernels than the number of data points: $M << N$.
- This leads to greater efficiency and generalization.

RBF Networks

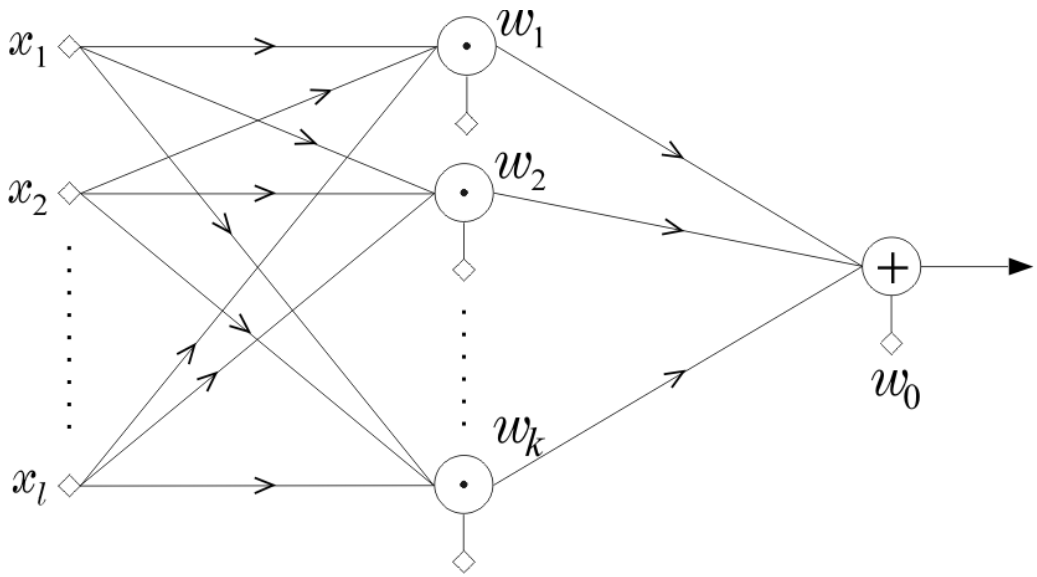
- The Linear Classifier with nonlinear radial basis functions can be considered an artificial neural network where
  - The hidden nodes are nonlinear (e.g., Gaussian).
  - The output node is linear.



RBF Network for 2 Classes

RBF Networks vs Perceptrons

- Recall that for a perceptron, the output of a hidden unit is invariant on a hyperplane.
- For an RBF, the output of a hidden unit is invariant on a circle centred on μ_i.
- Thus hidden units are global in a perceptron, but local in an RBF network.



RBF Network for 2 Classes

RBF Networks vs Perceptrons

- This difference has consequences:
  - Multilayer perceptrons tend to learn slower than RBFs.
  - However, multilayer perceptrons tend to have better generalization properties, especially in regions of the input space where training data are sparse.
  - Typically, more neurons are needed for an RBF than for a multilayer perceptron to solve a given problem.

Parameters

- There are two options for choosing the parameters (centres and scales) of the RBFs:
  1. **Fixed.**
     - For example, randomly select a subset of M of the input vectors and use these as centres. Use a common scale based upon your judgement.
  2. **Learned.**
     - Note that when the RBF parameters are fixed, the weights could be learned using linear classifier techniques (e.g., least squares).
     - Thus the RBF parameters could be learned in an outer loop, by gradient descent.

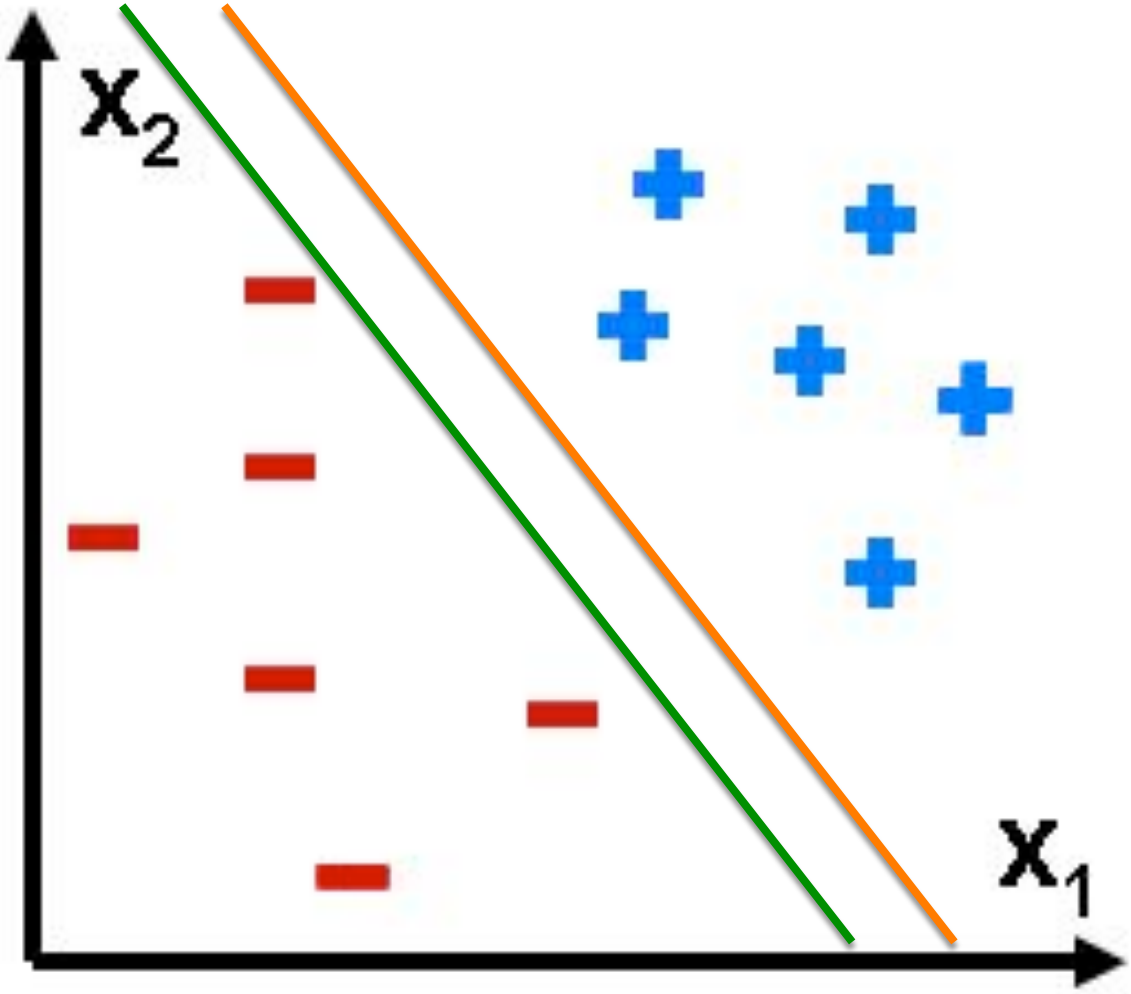
Kernel Methods: Outline

- Generalized Linear Models
- Radial Basis Function Networks
- **Support Vector Machines**
  - **Separable classes**
  - Non-separable classes
- The Kernel Trick

Motivation

- The perceptron algorithm is guaranteed to provide a linear decision surface that separates the training data, if one exists.
- However, if the data are linearly separable, there are in general an infinite number of solutions, and the solution returned by the perceptron algorithm depends in a complex way on the initial conditions, the learning rate and the order in which training data are processed.
- While all solutions achieve a perfect score on the training data, they won't all necessarily generalize as well to new inputs.

Which solution would you choose?



The Large Margin Classifier

- Unlike the Perceptron Algorithm, the Support Vector Machine solves a problem that has a unique solution: it returns the linear classifier with the maximum margin, that is, the hyperplane that separates the data and is farthest from any of the training vectors.
- Why is this good?

Support Vector Machines

SVMs are based on the linear model $y(\mathbf{x}) = \mathbf{w}^t \phi(\mathbf{x}) + b$

Assume training data $\mathbf{x}_1,\ldots,\mathbf{x}_N$ with corresponding target values $t_1,\ldots,t_N$, $t_n \in \{-1,1\}$.

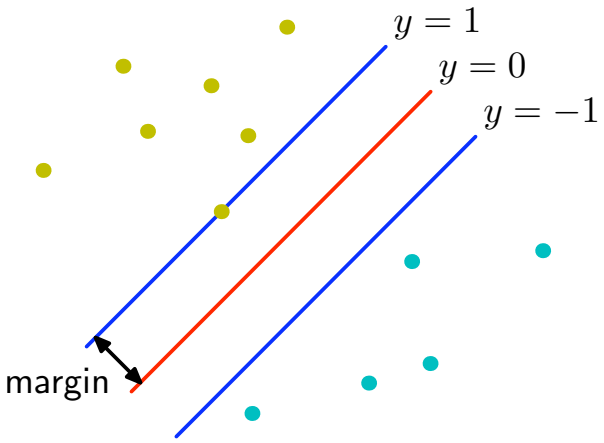
$\mathbf{x}$ classified according to sign of $y(\mathbf{x})$.

Assume for the moment that the training data are linearly separable in feature space.

Then $\exists \mathbf{w}, b : t_n y(\mathbf{x}_n) > 0 \;\; \forall n \in [1,\ldots N]$

Maximum Margin Classifiers

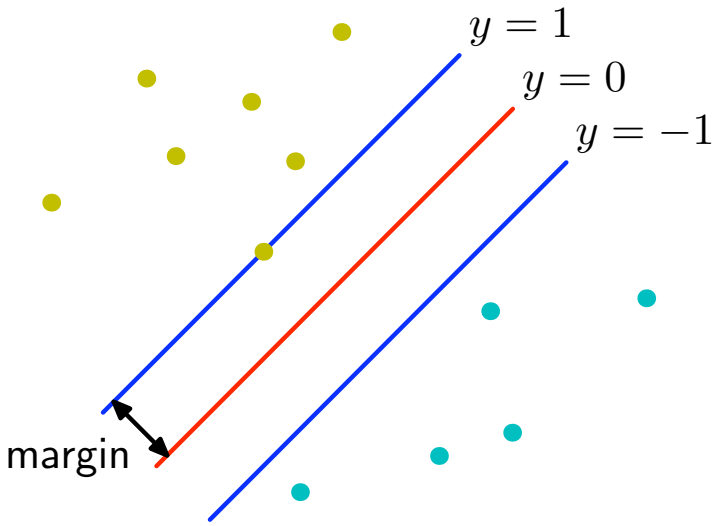
- When the training data are linearly separable, there are generally an infinite number of solutions for (**w**, *b*) that separate the classes exactly.
- The **margin** of such a classifier is defined as the orthogonal distance in feature space between the decision boundary and the closest training vector.
- SVMs are an example of a **maximum margin classifer**, which finds the linear classifier that maximizes the margin.



Probabilistic Motivation

- The maximum margin classifier has a probabilistic motivation.

If we model the class-conditional densities with a KDE using Gaussian kernels with variance σ^2, then in the limit as $\sigma \rightarrow 0$, the optimal linear decision boundary $\rightarrow$ maximum margin linear classifier.



Two Class Discriminant Function

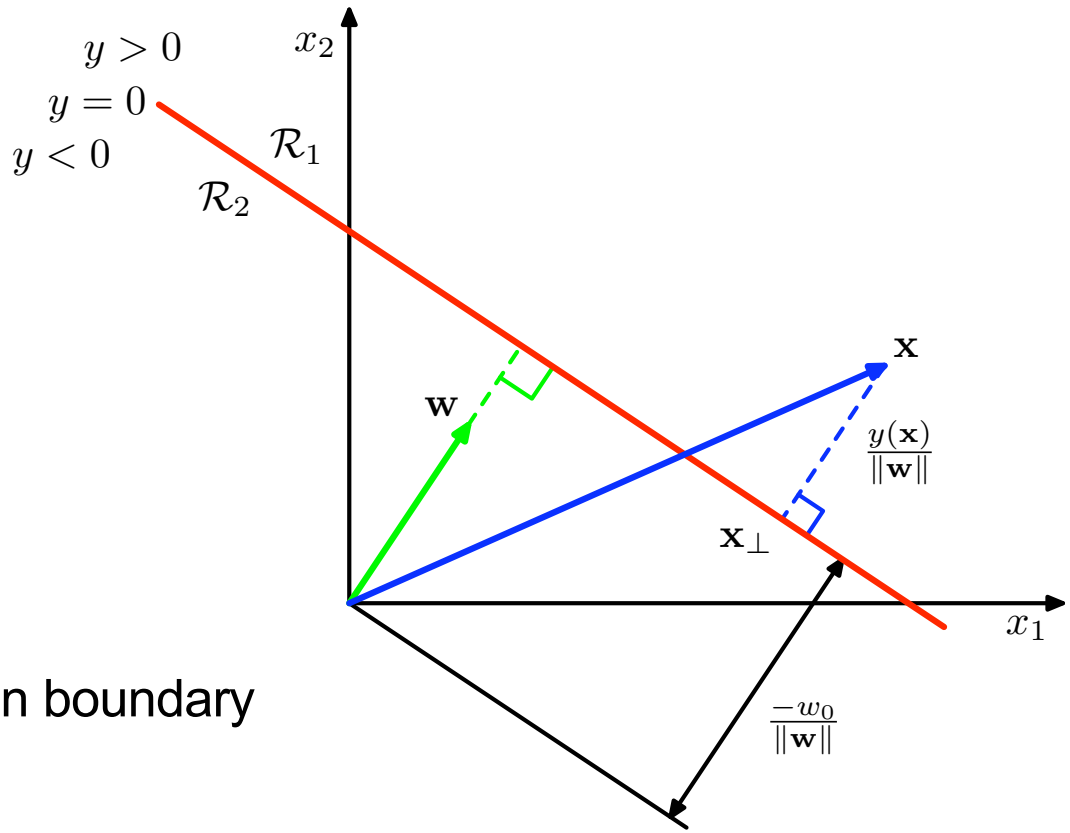
Recall:

$$y(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0$$

$y(\mathbf{x}) \geq 0 \rightarrow \mathbf{x}$ assigned to C_1

$y(\mathbf{x}) < 0 \rightarrow \mathbf{x}$ assigned to C_2

Thus $y(\mathbf{x}) = 0$ defines the decision boundary



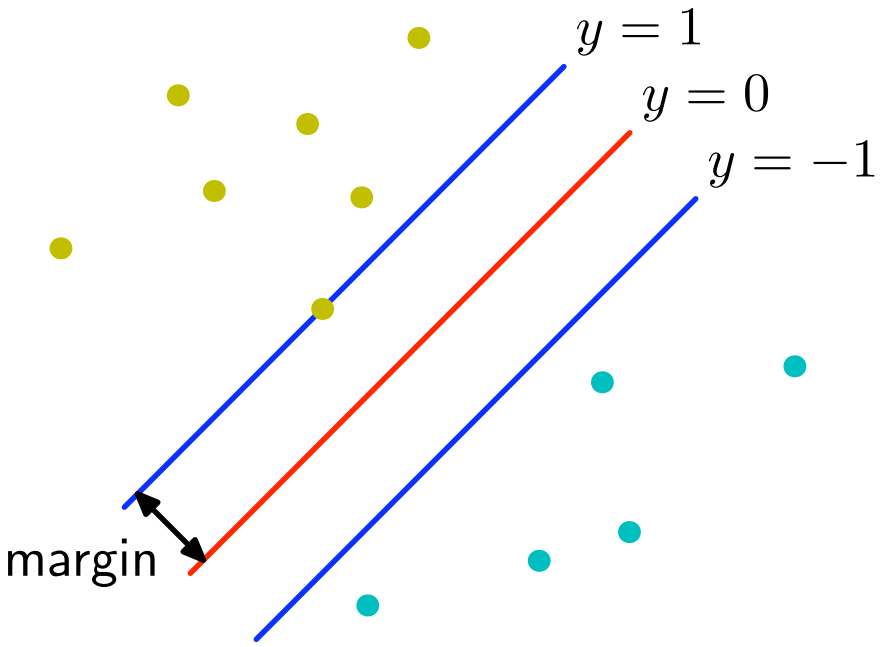
Maximum Margin Classifiers

Distance of point $\mathbf{x}_n$ from decision surface is given by:

$$\frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} = \frac{t_n\left(\mathbf{w}^t\phi(\mathbf{x}_n)+b\right)}{\|\mathbf{w}\|}$$

Thus we seek:

$$\arg\max_{\mathbf{w},b}\left\{\frac{1}{\|\mathbf{w}\|}\min_n\left[t_n\left(\mathbf{w}^t\phi(\mathbf{x}_n)+b\right)\right]\right\}$$



Maximum Margin Classifiers

Distance of point $\mathbf{x}_n$ from decision surface is given by:

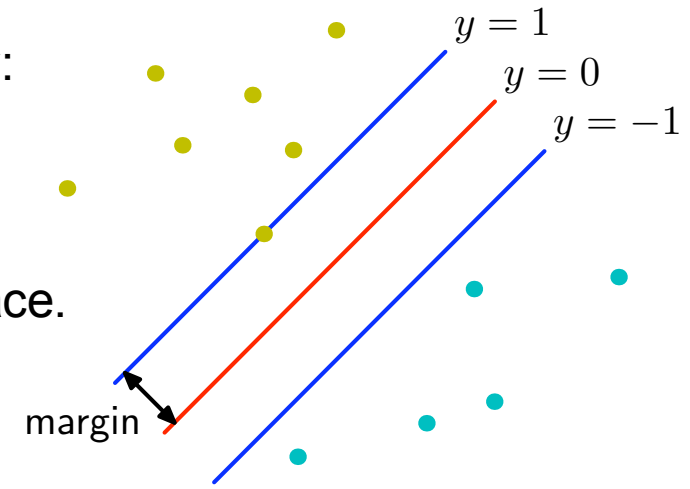
$$\frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} = \frac{t_n\left(\mathbf{w}^t\phi(\mathbf{x}_n)+b\right)}{\|\mathbf{w}\|}$$

Note that rescaling $\mathbf{w}$ and b by the same factor leaves the distance to the decision surface unchanged.

Thus, wlog, we consider only solutions that satisfy:

$$t_n\left(\mathbf{w}^t\phi(\mathbf{x}_n)+b\right)=1.$$

for the point $\mathbf{x}_n$ that is closest to the decision surface.



Quadratic Programming Problem

Then all points $\mathbf{x}_n$ satisfy $t_n\left(\mathbf{w}^t\phi\left(\mathbf{x}_n\right)+b\right)\geq 1$

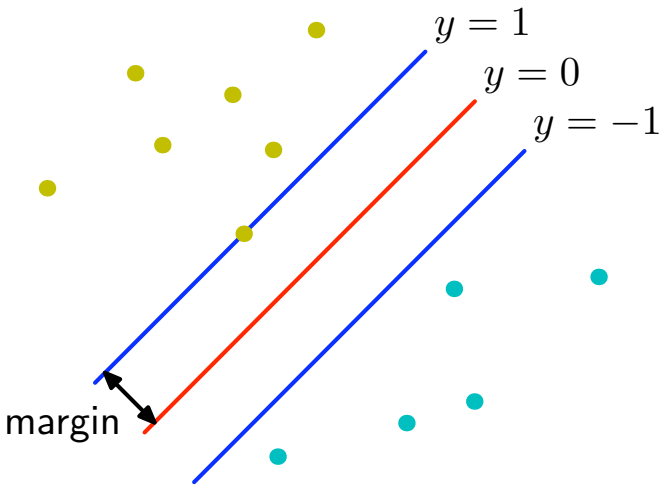
Points for which equality holds are said to be **active**.

All other points are **inactive**.

Now $\arg\max_{\mathbf{w},b}\left\{\frac{1}{\|\mathbf{w}\|}\min_n\left[t_n\left(\mathbf{w}^t\phi\left(\mathbf{x}_n\right)+b\right)\right]\right\}$

$\leftrightarrow \frac{1}{2}\arg\min_{\mathbf{w}}\|\mathbf{w}\|^2$

Subject to $t_n\left(\mathbf{w}^t\phi\left(\mathbf{x}_n\right)+b\right)\geq 1\ \forall\mathbf{x}_n$



This is a **quadratic programming** problem.

Solving this problem will involve **Lagrange multipliers**.

Quadratic Programming Problem

$$\frac{1}{2}\arg\min_{\mathbf{w}}\|\mathbf{w}\|^2, \text{ subject to } t_n\left(\mathbf{w}^t\phi\left(\mathbf{x}_n\right)+b\right)\geq 1\ \forall \mathbf{x}_n$$

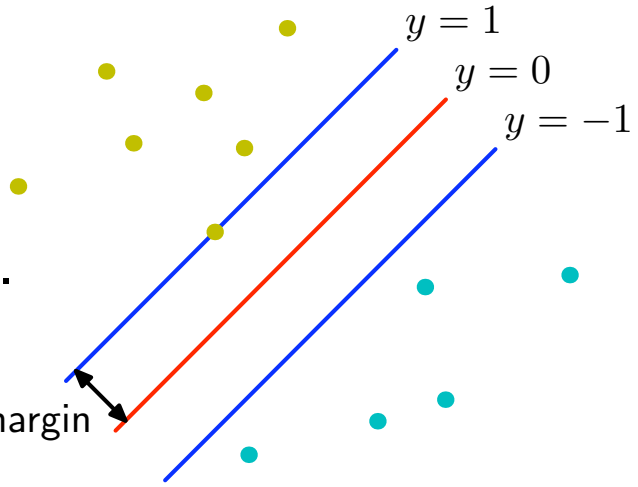
Solve using Lagrange multipliers a_n :

$$L(\mathbf{w},b,\mathbf{a})=\frac{1}{2}\|\mathbf{w}\|^2-\sum_{n=1}^{N}a_n\left\{t_n\left(\mathbf{w}^t\phi\left(\mathbf{x}_n\right)+b\right)-1\right\}$$

Always ≥ 0 Always ≥ 0

By convention, we maximize L with respect to the a_n.

Subtracting the Lagrange term $\rightarrow$ when $t_n y_n > 1$, $a_n = 0$.



Dual Representation

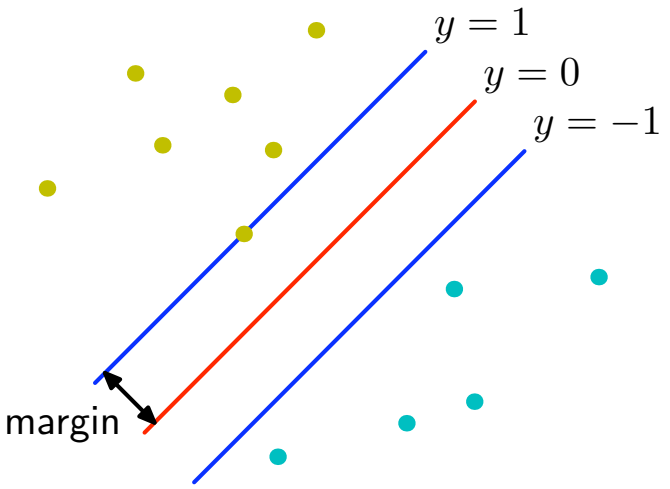
Solve using Lagrange multipliers a_n :

$$L(\mathbf{w},b,\mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n \left(\mathbf{w}^t \phi(\mathbf{x}_n) + b \right) - 1 \right\}$$

Setting derivatives with respect to $\mathbf{w}$ and b to 0, we get:

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n)$$

$$\sum_{n=1}^{N} a_n t_n = 0$$



Dual Representation

$$L(\mathbf{w},b,\mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n \left(\mathbf{w}^t \phi(\mathbf{x}_n) + b \right) - 1 \right\}$$

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n)$$

$$\sum_{n=1}^{N} a_n t_n = 0$$

Substituting leads to the dual representation

of the maximum margin problem, in which we maximize:

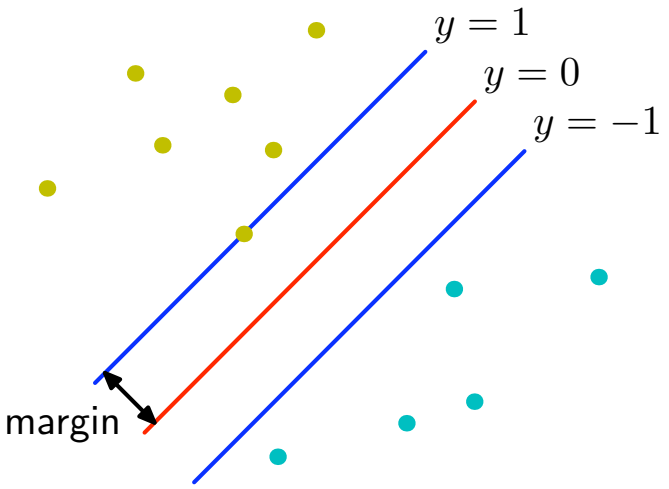
$$\tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

with respect to **a**, subject to:

$a_n \geq 0 \ \forall n$

$$\sum_{n=1}^{N} a_n t_n = 0$$

and where $k(\mathbf{x},\mathbf{x}') = \phi(\mathbf{x})^t \phi(\mathbf{x}')$



Dual Representation

Using $\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n)$, a new point x is classified by computing

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b$$

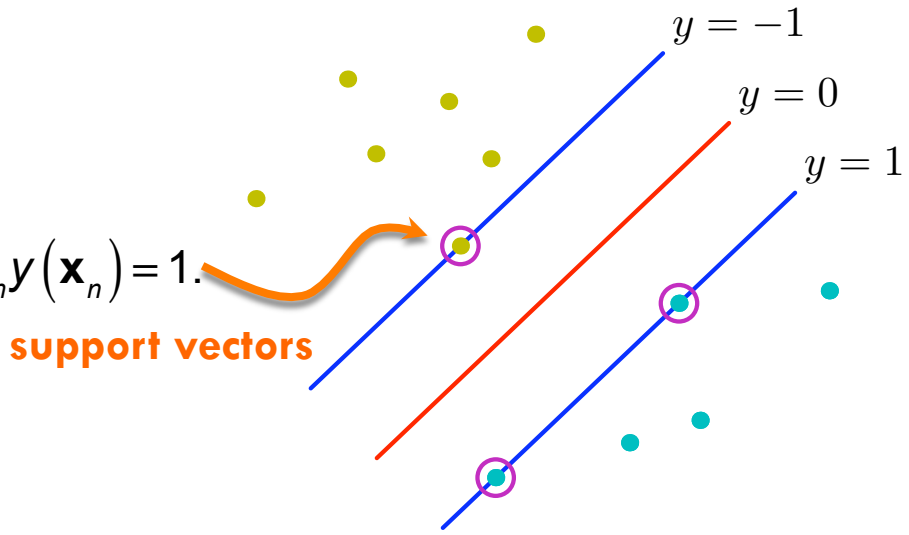
The Karush-Kuhn-Tucker (KKT) conditions for this constrained optimization problem are:

$a_n \geq 0$

$t_n y(\mathbf{x}_n) - 1 \geq 0$

$a_n \{t_n y(\mathbf{x}_n) - 1\} = 0$

Thus for every data point, either $a_n = 0$ or $t_n y(\mathbf{x}_n) = 1$.



Solving for the Bias

Once the optimal **a** is determined, the bias b can be computed by noting that any support vector $\mathbf{x}_n$ satisfies $t_n y(\mathbf{x}_n) = 1$.

Using $y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b$

we have $t_n \left(\sum_{m=1}^{N} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) + b \right) = 1$

and so $b = t_n - \sum_{m=1}^{N} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m)$

A more numerically accurate solution can be obtained by averaging over all support vectors:

$$b = \frac{1}{N_S} \sum_{n \in S} \left(t_n - \sum_{m \in S} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right)$$

where S is the index set of support vectors and N_S is the number of support vectors.

Kernel Methods: Outline

- Generalized Linear Models
- Radial Basis Function Networks
- Support Vector Machines
  - Separable classes
  - **Non-separable classes**
- The Kernel Trick

Overlapping Class Distributions

- The SVM for non-overlapping class distributions is determined by solving

$$\frac{1}{2}\arg\min_{\mathbf{w}}\|\mathbf{w}\|^2, \text{ subject to } t_n\left(\mathbf{w}^t\phi\left(\mathbf{x}_n\right)+b\right)\geq 1\ \forall\mathbf{x}_n$$

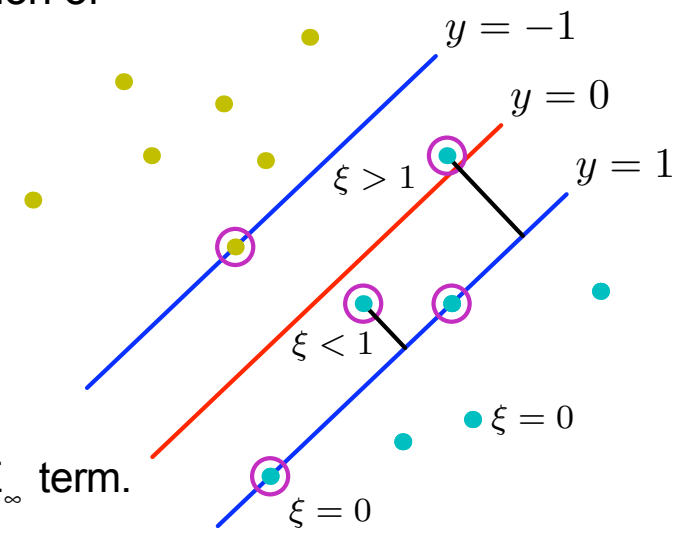
Alternatively, this can be expressed as the minimization of

$$\sum_{n=1}^{N} E_\infty\left(y\left(\mathbf{x}_n\right)t_n-1\right)+\lambda\|\mathbf{w}\|^2$$

where $E_\infty(z)$ is 0 if $z\geq 0$, and ∞ otherwise.

This forces all points to lie on or outside the margins, on the correct side for their class.

To allow for misclassified points, we have to relax this E_∞ term.



Slack Variables

To this end, we introduce N **slack variables** $\xi_n \geq 0,\ n = 1,\ldots N$.

$\xi_n = 0$ for points on or on the correct side of the margin boundary for their class

$\xi_n = \left| t_n - y\left(\mathbf{x}_n\right) \right|$ for all other points.

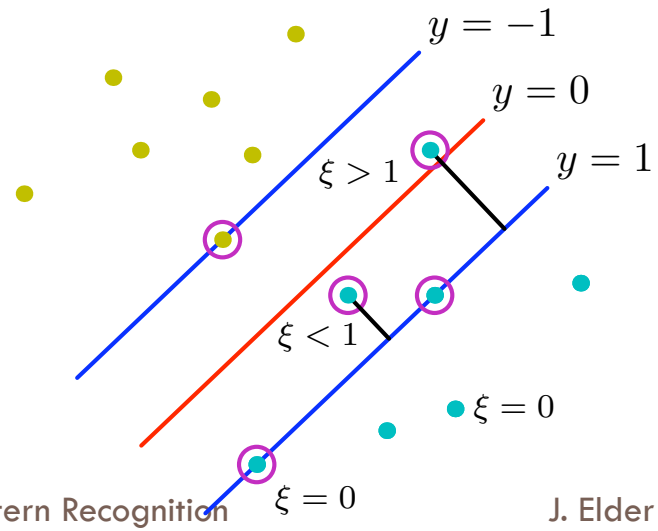
Thus $\xi_n < 1$ for points that are correctly classified

$\xi_n > 1$ for points that are incorrectly classified

We now minimize $C\sum_{n=1}^{N} \xi_n + \frac{1}{2}\|\mathbf{w}\|^2$, where $C > 0$.

subject to $t_n y\left(\mathbf{x}_n\right) \geq 1 - \xi_n$, and $\xi_n \geq 0,\ n = 1,\ldots N$

Think of ξ_n as the amount you must add to $t_n y\left(\mathbf{x}_n\right)$ to push it over to the right side of its margin.



Dual Representation

This leads to a dual representation, where we maximize

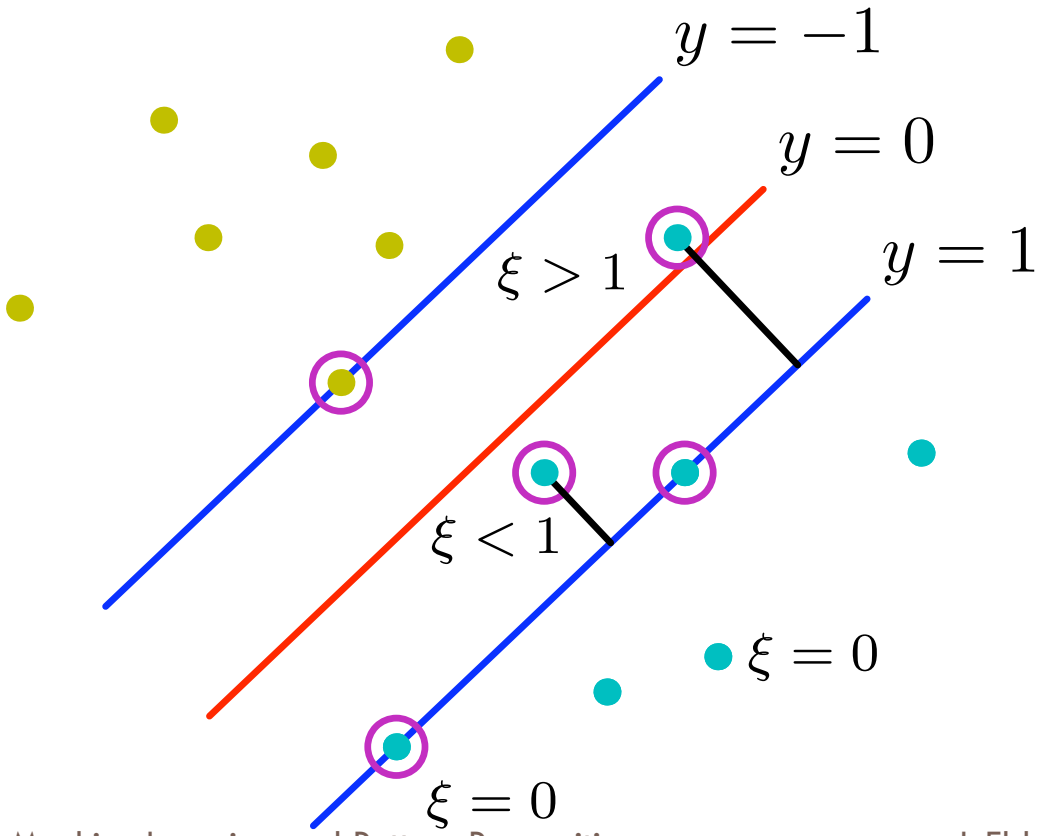
$$\tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

with constraints

$$0 \le a_n \le C$$

and

$$\sum_{n=1}^{N} a_n t_n = 0$$



Support Vectors

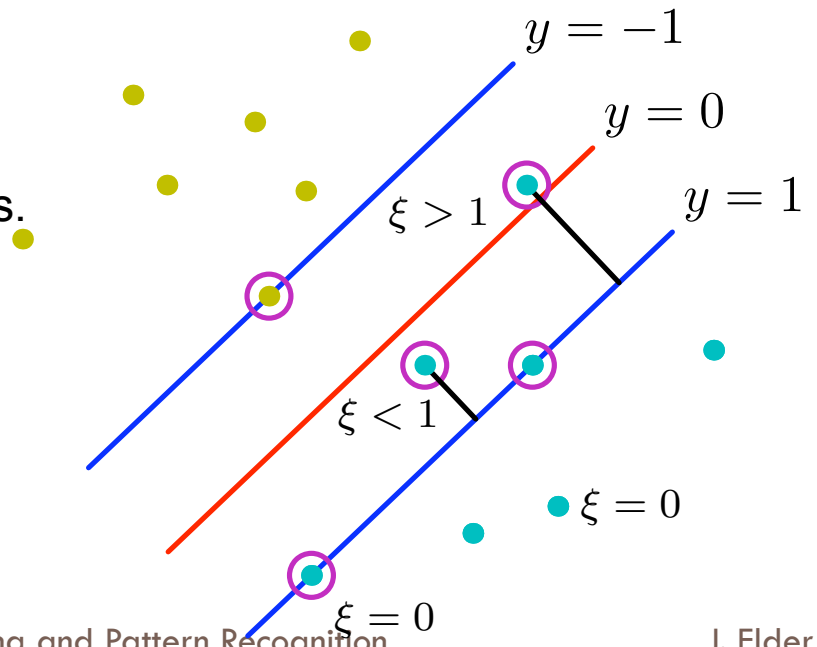
Again, a new point **x** is classified by computing

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b$$

For points that are on the correct side of the margin, $a_n = 0$.

Thus support vectors consist of points between their margin and the decision boundary, as well as misclassified points.

In other words, all points that are not on the right side of their margin are support vectors.



Bias

Again, a new point **x** is classified by computing

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b$$

Once the optimal **a** is determined, the bias b can be computed from

$$b = \frac{1}{N_{\mathcal{M}}} \sum_{n \in \mathcal{M}} \left(t_n - \sum_{m \in S} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right)$$

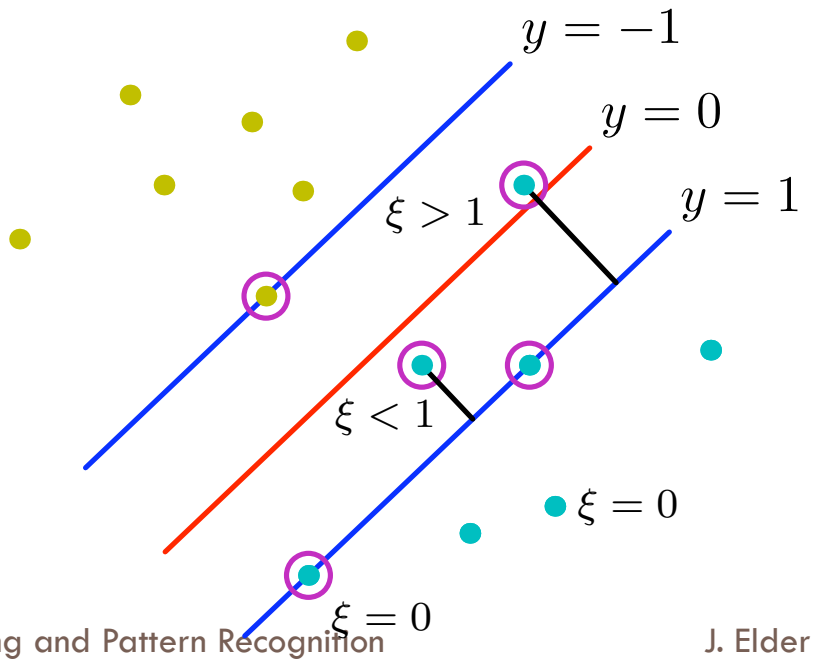
where

S is the index set of support vectors

N_s is the number of support vectors

$\mathcal{M}$ is the index set of points on the margins

$N_{\mathcal{M}}$ is the number of points on the margins



Solving the Quadratic Programming Problem

$$\text{Maximize } \tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

$$\text{subject to } 0 \le a_n \le C \text{ and } \sum_{n=1}^{N} a_n t_n = 0$$

- Problem is convex.
- Standard solutions are generally $O(N^3)$.
- Traditional quadratic programming techniques often infeasible due to computation and memory requirements.
- Instead, methods such as **sequential minimal optimization** can be used, that in practice are found to scale as $O(N)$ - $O(N^2)$.

Chunking

$$\text{Maximize } \tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

$$\text{subject to } 0 \le a_n \le C \text{ and } \sum_{n=1}^{N} a_n t_n = 0$$

- Conventional quadratic programming solution requires that matrices with N^2 elements be maintained in memory.

$$K \sim O(N^2), \text{ where } K_{nm} = k(\mathbf{x}_n, \mathbf{x}_m)$$

$$T \sim O(N^2), \text{ where } T_{nm} = t_n t_m$$

$$A \sim O(N^2), \text{ where } A_{nm} = a_n a_m$$

- This becomes infeasible when N exceeds ~10,000.

Chunking

$$\text{Maximize } \tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

$$\text{subject to } 0 \le a_n \le C \text{ and } \sum_{n=1}^{N} a_n t_n = 0$$

- Chunking (Vapnik, 1982) exploits the fact that the value of the Lagrangian is unchanged if we remove the rows and columns of the kernel matrix where $a_n = 0$ or $a_m = 0$.

Chunking

Minimize $C\sum_{n=1}^{N}\xi_n + \frac{1}{2}\|\mathbf{w}\|^2$, where $C > 0$.

$\xi_n = 0$ for points on or on the correct side of the margin boundary for their class

$\xi_n = |t_n - y(\mathbf{x}_n)|$ for all other points.

- Chunking (Vapnik, 1982)
  1. Select a small number (a 'chunk') of training vectors
  2. Solve the QP problem for this subset
  3. Retain only the support vectors
  4. Consider another chunk of the training data
  5. Ignore the subset of vectors in all chunks considered so far that lie on the correct side of the margin, since these do not contribute to the cost function
  6. Add the remainder to the current set of support vectors and solve the new QP problem
  7. Return to Step 4
  8. Repeat until the set of support vectors does not change.

This method reduces memory requirements to $O(N_s^2)$, where N_s is the number of support vectors.

This may still be big!

End of Lecture

November 19, 2012

Decomposition Methods

- It can be shown that the global QP problem is solved when all training vectors satisfy the following optimality conditions:

$$a_i = 0 \Leftrightarrow t_i y(\mathbf{x}_i) \geq 1.$$
$$0 < a_i < C \Leftrightarrow t_i y(\mathbf{x}_i) = 1.$$
$$a_i = C \Leftrightarrow t_i y(\mathbf{x}_i) \leq 1.$$

- Decomposition methods decompose this large QP problem into a series of smaller subproblems.
- Decomposition (Osuna et al, 1997)
  - Partition the training data into a small working subset B and a fixed subset N.
  - Minimize the global objective function by adjusting the coefficients in B
  - Swap 1 or more vectors in B for an equal number in N that fail to satisfy the optimality conditions
  - Re-solve the global QP problem for B
- Each step is $O(B)^2$ in memory.
- Osuna et al (1997) proved that the objective function decreases on each step and will converge in a finite number of iterations.

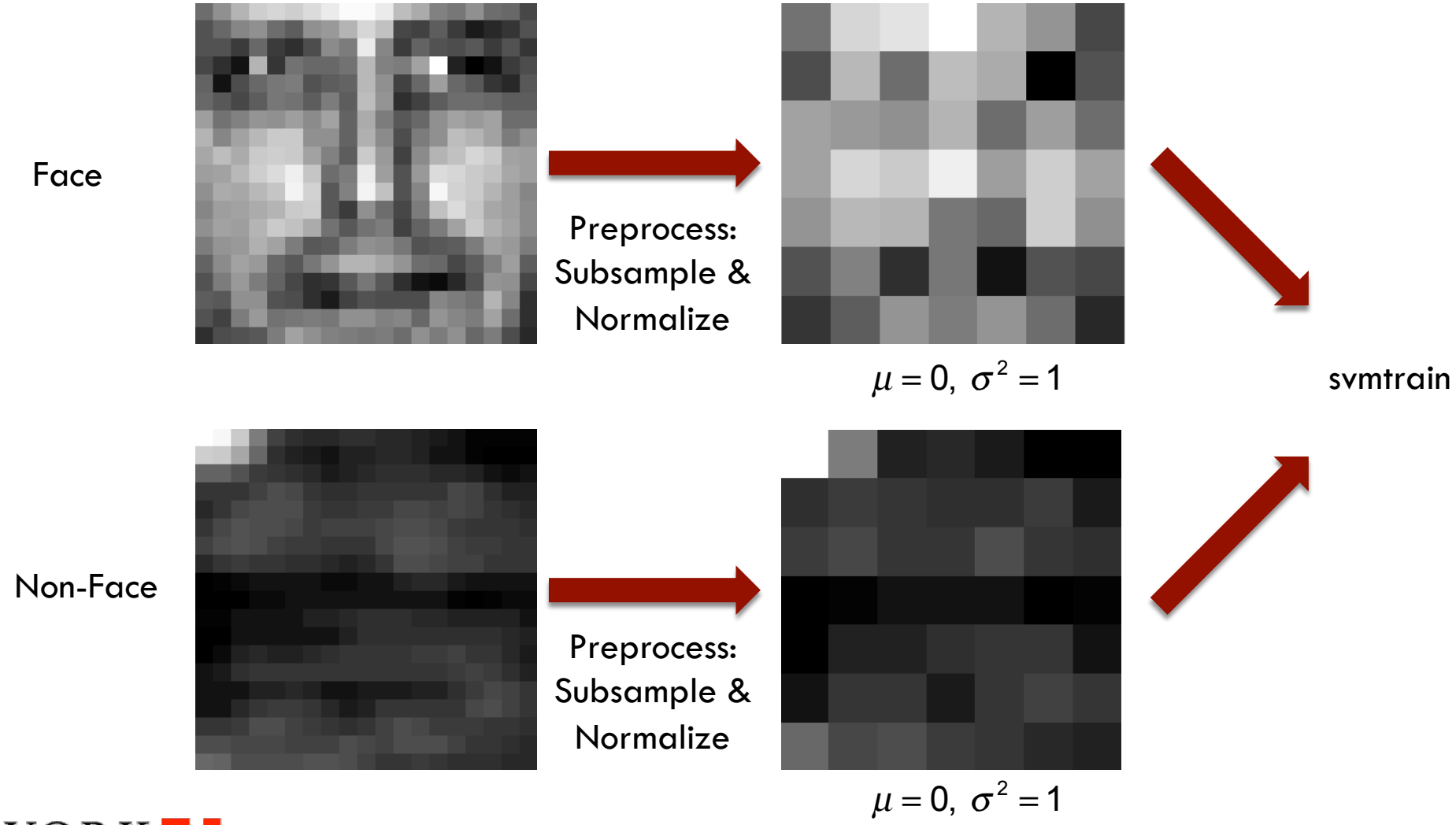
Sequential Minimal Optimization

- Sequential Minimal Optimization (Platt 1998) takes decomposition to the limit.
- On each iteration, the working set consists of just two vectors.
- The advantage is that in this case, the QP problem can be solved analytically.
- Memory requirement are $O(N)$.
- Compute time is typically $O(N) - O(N^2)$.

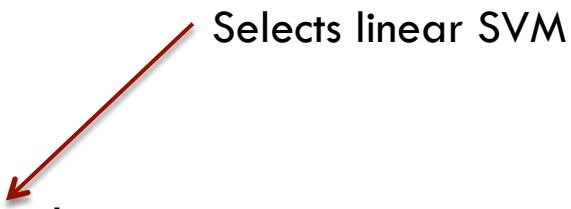
LIBSVM

- LIBSVM is a widely used library for SVMs developed by Chang & Lin (2001).
  - Can be downloaded from www.csie.ntu.edu.tw/~cjlin/libsvm
  - MATLAB interface
  - Uses SMO
  - Will use for Assignment 2.

LIBSVM Example: Face Detection



LIBSVM Example: MATLAB Interface




```
model=svmtrain(traint, trainx, '-t 0');

[predicted_label, accuracy, decision_values] = svmpredict(testt, testx, model);
```

Accuracy = 70.0212% (661/944) (classification)

Relation to Logistic Regression

The objective function for the soft-margin SVM can be written as:

$$\sum_{n=1}^{N} E_{SV}\left(y_n t_n\right) + \lambda \left\|\mathbf{w}\right\|^2$$

where $E_{SV}\left(z\right) = \left[1 - z\right]_+$ is the **hinge error function**,

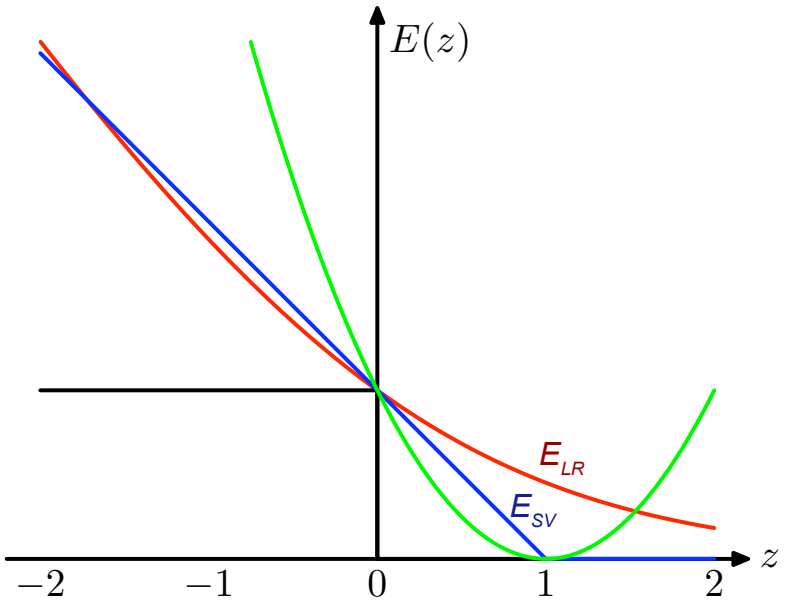
and $\left[z\right]_+ = z$ *if* $z \geq 0$

$= 0$ otherwise.

For $t \in \{-1, 1\}$, the objective function for a regularized version of logistic regression can be written as:

$$\sum_{n=1}^{N} E_{LR}\left(y_n t_n\right) + \lambda \left\|\mathbf{w}\right\|^2$$

where $E_{LR}\left(z\right) = \log\left(1 + \exp(-z)\right)$.



Kernel Methods: Outline

- Generalized Linear Models
- Radial Basis Function Networks
- Support Vector Machines
  - Separable classes
  - Non-separable classes
- **The Kernel Trick**

The Kernel Function

- Recall that an SVM is the solution to the problem

$$\text{Maximize } \tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

$$\text{subject to } 0 \le a_n \le C \text{ and } \sum_{n=1}^{N} a_n t_n = 0$$

- A new input x is classified by computing

$$y(\mathbf{x}) = \sum_{n \in S} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b$$

- Where S is the set of support vectors.
- Here we introduced the kernel function $k(x, x')$, defined as

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^t \phi(\mathbf{x}')$$

- This is more than a notational convenience!!

The Kernel Trick

$$\text{Maximize } \tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

$$\text{subject to } 0 \le a_n \le C \text{ and } \sum_{n=1}^{N} a_n t_n = 0$$

$$\text{where } k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^t \phi(\mathbf{x}')$$

- Note that the basis functions and individual training vectors are no longer part of the objective function.
- Instead all we need is the kernel value (like a distance measure) for all pairs of training vectors.

The Kernel Function

The kernel function $k(\mathbf{x},\mathbf{x}')$ measures the 'similarity' of input vectors $\mathbf{x}$ and $\mathbf{x}'$ as an inner product in a feature space defined by the feature space mapping $\phi(\mathbf{x})$:

$$k(\mathbf{x},\mathbf{x}') = \phi(\mathbf{x})^t \phi(\mathbf{x}')$$

If $k(\mathbf{x},\mathbf{x}') = k(\mathbf{x}-\mathbf{x}')$ we say that the kernel is *stationary*

If $k(\mathbf{x},\mathbf{x}') = k\left(\left\|\mathbf{x}-\mathbf{x}'\right\|\right)$ we call it a *radial basis function*.

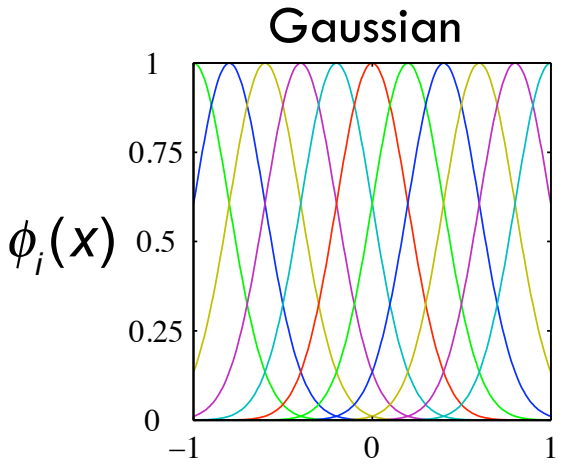
Constructing Kernels

- We can construct a kernel by selecting a feature space mapping $\phi(x)$ and then defining

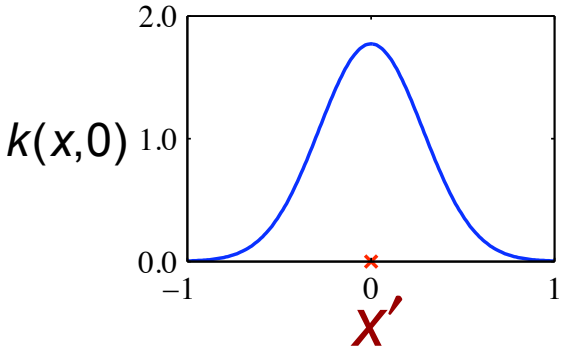
$$k(\mathbf{x},\mathbf{x}') = \phi(\mathbf{x})^t \phi(\mathbf{x}') = \sum_{i=1}^{M} \phi_i(\mathbf{x})^t \phi_i(\mathbf{x}')$$

- 1D Example:

$$\phi_i(x) = \exp\left(-\frac{(x-\mu_i)^2}{2\sigma^2}\right)$$



$$k(x,0) = \sum_{i=1}^{M} \exp\left(-\frac{(x-\mu_i)^2}{2\sigma^2}\right) \exp\left(-\frac{\mu_i^2}{2\sigma^2}\right)$$



Constructing Kernels

- Alternatively, we can construct the kernel function directly, ensuring that it corresponds to an inner product in some (possibly infinite-dimensional) feature space.

Constructing Kernels

$$k(x) = \phi(x)^t \phi(x')$$

Example 1: $k(x,z) = x^t z$

Example 2: $k(x,z) = x^t z + c, \; c > 0$

Example 3: $k(x,z) = \left(x^t z\right)^2$

Kernel Properties

- Kernels obey certain properties that make it easy to construct complex kernels from simpler ones.

Kernel Properties

Given valid kernels $k_1(\boldsymbol{x}, \boldsymbol{x}')$ and $k_2(\boldsymbol{x}, \boldsymbol{x}')$ the following kernels will also be valid:

$$
\begin{aligned}
k(\boldsymbol{x}, \boldsymbol{x}') &= c k_1(\boldsymbol{x}, \boldsymbol{x}') && (6.13)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= f(\boldsymbol{x}) k_1(\boldsymbol{x}, \boldsymbol{x}') f(\boldsymbol{x}') && (6.14)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= q(k_1(\boldsymbol{x}, \boldsymbol{x}')) && (6.15)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= exp(k_1(\boldsymbol{x}, \boldsymbol{x}')) && (6.16)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= k_1(\boldsymbol{x}, \boldsymbol{x}') + k_2(\boldsymbol{x}, \boldsymbol{x}') && (6.17)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= k_1(\boldsymbol{x}, \boldsymbol{x}') k_2(\boldsymbol{x}, \boldsymbol{x}') && (6.18)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= k_3(\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}')) && (6.19)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= \boldsymbol{x}^T \mathbf{A} \boldsymbol{x}' && (6.20)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= k_a(\boldsymbol{x}_a, \boldsymbol{x}'_a) + k_b(\boldsymbol{x}_b, \boldsymbol{x}'_b) && (6.21)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= k_a(\boldsymbol{x}_a, \boldsymbol{x}'_a) k_b(\boldsymbol{x}_b, \boldsymbol{x}'_b) && (6.22)
\end{aligned}
$$

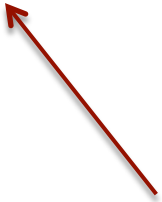
where $c > 0$, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\phi(\mathbf{x})$ is a mapping from $\mathbf{x} \rightarrow \mathbb{R}^M$, k_3 is a valid kernel on $\mathbb{R}^M$, A is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables such that $\mathbf{x}^t = \left(\mathbf{x}_a^t, \mathbf{x}_b^t\right)$ and k_a, k_b are valid kernels over their respective spaces.

Constructing Kernels

- Examples:

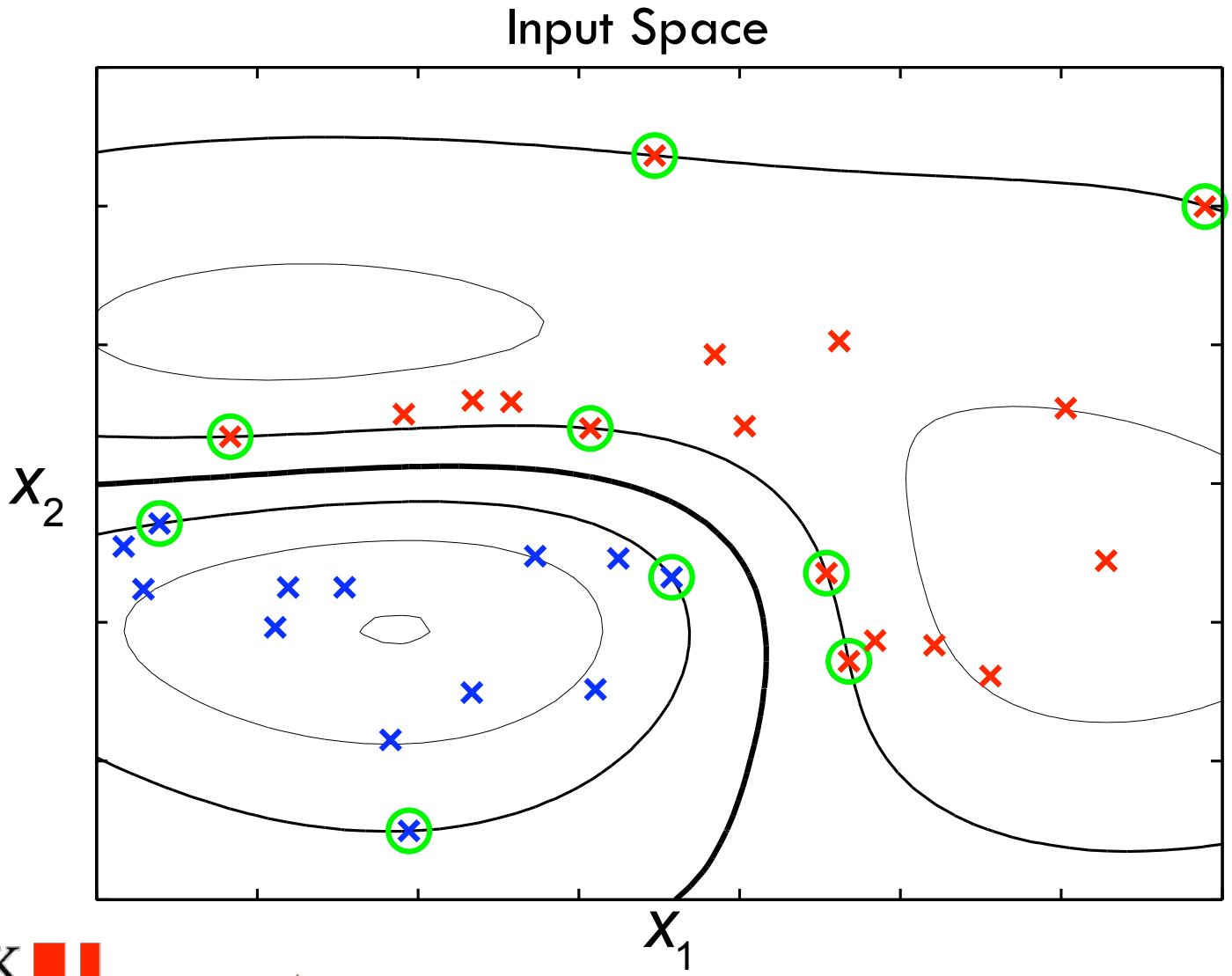
$$k(x,x') = \left(x^t x' + c\right)^M, c > 0 \qquad \text{(Use 6.18)}$$

$$k(x,x') = \exp\left(-\left\|x - x'\right\|^2 / 2\sigma^2\right) \qquad \text{(Use 6.14 and 6.16.)}$$



Corresponds to infinite-dimensional feature vector

Nonlinear SVM Example (Gaussian Kernel)



Kernel Methods: Outline

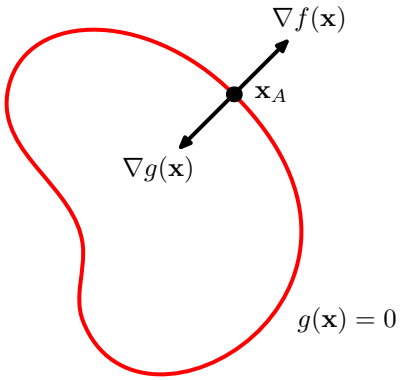
- Generalized Linear Models
- Radial Basis Function Networks
- Support Vector Machines
  - Separable classes
  - Non-separable classes
- The Kernel Trick

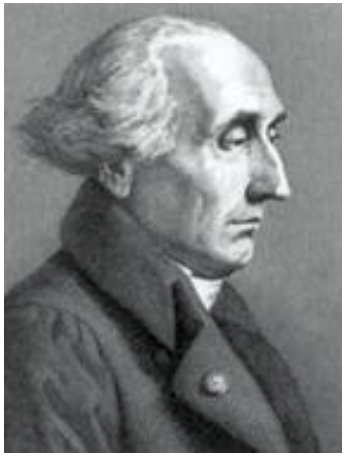
Lagrange Multipliers

Lagrange Multipliers (Appendix C.4 in Bishop)

- Used to find stationary points of a function subject to one or more constraints.
- Example (equality constraint):

Maximize $f(\mathbf{x})$ subject to $g(\mathbf{x}) = 0$.





Joseph-Louis Lagrange
1736-1813

- Observations:

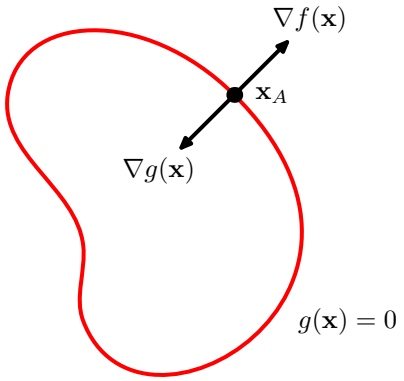
1. At any point on the constraint surface, $\nabla g(\mathbf{x})$ must be orthogonal to the surface.

2. Let $\mathbf{x}^*$ be a point on the constraint surface where $f(\mathbf{x})$ is maximized.
   Then $\nabla f(\mathbf{x})$ is also orthogonal to the constraint surface.

3. $\rightarrow \exists \lambda \neq 0$ such that $\nabla f(\mathbf{x}) + \lambda \nabla g(\mathbf{x}) = 0$ at $\mathbf{x}^*$.
   λ is called a **Lagrange multiplier**.

Lagrange Multipliers (Appendix C.4 in Bishop)

$\exists \lambda \neq 0$ such that $\nabla f(\mathbf{x}) + \lambda \nabla g(\mathbf{x}) = 0$ at $\mathbf{x}^*$.



- Defining the **Lagrangian** function as:

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$$

we then have

$$\nabla_{\mathbf{x}} L(\mathbf{x}, \lambda) = 0.$$

and

$$\frac{\partial L(\mathbf{x}, \lambda)}{\partial \lambda} = 0.$$

Example

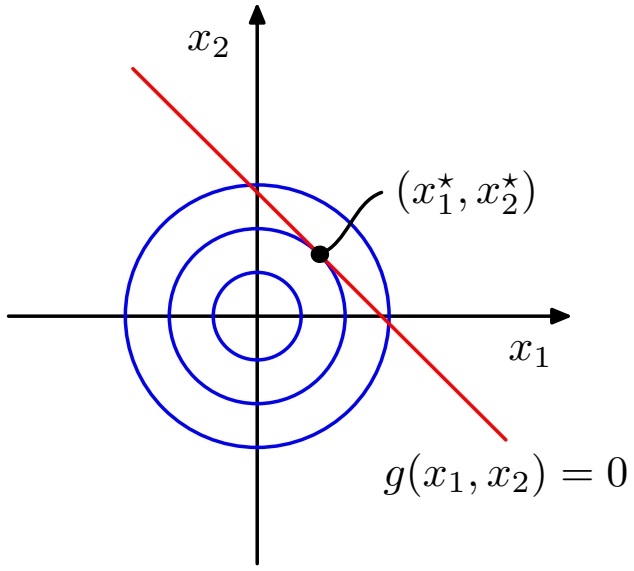
$$L(\mathbf{x},\lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$$

- Find the stationary point of

$$f(x_1, x_2) = 1 - x_1^2 - x_2^2$$

subject to

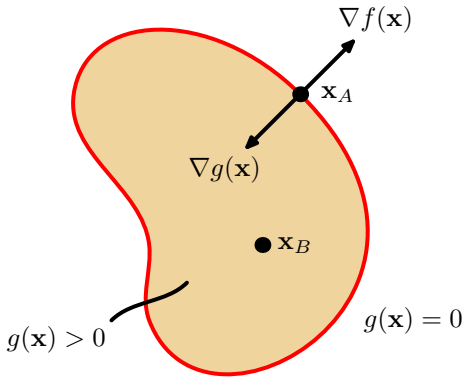
$$g(x_1, x_2) = x_1 + x_2 - 1 = 0$$



Inequality Constraints

Maximize $f(\mathbf{x})$ subject to $g(\mathbf{x}) \geq 0$.

- There are 2 cases:
  1. $\mathbf{x}^*$ on the interior (e.g., $\mathbf{x}_B$)
     - Here g(x) > 0 and the stationary condition is simply
       $\nabla f(\mathbf{x}) = 0.$
     - This corresponds to a stationary point of the Lagrangian where
       $\lambda = 0.$
  2. $\mathbf{x}^*$ on the boundary (e.g., $\mathbf{x}_A$)
     - Here g(x) = 0 and the stationary condition is
       $\nabla f(\mathbf{x}) = -\lambda \nabla g(x),\ \lambda > 0.$
     - This corresponds to a stationary point of the Lagrangian where
       $\lambda > 0.$



$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$$

- Thus the general problem can be expressed as maximizing the Lagrangian subject to
  1. $g(x) \geq 0$
  2. $\lambda \geq 0$
  3. $\lambda g(x) = 0$

  } **Karush-Kuhn-Tucker (KKT) conditions**

Minimizing vs Maximizing

- If we want to **minimize** $f(x)$ subject to $g(x) \geq 0$, then the Lagrangian becomes

$$L(\mathbf{x},\lambda) = f(\mathbf{x}) - \lambda g(\mathbf{x})$$

with $\lambda \geq 0$.

Extension to Multiple Constraints

- Suppose we wish to maximize $f(\mathbf{x})$ subject to

$$g_j(\mathbf{x}) = 0 \text{ for } j = 1,\ldots,J$$

$$h_k(\mathbf{x}) \geq 0 \text{ for } k = 1,\ldots,K$$

- We then find the stationary points of

$$L(\mathbf{x},\lambda) = f(\mathbf{x}) + \sum_{j=1}^{J} \lambda_j g_j(\mathbf{x}) + \sum_{k=1}^{K} \mu_k h_k(\mathbf{x})$$

subject to

$$h_k(\mathbf{x}) \geq 0$$

$$\mu_k \geq 0$$

$$\mu_k h_k(\mathbf{x}) = 0$$