

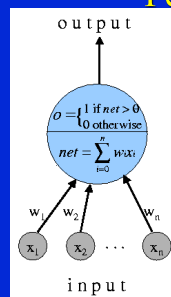


Linear Discriminant Functions and Support Vector Machines

Biometrics
CSE 190
Lecture 11

CSE190, Winter 2011

Perceptron



Linear, threshold units

x_i : inputs

w_i : weights

θ : threshold

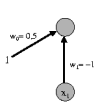
$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_1 x_1 + \dots + w_n x_n > \theta \\ -1 & \text{otherwise.} \end{cases}$$

How powerful is a perceptron?

Threshold = 0

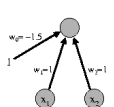
Inverter

input x_1	output
0	1
1	0



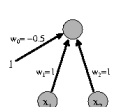
Boolean AND

input x_1	input x_2	output
0	0	0
0	1	0
1	0	0
1	1	1



Boolean OR

input x_1	input x_2	output
0	0	0
0	1	1
1	0	1
1	1	1



Boolean XOR

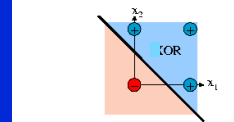
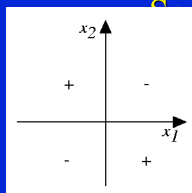
input x_1	input x_2	output
0	0	0
0	1	1
1	0	1
1	1	0



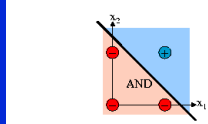
Concept Space & Linear

Generalization

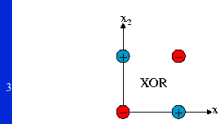
Linear Separability



Linear Separability



Linear Separability



Training Perceptron

Perceptron Training Rule

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta (t - o) x_i$$

Where:

- t is target value
- o is perceptron output
- η is small constant (e.g., .1) called *learning rate*

Converges, if...

- ... training data linearly separable
- ... step size η sufficiently small
- ... no "hidden" units

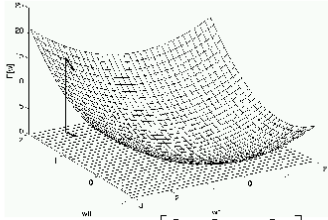
Gradient Descent

- Learn w_i 's that minimize squared error

$$E[\vec{w}] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

D = training data

Gradient Descent



Gradient: $\nabla E[\vec{w}] = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$

Training rule: $\Delta \vec{w} = -\eta \nabla E[\vec{w}]$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

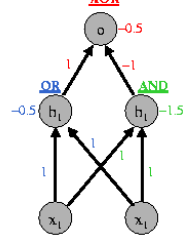
Summary: Single Layer Networks

- Variety of update rules
 - Multiplicative $\Delta w_i = \mathbf{R}(\mathbf{t}_d - \mathbf{o}_d) \mathbf{x}_{id}$
 - Additive
- Batch and incremental algorithms
- Various convergence and efficiency conditions
- There are other ways to learn linear functions
 - Linear Programming (general purpose)
 - Probabilistic Classifiers (some assumption)
- Although simple and restrictive -- linear predictors perform very well on many realistic problems
- However, the representational restriction is limiting in many applications

Increasing Expressiveness: Multi-Layer Neural Networks

Boolean XOR

input x1	input x2	output
0	0	0
0	1	1
1	0	1
1	1	0



2-layer Neural Net

Support Vector Machines

Courtesy of University of Texas at Austin

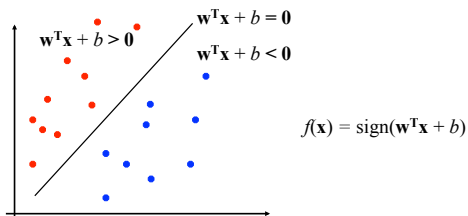
Reading:

- Duda, Hart & Stork, Sec. 5.11
- "A Tutorial on Support Vector Machines for Pattern Recognition," Christopher Burges

9

Perceptron Revisited: Linear Separators

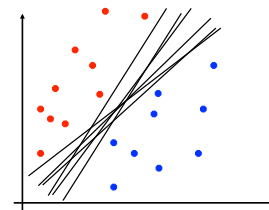
- Binary classification can be viewed as the task of separating classes in feature space:



10

Linear Separators

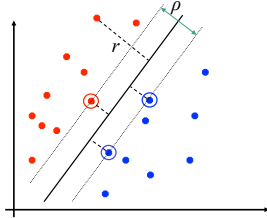
- Which of the linear separators is optimal?



11

Classification Margin

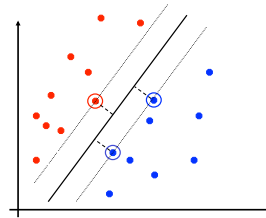
- Distance from example $\mathbf{x}_i$ to the separator is $r = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$
- Training examples closest to the hyperplane are **support vectors**.
- Margin** ρ of the separator is the distance from the separator to support vectors.



12

Maximum Margin Classification

- Maximizing the margin is good according to intuition and PAC (Probably Approximately Correct) theory.
- Implies that only support vectors matter; other training examples are ignorable.



- Solved as a Quadratic Programming Problem

13

Linear SVM Mathematically

- Let training set $\{(\mathbf{x}_i, y_i)\}_{i=1..n}$, $\mathbf{x}_i \in \mathbb{R}^d$, $y_i \in \{-1, 1\}$ be separated by a hyperplane with margin ρ . Then for each training example $(\mathbf{x}_i, y_i)$:

$$\begin{aligned} \mathbf{w}^T \mathbf{x}_i + b &\leq -\rho/2 & \text{if } y_i = -1 \\ \mathbf{w}^T \mathbf{x}_i + b &\geq \rho/2 & \text{if } y_i = 1 \end{aligned} \quad \Leftrightarrow \quad y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq \rho/2$$

- For every support vector $\mathbf{x}_s$ the above inequality is an equality. After rescaling $\mathbf{w}$ and b by $\rho/2$ in the equality, we obtain that distance between each $\mathbf{x}_s$ and the hyperplane is $r = \frac{y_s(\mathbf{w}^T \mathbf{x}_s + b)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$
- Then the margin can be expressed through (rescaled) $\mathbf{w}$ and b as:

$$\rho = 2r = \frac{2}{\|\mathbf{w}\|}$$

14

Linear SVMs Mathematically (cont.)

- Then we can formulate the *quadratic optimization problem*:

$$\begin{aligned} &\text{Find } \mathbf{w} \text{ and } b \text{ such that} \\ &\rho = \frac{2}{\|\mathbf{w}\|} \text{ is maximized} \\ &\text{and for all } (\mathbf{x}_i, y_i), i=1..n : y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \end{aligned}$$

Which can be reformulated as:

$$\begin{aligned} &\text{Find } \mathbf{w} \text{ and } b \text{ such that} \\ &\Phi(\mathbf{w}) = \|\mathbf{w}\|^2 = \mathbf{w}^T \mathbf{w} \text{ is minimized} \\ &\text{and for all } (\mathbf{x}_i, y_i), i=1..n : y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \end{aligned}$$

15

Solving the Optimization Problem

$$\begin{aligned} &\text{Find } \mathbf{w} \text{ and } b \text{ such that} \\ &\Phi(\mathbf{w}) = \mathbf{w}^T \mathbf{w} \text{ is minimized} \\ &\text{and for all } (\mathbf{x}_i, y_i), i=1..n : y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \end{aligned}$$

- Need to optimize a *quadratic* function subject to *linear* constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems for which several (non-trivial) algorithms exist.
- The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every inequality constraint in the primal (original) problem:

$$\begin{aligned} &\text{Find } \alpha_1 \dots \alpha_n \text{ such that} \\ &\mathbf{Q}(\boldsymbol{\alpha}) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \text{ is maximized and} \\ &(1) \sum \alpha_i y_i = 0 \\ &(2) \alpha_i \geq 0 \text{ for all } \alpha_i \end{aligned}$$

16

The Optimization Problem Solution

- Given a solution $\alpha_1 \dots \alpha_n$ to the dual problem, solution to the primal is:

$$\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i \quad b = y_k - \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x}_k \quad \text{for any } \alpha_k > 0$$

- Each non-zero α_i indicates that corresponding $\mathbf{x}_i$ is a support vector.
- Then the classifying function is (note that we don't need $\mathbf{w}$ explicitly):

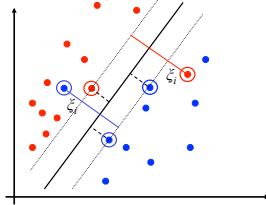
$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$$

- Notice that it relies on an *inner product* between the test point $\mathbf{x}$ and the support vectors $\mathbf{x}_i$ – we will return to this later.
- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x}_i^T \mathbf{x}_j$ between all training points.

17

Soft Margin Classification

- What if the training set is not linearly separable?
- Slack variables ξ_i can be added to allow misclassification of difficult or noisy examples, resulting margin called *soft*.



18

Soft Margin Classification Mathematically

- The old formulation:

Find $\mathbf{w}$ and b such that
 $\Phi(\mathbf{w}) = \mathbf{w}^T \mathbf{w}$ is minimized
 and for all $(\mathbf{x}_i, y_i), i=1..n$: $y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$

- Modified formulation incorporates slack variables:

Find $\mathbf{w}$ and b such that
 $\Phi(\mathbf{w}) = \mathbf{w}^T \mathbf{w} + C \sum \xi_i$ is minimized
 and for all $(\mathbf{x}_i, y_i), i=1..n$: $y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$

- Parameter C can be viewed as a way to control overfitting: it "trades off" the relative importance of maximizing the margin and fitting the training data.

19

Soft Margin Classification – Solution

- Dual problem is identical to separable case (would *not* be identical if the 2-norm penalty for slack variables $C \sum \xi_i^2$ was used in primal objective, we would need additional Lagrange multipliers for slack variables):

Find $a_1 \dots a_N$ such that
 $Q(\mathbf{a}) = \sum a_i - \frac{1}{2} \sum \sum a_i a_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and
 (1) $\sum a_i y_i = 0$
 (2) $0 \leq a_i \leq C$ for all a_i

- Again, $\mathbf{x}_i$ with non-zero a_i will be support vectors.
- Solution to the dual problem is:

$$\mathbf{w} = \sum a_i y_i \mathbf{x}_i$$

$$b = y_k (1 - \xi_k) - \sum a_i y_i \mathbf{x}_i^T \mathbf{x}_k \quad \text{for any } k \text{ s.t. } a_k > 0$$

Again, we don't need to compute $\mathbf{w}$ explicitly for classification:

$$f(\mathbf{x}) = \sum a_i y_i \mathbf{x}_i^T \mathbf{x} + b$$

20

Linear SVMs: Overview

- The classifier is a *separating hyperplane*.
- Most "important" training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points $\mathbf{x}_i$ are support vectors with non-zero Lagrangian multipliers a_i .
- Both in the dual formulation of the problem and in the solution training points appear only inside inner products:

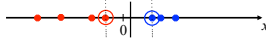
Find $a_1 \dots a_N$ such that
 $Q(\mathbf{a}) = \sum a_i - \frac{1}{2} \sum \sum a_i a_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and
 (1) $\sum a_i y_i = 0$
 (2) $0 \leq a_i \leq C$ for all a_i

$$f(\mathbf{x}) = \sum a_i y_i \mathbf{x}_i^T \mathbf{x} + b$$

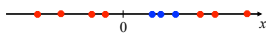
21

Non-linear SVMs

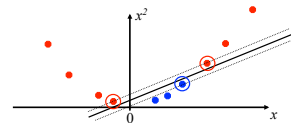
- Datasets that are linearly separable with some noise work out great:



- But what are we going to do if the dataset is just too hard?



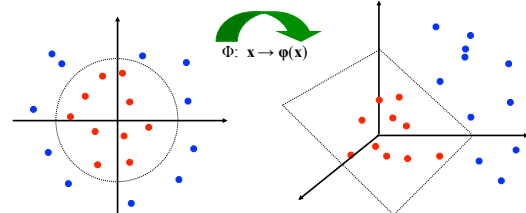
- How about... mapping data to a higher-dimensional space:



22

Non-linear SVMs: Feature spaces

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



Video Example:

<http://www.youtube.com/watch?v=3liCbRZPrZA>

23

The “Kernel Trick”

- The linear classifier relies on inner product between vectors $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- If every datapoint is mapped into high-dimensional space via some transformation $\Phi: \mathbf{x} \rightarrow \phi(\mathbf{x})$, the inner product becomes:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$
- A *kernel function* is a function that is equivalent to an inner product in some feature space.
- Example:
2-dimensional vectors $\mathbf{x} = [x_1 \ x_2]$; let $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$.
Need to show that $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$:

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= (1 + \mathbf{x}_i^T \mathbf{x}_j)^2 = 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2} \\ &= [1 \ x_{i1}^2 \ \sqrt{2} x_{i1} x_{i2} \ x_{i2}^2 \ \sqrt{2} x_{i1} \ \sqrt{2} x_{i2}]^T [1 \ x_{j1}^2 \ \sqrt{2} x_{j1} x_{j2} \ x_{j2}^2 \ \sqrt{2} x_{j1} \ \sqrt{2} x_{j2}] \\ &= \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j), \text{ where } \phi(\mathbf{x}) = [1 \ x_1^2 \ \sqrt{2} x_1 x_2 \ x_2^2 \ \sqrt{2} x_1 \ \sqrt{2} x_2] \end{aligned}$$
- Thus, a kernel function *implicitly* maps data to a high-dimensional space (without the need to compute each $\phi(\mathbf{x})$ explicitly).

24

What Functions are Kernels?

- For some functions $K(\mathbf{x}_i, \mathbf{x}_j)$ checking that $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ can be cumbersome.
- Mercer's theorem:
Every semi-positive definite symmetric function is a kernel
- Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:

$$K = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & K(\mathbf{x}_1, \mathbf{x}_3) & \dots & K(\mathbf{x}_1, \mathbf{x}_n) \\ K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) & K(\mathbf{x}_2, \mathbf{x}_3) & \dots & K(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ K(\mathbf{x}_n, \mathbf{x}_1) & K(\mathbf{x}_n, \mathbf{x}_2) & K(\mathbf{x}_n, \mathbf{x}_3) & \dots & K(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}$$

25

Examples of Kernel Functions

- Linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
– Mapping $\Phi: \mathbf{x} \rightarrow \phi(\mathbf{x})$, where $\phi(\mathbf{x})$ is $\mathbf{x}$ itself
- Polynomial of power p : $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$
– Mapping $\Phi: \mathbf{x} \rightarrow \phi(\mathbf{x})$, where $\phi(\mathbf{x})$ has $\binom{d+p}{p}$ dimensions

$$\phi(\mathbf{x}) = \left[1, x_1, x_2, \dots, x_1^p, x_2^p, \dots, x_1^{p-1} x_2, x_1 x_2^{p-1}, \dots, x_1^p x_2^p \right]^T$$
- Gaussian (radial-basis function): $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}}$
– Mapping $\Phi: \mathbf{x} \rightarrow \phi(\mathbf{x})$, where $\phi(\mathbf{x})$ is *infinite-dimensional*: every point is mapped to a *function* (a Gaussian); combination of functions for support vectors is the separator.
- Higher-dimensional space still has *intrinsic* dimensionality d (the mapping is not *onto*), but linear separators in it correspond to *non-linear* separators in original space.

26

Non-linear SVMs Mathematically

- Dual problem formulation:
Find $\alpha_1, \dots, \alpha_n$ such that

$$\mathbf{Q}(\mathbf{a}) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$
 is maximized and
 (1) $\sum \alpha_i y_i = 0$
 (2) $\alpha_i \geq 0$ for all α_i
- The solution is:

$$f(\mathbf{x}) = \sum \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$
- Optimization techniques for finding α_i 's remain the same!

27

SVM applications

- SVMs were originally proposed by Boser, Guyon and Vapnik in 1992 and gained increasing popularity in late 1990s.
- SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.
- SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.
- SVM techniques have been extended to a number of tasks such as regression [Vapnik *et al.* '97], principal component analysis [Schölkopf *et al.* '99], etc.
- Tuning SVMs remains a black art: selecting a specific kernel and parameters is usually done in a try-and-see manner.
- Implementations: See libsvm, svmlight, and others

28