

Data Mining

Practical Machine Learning Tools and Techniques

Slides for Section 6.3

Extending linear classification

- Linear classifiers can't model nonlinear class boundaries
- Simple trick:
 - Map attributes into new space consisting of combinations of attribute values
 - E.g.: all products of n factors that can be constructed from the attributes
- Example with two attributes and $n = 3$:

$$x = w_1 a_1^3 + w_2 a_1^2 a_2 + w_3 a_1 a_2^2 + w_4 a_2^3$$

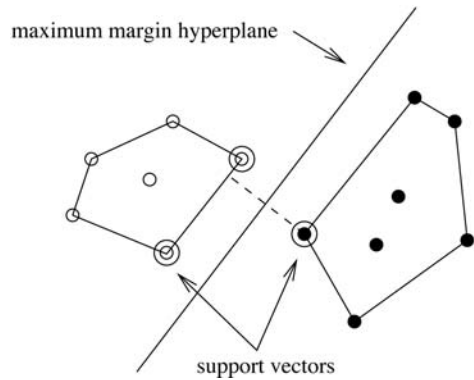
Problems with this approach

- 1st problem: speed
 - 10 attributes, and $n = 5 \Rightarrow >2000$ coefficients
 - Use linear regression with attribute selection
 - Run time is cubic in number of attributes
- 2nd problem: overfitting
 - Number of coefficients is large relative to the number of training instances
 - *Curse of dimensionality* kicks in

Support vector machines

- *Support vector machines* are algorithms for learning linear classifiers
- Resilient to overfitting because they learn a particular linear decision boundary:
 - The *maximum margin hyperplane*
- Fast in the nonlinear case
 - Use a mathematical trick to avoid creating “pseudo-attributes”
 - The nonlinear space is created implicitly

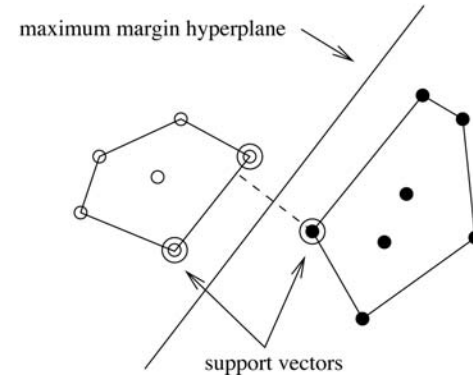
The maximum margin hyperplane



- The instances closest to the maximum margin hyperplane are called *support vectors*

Support vectors

- The support vectors define the maximum margin hyperplane
 - All other instances can be deleted without changing its position and orientation



- This means the hyperplane $x = w_0 + w_1 a_1 + w_2 a_2$ can be written as $x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i \vec{a}(i) \cdot \vec{a}$

Finding support vectors

$$x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i \vec{a}(i) \cdot \vec{a}$$

- Support vector: training instance for which $\alpha_i > 0$
- Determine α_i and b ?—
 - A *constrained quadratic optimization* problem
 - Off-the-shelf tools for solving these problems
 - However, special-purpose algorithms are faster
 - Example: Platt's *sequential minimal optimization* algorithm (implemented in WEKA)
- Note: all this assumes separable data!

Nonlinear SVMs

- “Pseudo attributes” represent attribute combinations
- Overfitting not a problem because the maximum margin hyperplane is stable
 - There are usually few support vectors relative to the size of the training set
- Computation time still an issue
 - Each time the dot product is computed, all the “pseudo attributes” must be included

A mathematical trick

- Avoid computing the “pseudo attributes”
- Compute the dot product before doing the nonlinear mapping
- Example:
$$x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i (\vec{a}(i) \cdot \vec{a})^n$$
- Corresponds to a map into the instance space spanned by all products of n attributes

Other kernel functions

- Mapping is called a “kernel function”
- Polynomial kernel $x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i (\vec{a}(i) \cdot \vec{a})^n$
- We can use others: $x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i K(\vec{a}(i) \cdot \vec{a})$
- Only requirement: $K(\vec{x}_i, \vec{x}_j) = \phi(\vec{x}_i) \cdot \phi(\vec{x}_j)$
- Examples:
$$K(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j + 1)^d$$
$$K(\vec{x}_i, \vec{x}_j) = \exp\left(\frac{-(\vec{x}_i - \vec{x}_j)^2}{2\sigma^2}\right)$$

*

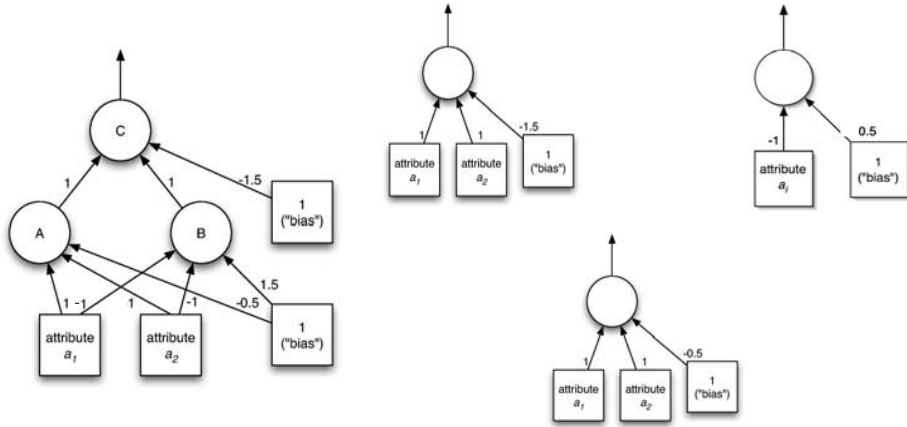
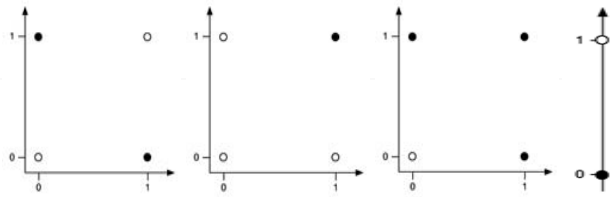
Applications

- Machine vision: e.g. face identification
 - Outperforms alternative approaches (1.5% error)
- Handwritten digit recognition: USPS data
 - Comparable to best alternative (0.8% error)
- Bioinformatics: e.g. prediction of protein secondary structure
- Text classification
- Can modify SVM technique for numeric prediction problems

Multilayer perceptrons

- Using kernels is only one way to build nonlinear classifier based on perceptrons
- Can create network of perceptrons to approximate arbitrary target concepts
- *Multilayer perceptron* is an example of an artificial neural network
 - Consists of: input layer, hidden layer(s), and output layer
- Structure of MLP is usually found by experimentation
- Parameters can be found using *backpropagation*

Examples



Backpropagation

- How to learn weights given network structure?
 - Cannot simply use perceptron learning rule because we have hidden layer(s)
 - Function we are trying to minimize: error
 - Can use a general function minimization technique called *gradient descent*

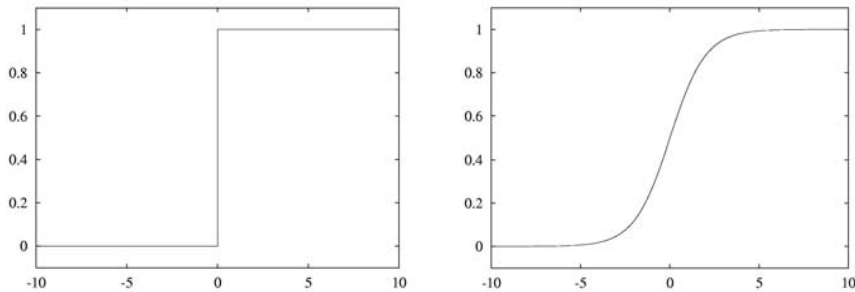
- Need differentiable *activation function*: use *sigmoid function* instead of threshold function

$$f(x) = \frac{1}{1 + \exp(-x)}$$

- Need differentiable error function: can't use zero-one loss, but can use squared error

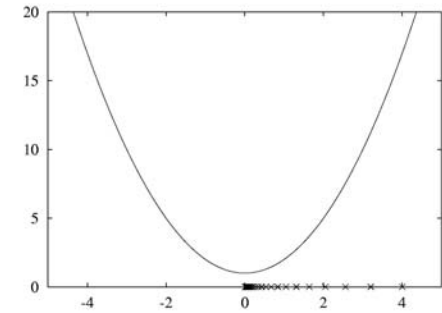
$$E = \frac{1}{2}(y - f(x))^2$$

The two activation functions



Gradient descent example

- Function: $x^2 + 1$
- Derivative: $2x$
- Learning rate: 0.1
- Start value: 4



Can only find a local minimum!