Support Vector Machines

- Hypothesis Space
  - variable size
  - deterministic
  - continuous parameters
- Learning Algorithm
  - linear and quadratic programming
  - eager
  - batch
- SVMs combine three important ideas
  - Apply optimization algorithms from Operations Reseach (Linear Programming and Quadratic Programming)
  - Implicit feature transformation using kernels
  - Control of overfitting by maximizing the margin

White Lie Warning

- This first introduction to SVMs describes a special case with simplifying assumptions
- We will revisit SVMs later in the quarter and remove the assumptions
- The material you are about to see does not describe "real" SVMs

Linear Programming

- The linear programming problem is the following:

  find $\mathbf{w}$

  minimize $c \cdot \mathbf{w}$

  subject to

  $\mathbf{w} \cdot a_i = b_i$ for $i = 1, \ldots, m$

  $\mathbf{w}_j \geq 0$ for $j = 1, \ldots, n$

- There are fast algorithms for solving linear programs including the simplex algorithm and Karmarkar's algorithm

Formulating LTU Learning as Linear Programming

- Encode classes as {+1, –1}
- LTU:

  $h(x) = +1$ if $\mathbf{w} \cdot \mathbf{x} \geq 0$

  $= -1$ otherwise
- An example $(\mathbf{x}_i, y_i)$ is classified correctly by h if

  $y_i \cdot \mathbf{w} \cdot \mathbf{x}_i > 0$
- Basic idea: The constraints on the linear programming problem will be of the form

  $y_i \cdot \mathbf{w} \cdot \mathbf{x}_i > 0$
- We need to introduce two more steps to convert this into the standard format for a linear program

Converting to Standard LP Form

- Step 1: Convert to equality constraints by using slack variables
  - Introduce one slack variable s_i for each training example $\mathbf{x}_i$ and require that $s_i \geq 0$:

    $y_i \cdot \mathbf{w} \cdot \mathbf{x}_i - s_i = 0$
- Step 2: Make all variables positive by subtracting pairs
  - Replace each w_j be a difference of two variables: $w_j = u_j - v_j$, where $u_j, v_j \geq 0$

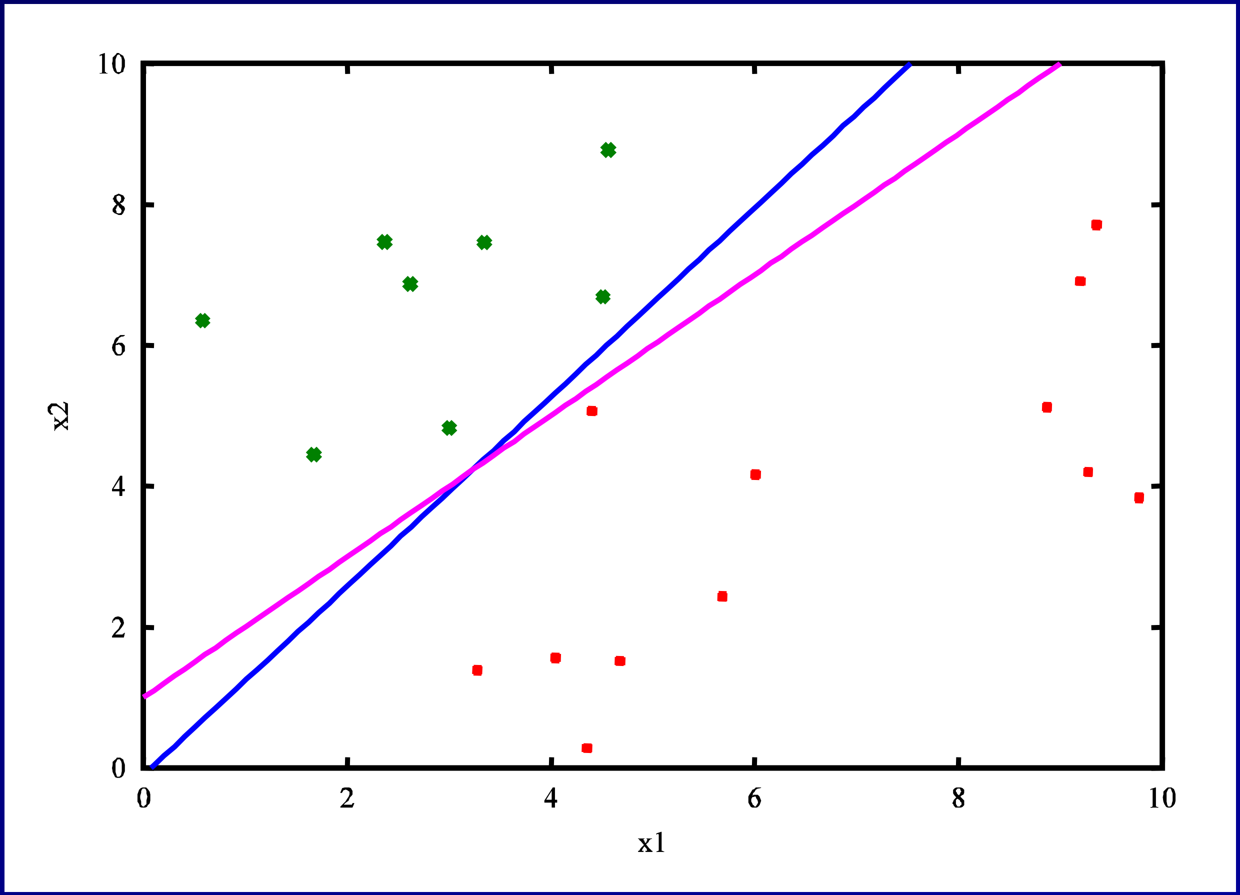
    $y_i \cdot (\mathbf{u} - \mathbf{v}) \cdot \mathbf{x}_i - s_i = 0$
- Linear program:
  - Find s_i, u_j, v_j
  - Minimize (no objective function)
  - Subject to:

    $y_i \left(\sum_j (u_j - v_j) x_{ij}\right) - s_i = 0$

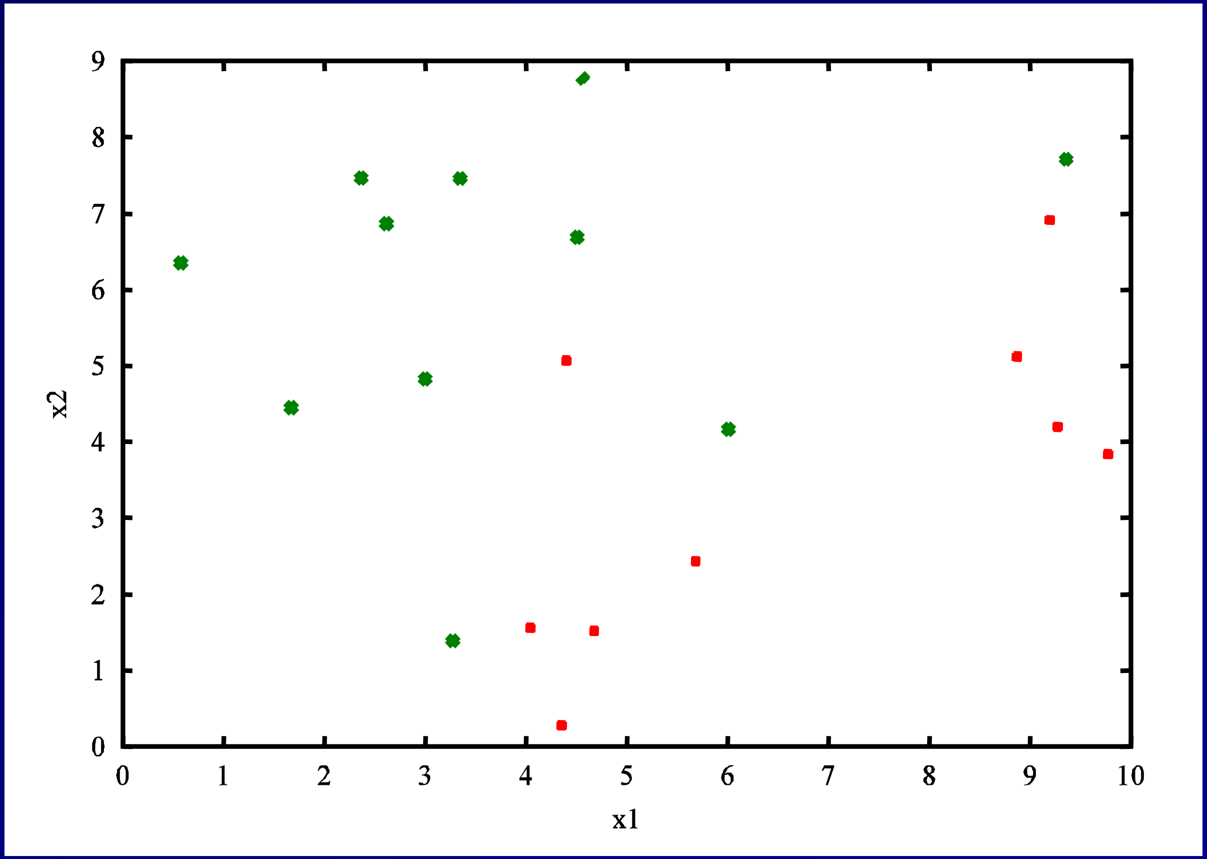
    $s_i \geq 0,\ u_j \geq 0,\ v_j \geq 0$
- The linear program will have a solution iff the points are linearly separable

Example

- 30 random data points labeled according to the line $x_2 = 1 + x_1$
- Pink line is true classifier
- Blue line is the linear programming fit



What Happens with Non-Separable Data?



- Bad News: Linear Program is Infeasible

Higher Dimensional Spaces

- Theorem: For <u>any</u> data set, there exists a mapping Φ to a higher-dimensional space such that the data is linearly separable
- $\Phi(\mathbf{X}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots, \phi_D(\mathbf{x}))$
- Example: Map to quadratic space
  - $\mathbf{x} = (x_1, x_2)$ (just two features)
  - $\Phi(\mathbf{x}) = (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2, \sqrt{2}\, x_1, \sqrt{2}\, x_2, 1)$
  - compute linear separator in this space

Drawback of this approach

- The number of features increases rapidly
- This makes the linear program much slower to solve

Kernel Trick

- A dot product between two higher-dimensional mappings can sometimes be implemented by a <u>kernel function</u>

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

- Example: Quadratic Kernel

$$
\begin{aligned}
K(\mathbf{x}_i, \mathbf{x}_j) &= (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^2 \\
&= (x_{i1}x_{j1} + x_{i2}x_{j2} + 1)^2 \\
&= x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} + x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + 1 \\
&= (x_{i1}^2, \sqrt{2}\, x_{i1}x_{i2}, x_{i2}^2, \sqrt{2}\, x_{i1}, \sqrt{2}\, x_{i2}, 1) \cdot \\
&\quad\; (x_{j1}^2, \sqrt{2}\, x_{j1}x_{j2}, x_{j2}^2, \sqrt{2}\, x_{j1}, \sqrt{2}\, x_{j2}, 1) \\
&= \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)
\end{aligned}
$$

Idea

- Reformulate the LTU linear program so that it only involves dot products between pairs of training examples
- Then we can use kernels to compute these dot products
- Running time of algorithm will not depend on number of dimensions D of high-dimensional space

Reformulating the LTU Linear Program

- Claim: In online Perceptron, **w** can be written as

  $\mathbf{w} = \sum_j \alpha_j \, y_j \, \mathbf{x}_j$

- Proof:
  - Each weight update has the form

    $\mathbf{w}_t := \mathbf{w}_{t-1} + \eta \, \mathbf{g}_{i,t}$
  - $\mathbf{g}_{i,t}$ is computed as

    $\mathbf{g}_{i,t} = \text{error}_{it} \, y_i \, \mathbf{x}_i$ ($\text{error}_{it} = 1$ if x_i misclassified in iteration t; 0 otherwise)
  - Hence

    $\mathbf{w}_t = \mathbf{w}_0 + \sum_t \sum_i \eta \, \text{error}_{it} \, y_i \, \mathbf{x}_i$
  - But $\mathbf{w}_0 = (0, 0, \ldots, 0)$, so

    $\mathbf{w}_t = \sum_t \sum_i \eta \, \text{error}_{it} \, y_i \, \mathbf{x}_i$

    $\mathbf{w}_t = \sum_i (\sum_t \eta \, \text{error}_{it}) \, y_i \, \mathbf{x}_i$

    $\mathbf{w}_t = \sum_i \alpha_i \, y_i \, \mathbf{x}_i$

Rewriting the Linear Separator Using Dot Products

$$\mathbf{w} \cdot \mathbf{x}_i = \left(\sum_j \alpha_j y_j \mathbf{x}_j \right) \cdot \mathbf{x}_i$$
$$= \sum_j \alpha_j y_j (\mathbf{x}_j \cdot \mathbf{x}_i)$$

- Change of variables
  - instead of optimizing **w**, optimize $\{\alpha_j\}$.
  - Rewrite the constraint

    $y_i \, \mathbf{w} \cdot \mathbf{x}_i > 0$ as

    $y_i \sum_j \alpha_j y_j (\mathbf{x}_j \cdot \mathbf{x}_i) > 0$ or

    $\sum_j \alpha_j y_j y_i (\mathbf{x}_j \cdot \mathbf{x}_i) > 0$

The Linear Program becomes

– Find $\{\alpha_j\}$
– minimize (no objective function)
– subject to

$\sum_j \alpha_j\, y_j\, y_i\, (\mathbf{x}_j \cdot \mathbf{x}_i) > 0$

$\alpha_j \geq 0$

– Notes:
  - The weight α_j tells us how "important" example $\mathbf{x}_j$ is. If α_j is non-zero, then $\mathbf{x}_j$ is called a "support vector"
  - To classify a new data point $\mathbf{x}$, we take its dot product with the support vectors

    $\sum_j \alpha_j\, y_j\, (\mathbf{x}_j \cdot \mathbf{x}) > 0$?

Kernel Version of Linear Program

– Find $\{\alpha_j\}$

– minimize (no objective function)

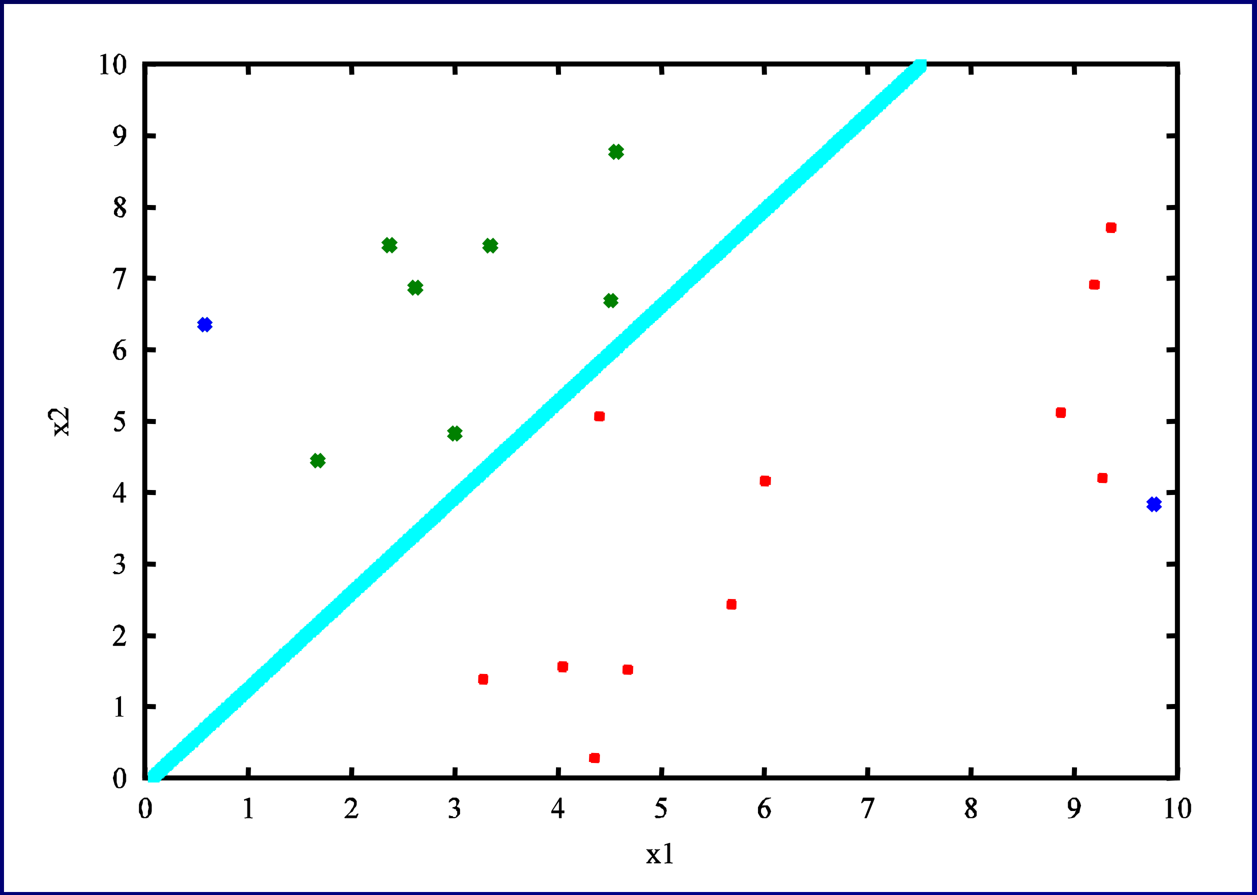
– subject to

$$\sum_j \alpha_j\, y_j\, y_i\, K(\mathbf{x}_j,\mathbf{x}_i) > 0$$

$$\alpha_j \geq 0$$

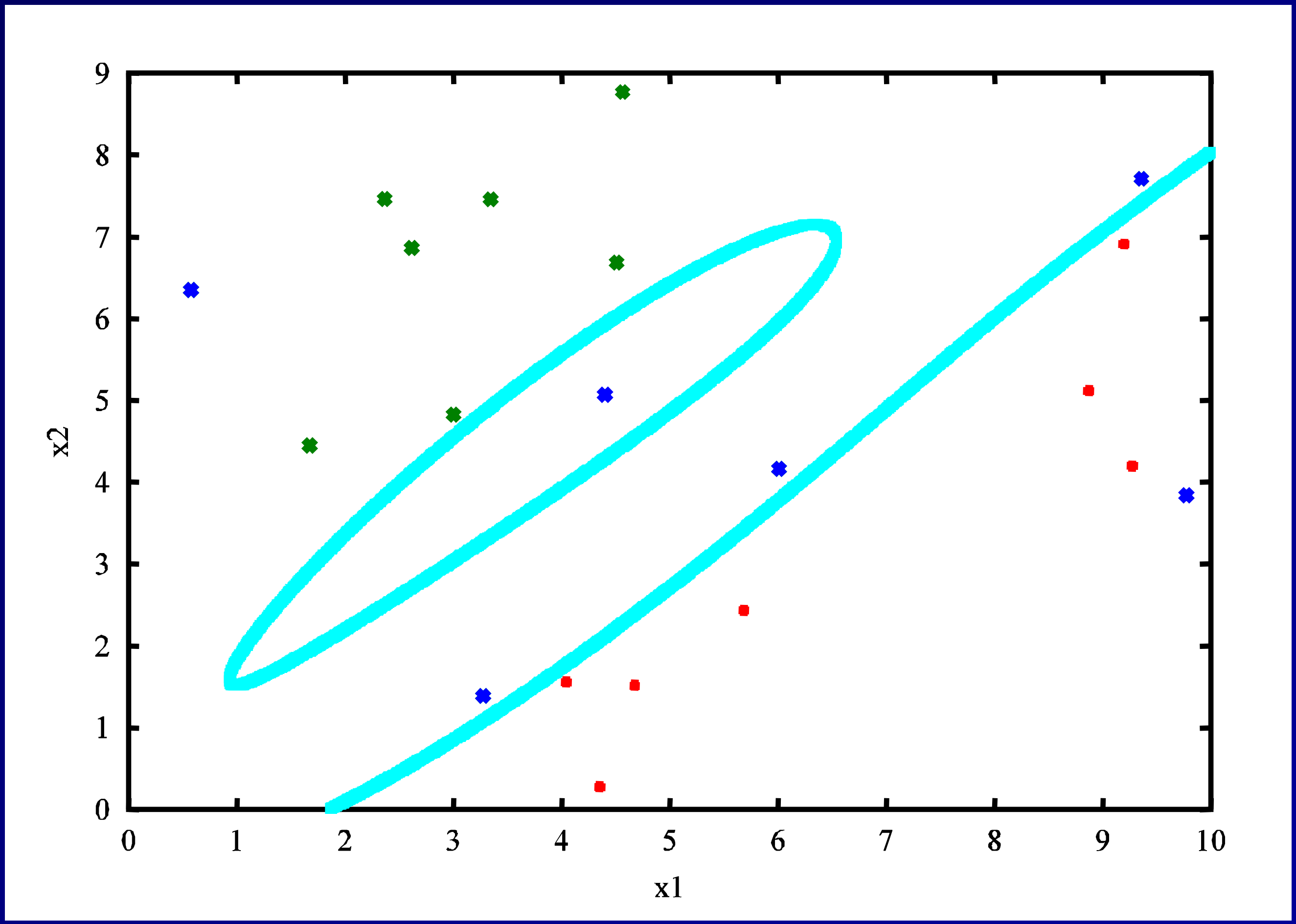
- Classify new x according to

$$\sum_j \alpha_j\, y_j\, K(\mathbf{x}_j,\mathbf{x}) > 0?$$



- Two support vectors (blue) with α_1 = 0.205 and α_2 = 0.338
- Equivalent to the line $x_2 = -0.0974 + x_1 * 1.341$

Solving the Non-Separable Case with Cubic Polynomial Kernel



Kernels

- Dot product

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

- Polynomial of degree d

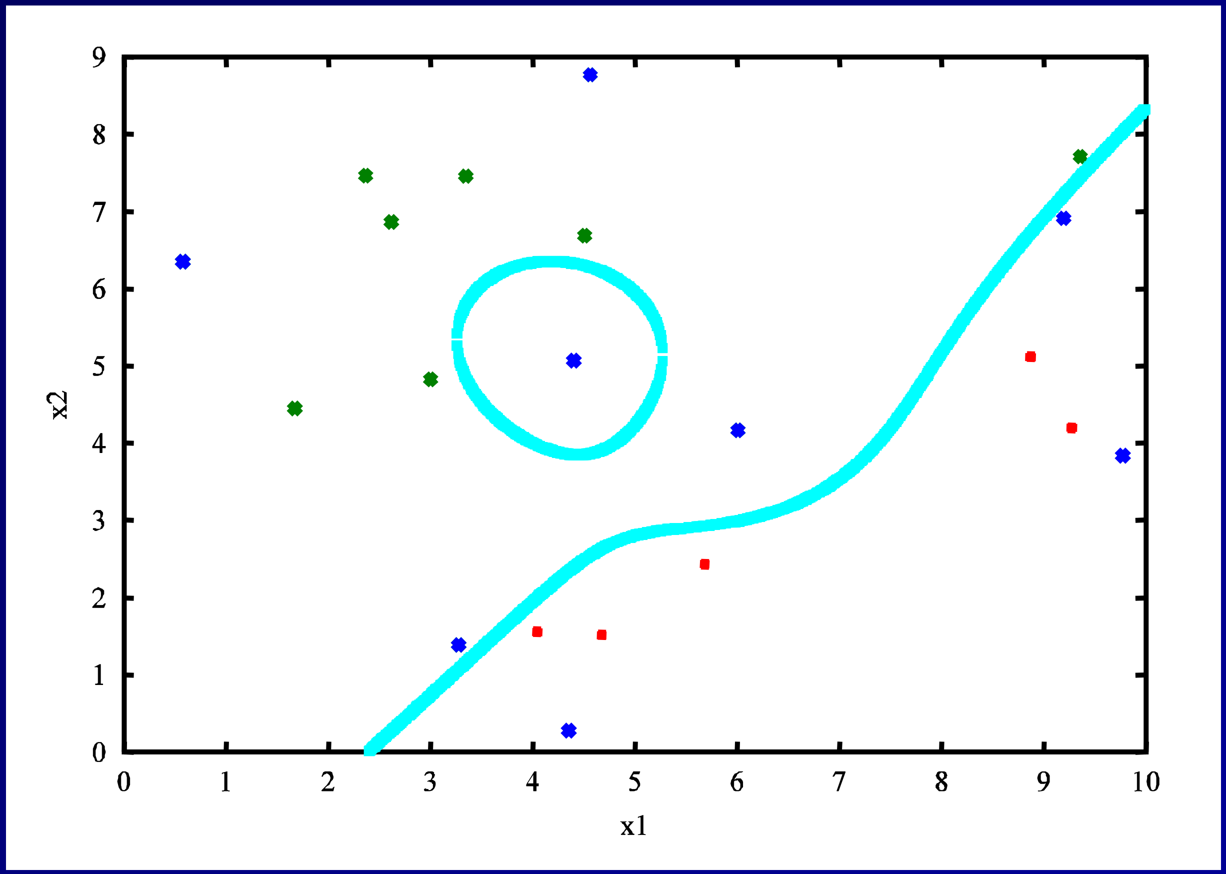
  $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^d$

- Gaussian with scale σ

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

- Polynomials often give strange boundaries. Gaussians generally work well.

Gaussian kernel with $\sigma^2 = 4$



- The gaussian kernel is equivalent to an infinite-dimensional feature space!

Evaluation of SVMs

Criterion	Perc	Logistic	LDA	Trees	Nets	NNbr	SVM
Mixed data	no	no	no	yes	no	no	no
Missing values	no	no	yes	yes	no	somewhat	no
Outliers	no	yes	no	yes	yes	yes	yes
Monotone transformations	no	no	no	yes	somewhat	no	no
Scalability	yes	yes	yes	yes	yes	no	no
Irrelevant inputs	no	no	no	somewhat	no	no	yes*
Linear combinations	yes	yes	yes	no	yes	somewhat	yes
Interpretable	yes	yes	yes	yes	no	no	yes**
Accurate	yes	yes	yes	no	yes	no	yes

* = dot product kernel with absolute value penalty

** = dot product kernel

Support Vector Machines Summary

- Advantages of SVMs
  - variable-sized hypothesis space
  - polynomial-time exact optimization rather than approximate methods
    - unlike decision trees and neural networks
  - Kernels allow very flexible hypotheses
- Disadvantages of SVMs
  - Must choose kernel and kernel parameters: Gaussian, σ
  - Very large problems are computationally intractable
    - quadratic in number of examples
    - problems with more than 20,000 examples are very difficult to solve exactly
  - Batch algorithm

SVMs Unify LTUs and Nearest Neighbor

- With Gaussian kernel
  - compute distance to a set of “support vector” nearest neighbors
  - transform through a gaussian (nearer neighbors get bigger votes)
  - take weighted sum of those distances for each class
  - classify to the class with most votes