SUPPORT VECTOR MACHINES

Today

- Reading
  - AIMA 18.9 (SVMs)
- Goals
  - Finish Backpropagation
  - Support vector machines

Backpropagation

1. Begin with randomly initialized weights
2. Apply the neural network to each training example (each pass through examples is called an epoch)
3. If it misclassifies an example **modify the weights**
4. Continue until the neural network classifies all training examples correctly

(Derive gradient-descent update rule)

Backpropagation

```
function BACK-PROP-LEARNING(examples, network) returns a neural network
  inputs: examples, a set of examples, each with input vector x and output vector y
          network, a multilayer network with L layers, weights w_{i,j}, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
     for each weight w_{i,j} in network do
        w_{i,j} ← a small random number
     for each example (x, y) in examples do
        /* Propagate the inputs forward to compute the outputs */
        for each node i in the input layer do
           a_i ← x_i
        for ℓ = 2 to L do
           for each node j in layer ℓ do
              in_j ← Σ_i w_{i,j} a_i
              a_j ← g(in_j)
        /* Propagate deltas backward from output layer to input layer */
        for each node j in the output layer do
           Δ[j] ← g′(in_j) × (y_j − a_j)
        for ℓ = L − 1 to 1 do
           for each node i in layer ℓ do
              Δ[i] ← g′(in_i) Σ_j w_{i,j} Δ[j]
        /* Update every weight in network using deltas */
        for each weight w_{i,j} in network do
           w_{i,j} ← w_{i,j} + α × a_i × Δ[j]
  until some stopping criterion is satisfied
  return network
```

Figure 18.24 The back-propagation algorithm for learning in multilayer networks.

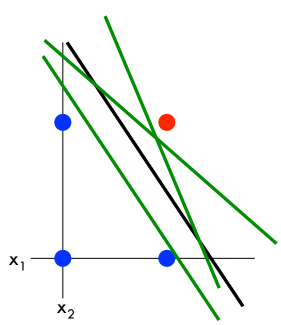
Support Vector Machines (SVMs)

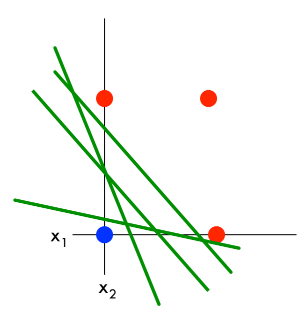
- SVMs are probably the most popular off-the-shelf classifier!
- Software Packages
  - LIBSVM (LIBLINEAR) – on the Resources page
  - SVM-Light

Which is the best decision boundary?

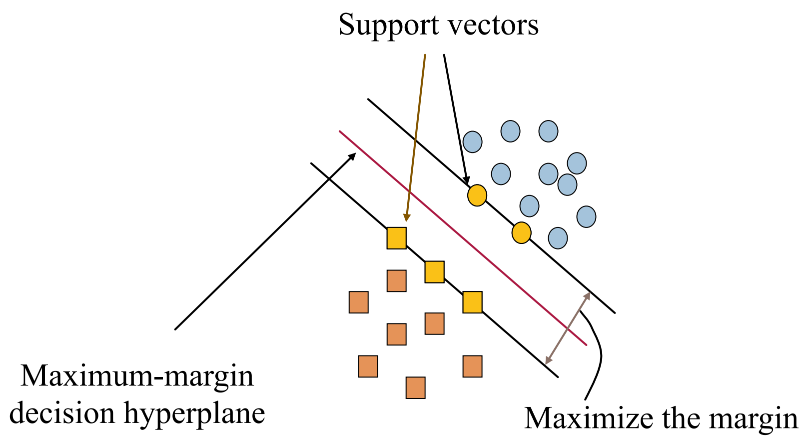
x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	x_1 or x_2
0	0	0
0	1	1
1	0	1
1	1	1

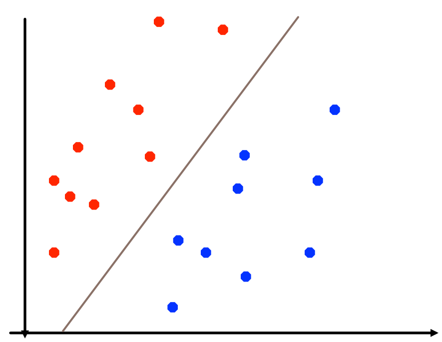




Support Vector Machines



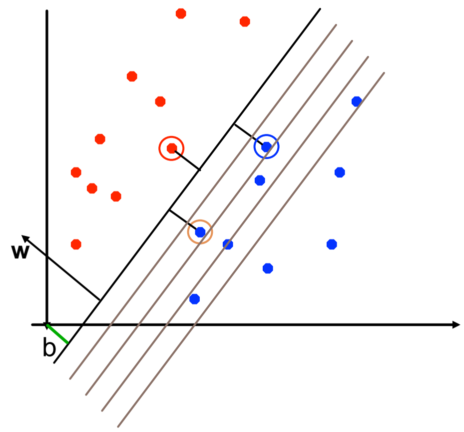
What defines a hyperplane?



What defines a hyperplane?

A hyperplane is defined by:

- A vector w
  - Perpendicular to the hyperplane
  - Often called the "weight" vector
- A scalar b
  - Selects the hyperplane that is distance b from the origin from among all possible hyperplanes



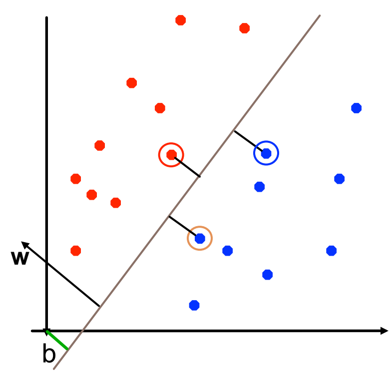
Classify a new instance (prediction)

$$D = \{(x_i, y_i) | i = 1 \ldots N\}$$

$$y_i \in \{-1, 1\}$$

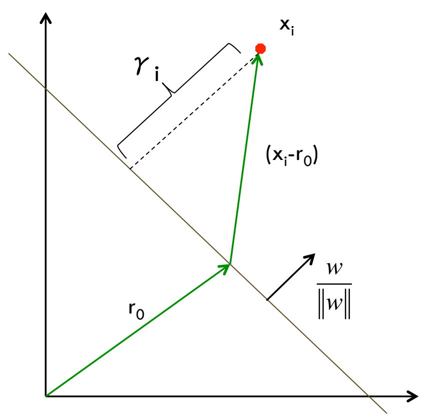
$w^\intercal x + b = 0$ x on the decision boundary
$w^\intercal x + b < 0$ x "below" the decision boundary
$w^\intercal x + b > 0$ x "above" the decision boundary

$$h(x) = \text{sign}(w \cdot x + b)$$



Learning (Training)

- How to calculate the margin?



- The margin γ_i is the length of the projection of the vector (x_i-r_0) onto the weight vector
- The length of the projection of one vector onto another is just the dot product!

$$\gamma_i = (x_i - r_0) \cdot \frac{w}{||w||}$$
$$= \frac{x_i \cdot w - r_0 \cdot w}{||w||}$$
$$= \frac{x_i \cdot w + b}{||w||}$$

Learning (Training)

$$\gamma_i = \frac{x_i \cdot w + b}{||w||} \qquad \gamma = \min_i \gamma_i$$

$$\max_{\gamma,w,b} \ \gamma \quad \text{s.t. } y_j\left(\frac{x_j \cdot w + b}{||w||}\right) \geq \gamma \ \forall j$$

Multiply by y_i to ensure positive

$$\max_{w,b} \ \frac{1}{||w||} \quad \text{s.t. } y_j\left(\frac{x_j \cdot w + b}{||w||}\right) \geq \frac{1}{||w||} \ \forall j$$

Scale w so that $\gamma = 1/||w||$

$$\max_{w,b} \ \frac{1}{||w||} \quad \text{s.t. } y_j(x_j \cdot w + b) \geq 1 \ \forall j$$

Simplify constraints

$$\min_{w,b} \ ||w|| \quad \text{s.t. } y_j(x_j \cdot w + b) \geq 1 \ \forall j$$

Max 1/x same as min x

Solving the Optimization Problem

$$\min_{w,b} \quad \frac{1}{2}||w||^2 \quad \text{s.t. } y_j\left(x_j \cdot w + b\right) \geq 1 \ \forall j$$

- Need to optimize a *quadratic* function subject to *linear* constraints
- The solution involves constructing a *dual problem* where a *Lagrange multiplier* (a scalar) is associated with every constraint in the primary problem

Solving the Optimization Problem

$$\min_{w,b} \frac{1}{2}||w||^2 \text{ such that } y^{(i)}\left(w^\intercal x^{(i)} + b\right) \geq 1 \quad \forall i$$

$$\max_{\alpha} \ \min_{w,b} \ \frac{1}{2}||w||^2 - \sum_{i=1}^{N} \alpha_i \left[y_i(w \cdot x_i + b) - 1\right]$$

Dual

Lagrange multipliers

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \left(x_i \cdot x_j\right)$$

$$\text{subject to } \alpha_i \geq 0 \text{ and } \sum_i \alpha_i y_i = 0$$

Solving the Optimization Problem

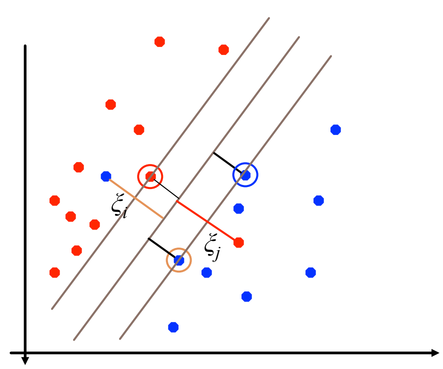
- The solution has the form:

$$w = \sum_{i=1}^{N} \alpha_i y_i \, x_i \text{ and } b = y_i - w \cdot x_i \text{ for any } x_i \text{ s.t. } \alpha_i \neq 0$$

- Each non-zero alpha indicates corresponding x_i is a *support vector*
- The classifying function has the form: $h(x) = \text{sign}\left(\sum_i \alpha_i y_i (x_i \cdot x) + b\right)$
- Relies on an dot product between the test point x and the support vectors x_i

Soft-margin Classification

- *slack variables* ξ_i can be added to allow misclassification of difficult or noisy examples.
- Still, try to minimize training set errors, and to place hyperplane "far" from each class (large margin)



How many support vectors?

- Determined by alphas in optimization
- Typically only a small proportion of the training data
- The number of support vectors determines the run time for prediction

How fast are SVMs?

Training

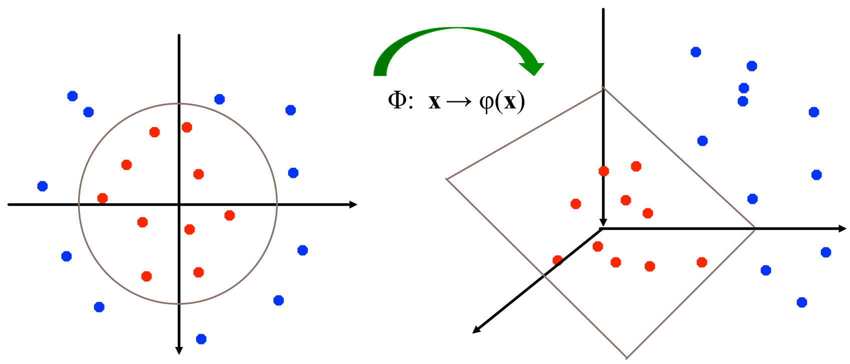
- Time for training is dominated by the time for solving the underlying quadratic programming problem
- Slower than Naïve Bayes
- Non-linear SVMs are worse

Testing (Prediction)

- Fast - as long as we don't have too many support vectors

Non-linear SVMs

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



The "Kernel" trick

- The linear classifier relies on an inner product between vectors $x_i^T x_j$

$$g(x_i) = \text{sign}\left(\sum_i \alpha_i y^{(i)} x^{(i)} x + b\right)$$

- If every example is mapped into a high-dimensional space via some transformation $\Phi: \mathbf{x} \rightarrow \varphi(\mathbf{x})$ then the inner product becomes:

$$g(x_i) = \text{sign}\left(\sum_i \alpha_i y^{(i)} \varphi(x^{(i)})^\mathsf{T} \varphi(x) + b\right)$$

- A kernel function is some function that corresponds to a dot product in some transformed feature space:

$$K(\mathbf{x}_i,\mathbf{x}_j) = \varphi(\mathbf{x}_i)^\mathsf{T}\varphi(\mathbf{x}_j)$$

The "Kernel" trick

- The kernel K may be cheaper to compute then doing the actual transformation φ
  - Implicitly do the transformation

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix}$$

$$\begin{aligned} K(x,z) &= \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_i z_i\right) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j \\ &= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) \end{aligned}$$

*

Kernels

Why use kernels?

- Make non-separable problem separable.
- Map data into better representational space

Common kernels

- Linear
- Polynomial **$K(x,z) = (1+x^Tz)^d$**
- Radial basis function (infinite dimensional space)

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

Summary

- Support Vector Machines (SVMs)
  - Find the maximum margin hyperplane
  - Only the support vectors needed to determine hyperplane
  - Use slack variables to allow some error
  - Use a kernel function to make non-separable data separable
  - Often among the best performing classifiers

*