CSC2515 Fall 2015
Introduction to Machine Learning

Lecture 9: Support Vector Machines

All lecture slides will be available at
http://www.cs.toronto.edu/~urtasun/courses/CSC2515/CSC2515_Winter15.html

Many of the figures are provided by Chris Bishop
from his textbook: "Pattern Recognition and Machine Learning"

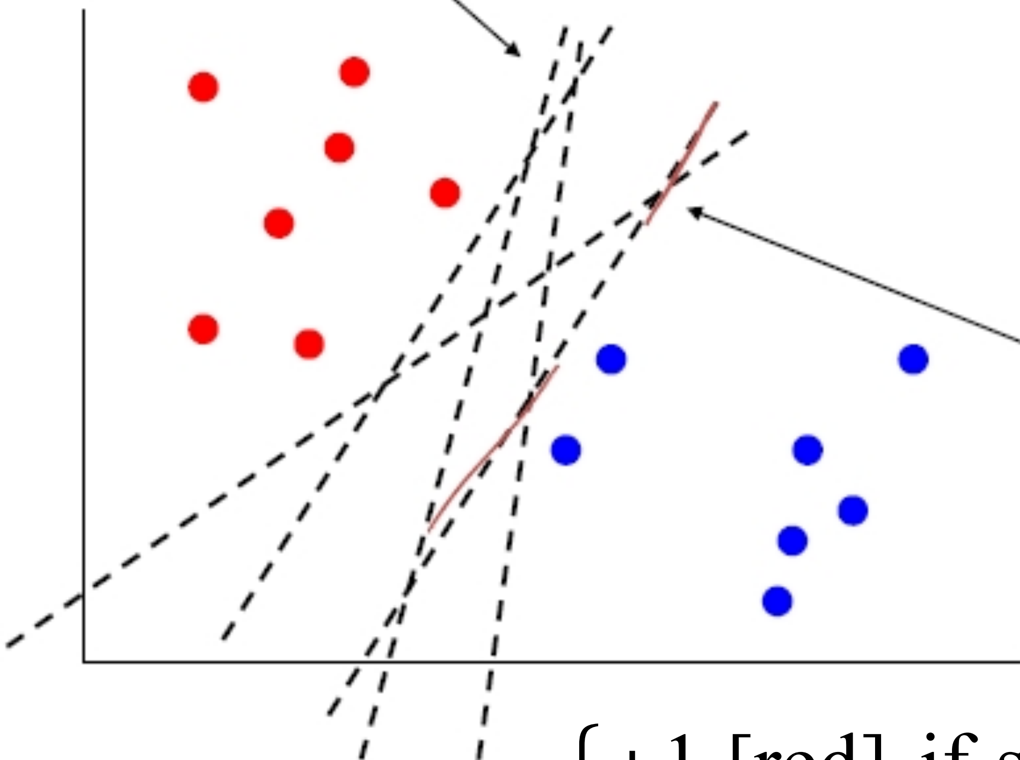
Logistic Regression

Recall logistic regression classifiers

Many more possible classifiers

$$\min_w \sum_i \ln(1+\exp(y^i w^{\mathsf{T}} x^i))$$

Goes over all training points x



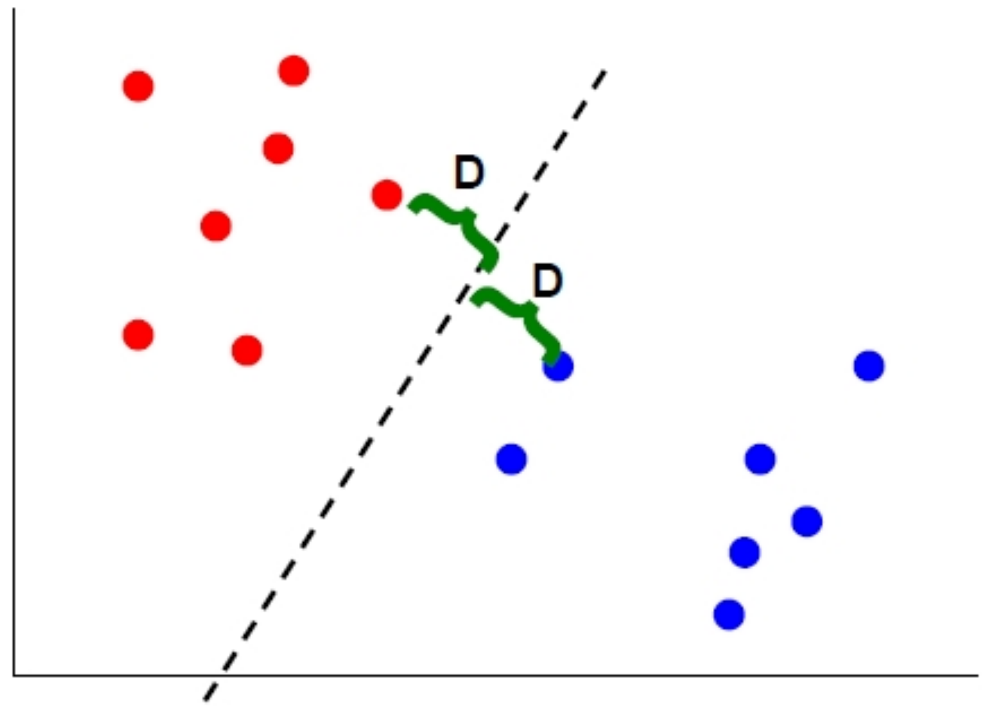
Line closer to the blue nodes since many of them are far away from the boundary

$$y = \begin{cases} +1 \text{ [red] } & \text{if } \operatorname{sign}(\mathbf{w}^T\mathbf{x} + b \geq 0) \\ -1 \text{ [blue] } & \text{if } \operatorname{sign}(\mathbf{w}^T\mathbf{x} + b < 0) \end{cases}$$

Max margin classification

Instead of fitting all the points, focus on boundary points

Aim: learn a boundary that leads to the largest margin (buffer) from points on both sides

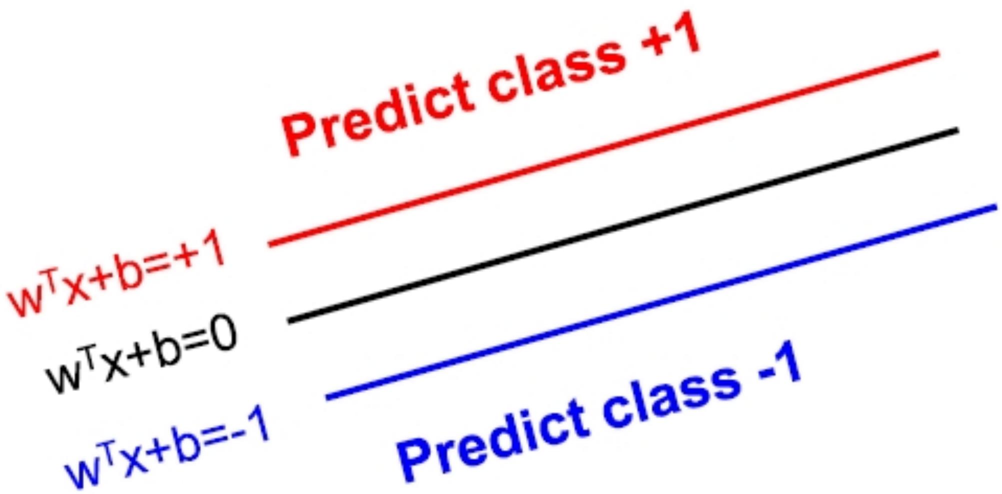


Why: intuition; theoretical support; and works well in practice

Subset of vectors that support (determine boundary) are called the support vectors

Linear SVM

Max margin classifier: inputs in margin are of unknown class



Classify as +1	if	$w^Tx+b \geq 1$
Classify as -1	if	$w^Tx+b \leq -1$
Undefined	if	$-1 < w^Tx+b < 1$

Maximizing the Margin

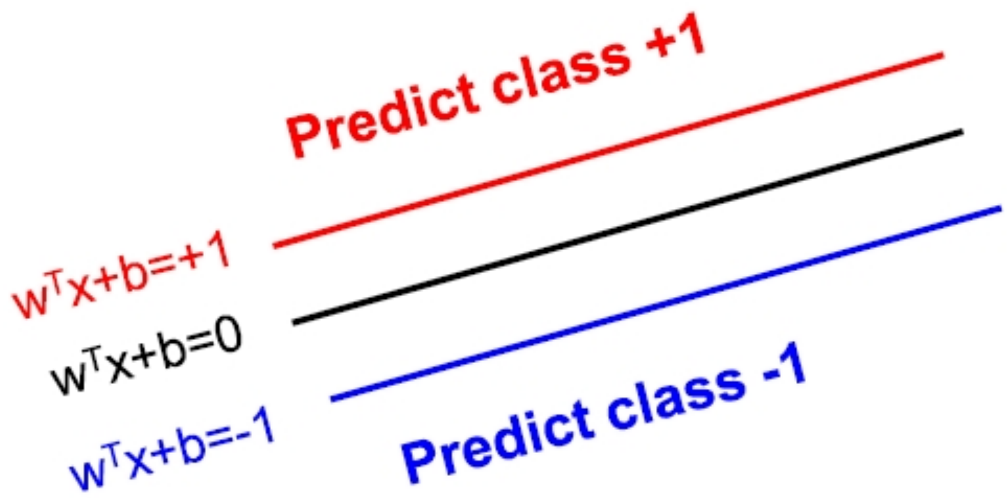
First note that the **w** vector is orthogonal to the +1 plane

  if **u** and **v** are two points on that plane, then $\mathbf{w}^T(\mathbf{u}-\mathbf{v}) = 0$

Same is true for -1 plane

Also: for point **x+** on +1 plane and **x-** nearest point on -1 plane:

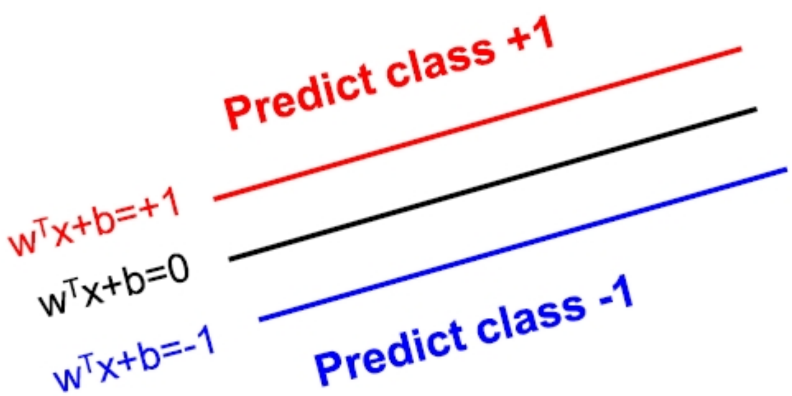
  $\mathbf{x}^+ = \lambda\mathbf{w} + \mathbf{x}^-$



Computing the Margin

Also: for point **x+** on +1 plane and **x-** nearest point on -1 plane:

x+ = λ**w** + **x-**



$$\mathbf{w}^T\mathbf{x}^+ + b = 1$$

$$\mathbf{w}^T(\lambda\mathbf{w} + \mathbf{x}^-) + b = 1$$

$$\mathbf{w}^T\mathbf{x}^- + b + \lambda\mathbf{w}^T\mathbf{w} = 1$$

$$-1 + \lambda\mathbf{w}^T\mathbf{w} = 1$$

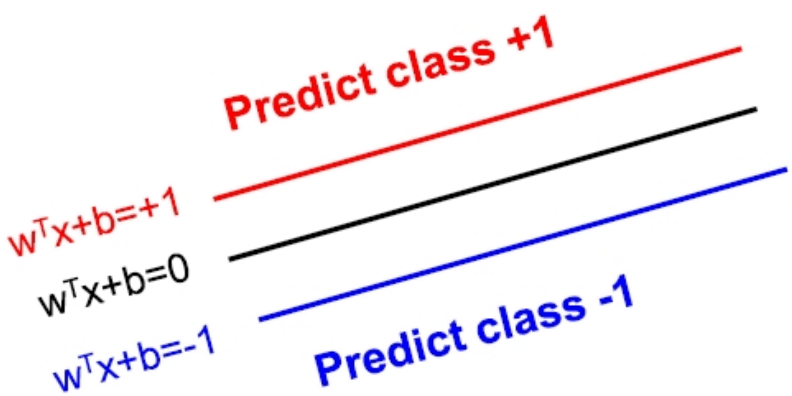
$$\lambda = 2/\mathbf{w}^T\mathbf{w}$$

Computing the Margin

Define the margin M to be the distance between the +1 and -1 planes

We can now express this in terms of w →

to maximize the margin we minimize the length of **w**



$$M = \left\| \mathbf{x}^{+} - \mathbf{x}^{-} \right\|$$

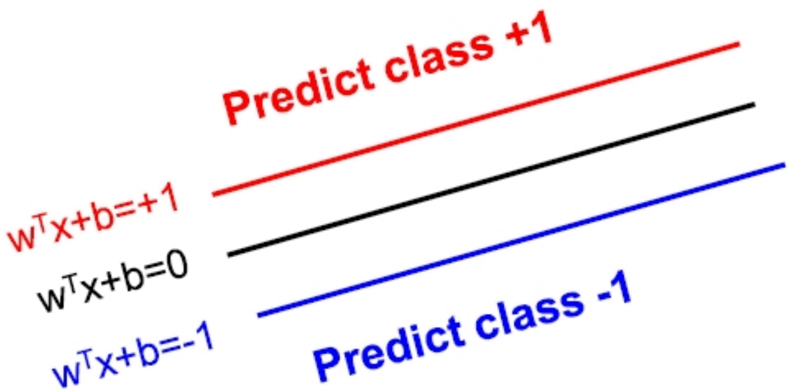
$$= \left\| \lambda \mathbf{w} \right\| = \lambda \sqrt{\mathbf{w}^{T}\mathbf{w}}$$

$$= 2 \frac{\sqrt{\mathbf{w}^{T}\mathbf{w}}}{\mathbf{w}^{T}\mathbf{w}} = \frac{2}{\sqrt{\mathbf{w}^{T}\mathbf{w}}}$$

Learning a Margin-Based Classifier

We can search for the optimal parameters (**w** and b) by finding a solution that:

1. Correctly classifies the training examples: $\{x_i, y_i\}$, i=1,..,n
2. Maximizes the margin (same as minimizing $\mathbf{w}^T\mathbf{w}$)



$$\min \frac{1}{2}\|\mathbf{w}\|^2$$

$$s.t.\ (\mathbf{w}^T\mathbf{x}_i + b)y_i \geq 1 \forall i$$

This is the primal formulation, can optimized via gradient descent, EM, etc.

Apply Lagrange multipliers: formulate equivalent problem

Learning a Linear SVM

Convert the constrained minimization to an unconstrained optimization problem: represent constraints as penalty terms:

$$\min \ \frac{1}{2}\|\mathbf{w}\|^2 + \text{penalty} - \text{term}$$

For data $\{(x_i, y_i)\}$ use the following penalty term:

$$\begin{Bmatrix} 0 & \text{if } (\mathbf{w}^T\mathbf{x}_i + b)y_i \geq 1 \\ \infty & \text{otherwise} \end{Bmatrix} = \max_{\alpha_i \geq 0} \ \alpha_i[1 - (\mathbf{w}^T\mathbf{x}_i + b)y_i]$$

Rewrite the minimization problem:

$$\min_{\mathbf{w},b}\{\frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{n}\max_{\alpha_i \geq 0}\alpha_i[1 - (\mathbf{w}^T\mathbf{x}_i + b)y_i]\}$$

Where $\{\alpha_i\}$ are the Lagrange multipliers

$$= \min_{\mathbf{w},b}\max_{\alpha_i \geq 0}\{\frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{n}\alpha_i[1 - (\mathbf{w}^T\mathbf{x}_i + b)y_i]\}$$

Solution to Linear SVM

Swap the 'max' and 'min':

$$\max_{\alpha_i \geq 0} \min_{\mathbf{w},b} \{\frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{n} \alpha_i [1-(\mathbf{w}^T\mathbf{x}_i + b)y_i]\}$$

$$= \max_{\alpha_i \geq 0} \min_{\mathbf{w},b} J(\mathbf{w}, b; \alpha)$$

First minimize J() w.r.t. {**w**,b} for any fixed setting of the Lagrange multipliers:

$$\frac{\partial}{\partial \mathbf{w}} J(\mathbf{w}, b; \alpha) = \mathbf{w} - \sum_{i=1}^{n} \alpha_i \mathbf{x}_i y_i = 0$$

$$\frac{\partial}{\partial b} J(\mathbf{w}, b; \alpha) = -\sum_{i=1}^{n} \alpha_i y_i = 0$$

Then substitute back to get final optimization:

$$L = \max_{\alpha_i \geq 0} \{\sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j)\}$$

Summary of Linear SVM

- Binary and linear separable classification
- Linear classifier with maximal margin
- Training SVM by maximizing

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

- Subject to $\alpha_i \geq 0; \sum_{i=1}^{n} \alpha_i \mathbf{x}_i = 0$
- Weights: $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$
- Only a small subset of α_i's will be nonzero, and the corresponding x_i's are the support vectors **S**
- Prediction on a new example:

$$y = \text{sign}[b + \mathbf{x} \cdot (\sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i)] = \text{sign}[b + \mathbf{x} \cdot (\sum_{i \in \mathbf{S}} y_i \alpha_i \mathbf{x}_i)]$$

What if data is not linearly separable?

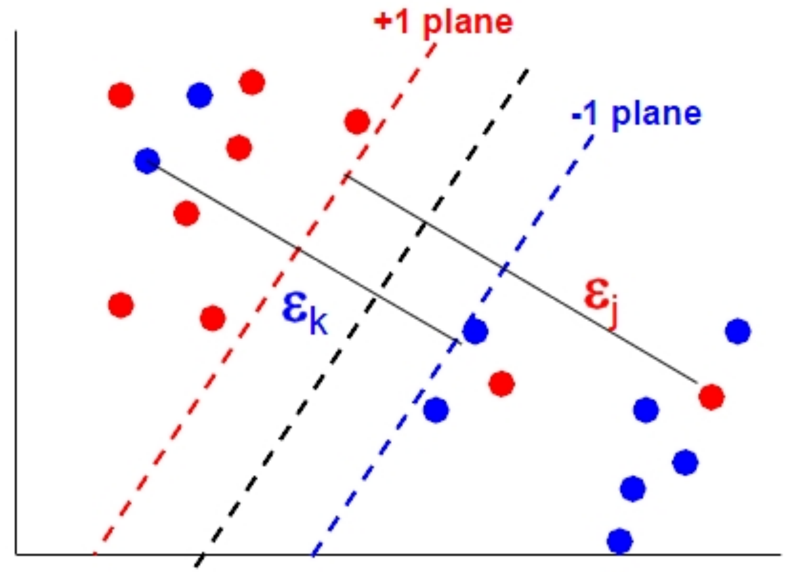
- Introduce slack variables ξ_i

$$\min\left[\frac{1}{2}\|\mathbf{w}\|^2 + \lambda\sum_{i=1}^{n}\xi_i\right]$$

subject to constraints (for all *i*):

$$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$



- Example lies on wrong side of hyperplane: $\xi_i > 1 \Rightarrow \sum_i \xi_i$ is upper bound on number of training errors

- Λ trades off training error versus model complexity

- This is known as the soft-margin extension

Non-linear decision boundaries

- Note that both the learning objective and the decision function depend only on dot products between patterns

$$L = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j) \qquad y = \text{sign}[b + \mathbf{x} \cdot (\sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i)]$$

- How to form non-linear decision boundaries in input space?

- Basic idea:

1. Map data into feature space $\mathbf{x} \to \phi(\mathbf{x})$

2. Replace dot products between inputs with feature points

$$\mathbf{x}_i \cdot \mathbf{x}_j \to \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$$

3. Find linear decision boundary in feature space

- Problem: what is a good feature function $\phi(\mathbf{x})$?

Kernel Trick

- Kernel trick: dot-products in feature space can be computed as a kernel function

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

- Idea: work directly on **x**, avoid having to compute ϕ(**x**)

- Example:

$$K(\mathbf{a},\mathbf{b}) = (\mathbf{a}\cdot\mathbf{b})^3 = ((a_1, a_2)\cdot(b_1, b_2))^3$$
$$= (a_1 b_1 + a_2 b_2)^3$$
$$= a_1{}^3 b_1{}^3 + 3a_1{}^2 b_1{}^2 a_2 b_2 + 3a_1 b_1 a_2{}^2 b_2{}^2 + a_2{}^3 b_2{}^3$$
$$= (a_1{}^3, \sqrt{3}a_1{}^2 a_2, \sqrt{3}a_1 a_2{}^2, a_2{}^3)\cdot(b_1{}^3, \sqrt{3}b_1{}^2 b_2, \sqrt{3}b_1 b_2{}^2, b_2{}^3)$$
$$= \phi(\mathbf{a})\cdot\phi(\mathbf{b})$$

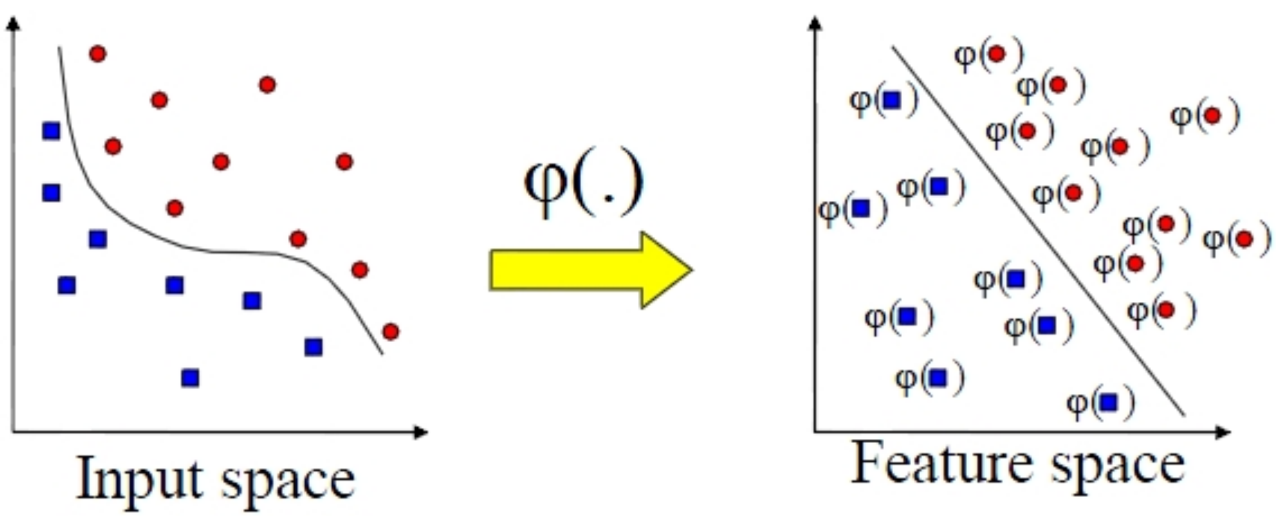
Input transformation

Mapping to a feature space can produce problems:

- High computational burden due to high dimensionality
- Many more parameters

SVM solves these two issues simultaneously

- Kernel trick produces efficient classification
- Dual formulation only assigns parameters to samples, not features



Kernels

Examples of kernels (kernels measure similarity):

1. Polynomial $K(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2 + 1)^2$
2. Gaussian $K(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\|\mathbf{x}_1 - \mathbf{x}_2\|^2 / 2\sigma^2)$
3. Sigmoid $K(\mathbf{x}_1, \mathbf{x}_2) = \tanh(\kappa(\mathbf{x}_1 \cdot \mathbf{x}_2) + a)$

Each kernel computation corresponds to dot product calculation for particular mapping $\phi(x)$: implicitly maps to high-dimensional space

Why is this useful?

1. Rewrite training examples using more complex features
2. Dataset not linearly separable in original space may be linearly separable in higher dimensional space

Classification with non-linear SVMs

Non-linear SVM using kernel function K():

$$L_K = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$$

Maximize L_K w.r.t. {α}, under constraints α≥0

Unlike linear SVM, cannot express w as linear combination of support vectors – now must retain the support vectors to classify new examples

Final decision function:

$$y = \mathrm{sign}[b + \sum_{i=1}^{n} y_i \alpha_i K(\mathbf{x}, \mathbf{x}_i)]$$

Kernel Functions

Mercer's Theorem (1909): any reasonable kernel corresponds to some feature space

Reasonable means that the Gram matrix is positive definite

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

Feature space can be very large, e.g., polynomial kernel $(1+\mathbf{x}_i + \mathbf{x}_j)^d$ corresponds to feature space exponential in d

Linear separators in these super high-dim spaces correspond to highly nonlinear decision boundaries in input space

Summary

Advantages:

- Kernels allow very flexible hypotheses
- Poly-time exact optimization methods rather than approximate methods
- Soft-margin extension permits mis-classified examples
- Variable-sized hypothesis space
- Excellent results (1.1% error rate on handwritten digits vs. LeNet's 0.9%)

Disadvantages:

- Must choose kernel parameters
- Very large problems computationally intractable
- Batch algorithm

Kernelizing

A popular way to make an algorithm more powerful is to develop a kernelized version of it

- We can rewrite a lot of algorithms to be defined only in terms of inner product
- For example: k-nearest neighbors

$$\mathbf{z} = \varphi(\mathbf{x})$$

$$(\mathbf{z}_i - \mathbf{z}_j)^2 = K(\mathbf{x}_i, \mathbf{x}_i) + K(\mathbf{x}_j, \mathbf{x}_j) - 2K(\mathbf{x}_i, \mathbf{x}_j)$$

More Summary

Software:

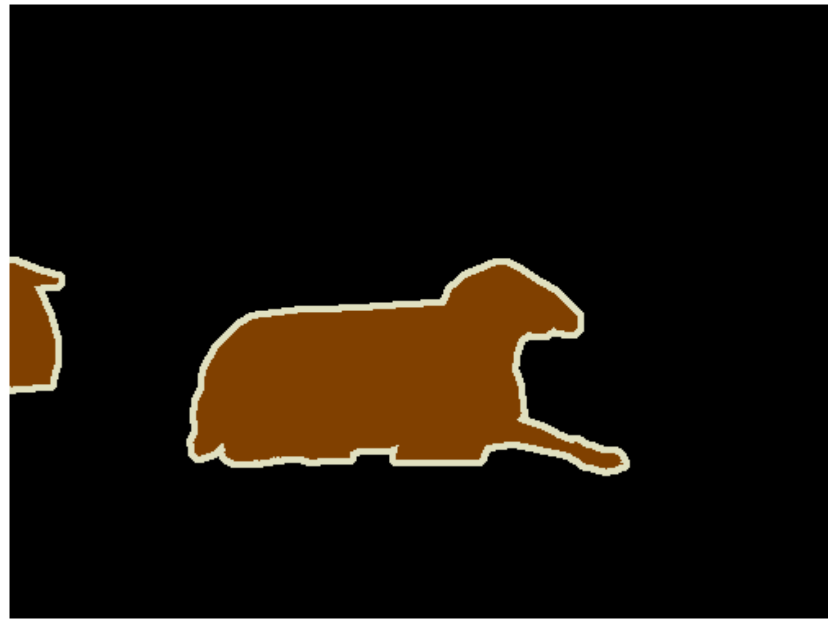
- A list of SVM implementations can be found at http://www.kernel-machines.org/software.html
- Some implementations (such as LIBSVM) can handle multi-class classification
- SVMLight is among the earliest implementations
- Several Matlab toolboxes for SVM are also available

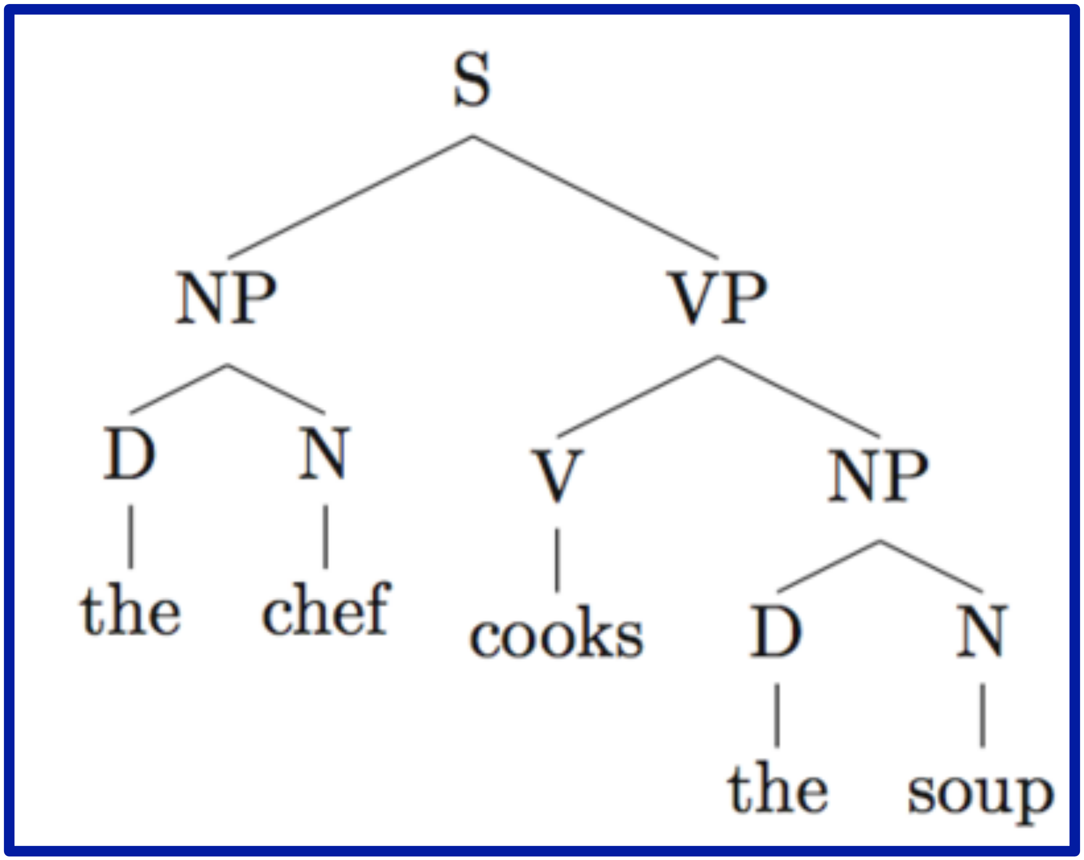
Key points:

- Difference between logistic regression and SVMs
- Maximum margin principle
- Target function for SVMs
- Slack variables for mis-classified points
- Kernel trick allows non-linear generalizations

Structured Output Problems

- Output is multi-dimensional, with dependencies between the dimensions
- Examples:
  - natural language sentence → annotated parse tree
  - Image → labeled pixels
- Aim: produce best structured output on test examples


S
NP
VP
D
N
V
NP
the
chef
cooks
D
N
the
soup

Structured Output SVM

- Training set of N examples
- Use analogous loss function to single-output SVM

$$\min_{\mathbf{w}} \left\| \mathbf{w}^2 \right\|$$

$$s.t. \quad \forall n, \mathbf{y} \qquad \mathbf{w}\Psi(\mathbf{x}^{(n)}, \mathbf{y}) - \mathbf{w}\Psi(\mathbf{x}^{(n)}, \mathbf{y}^{(n)}) \geq 1$$