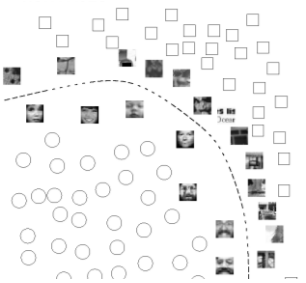
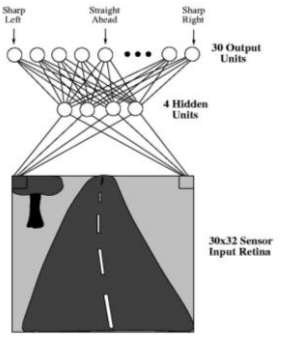
CSE 481C

A Primer on Machine Learning





Why Machine Learning for Robotics?

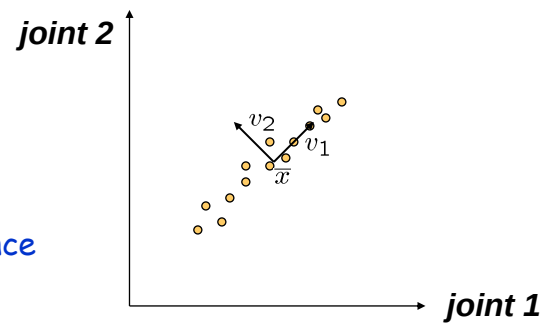
- Learning is essential for unknown environments
  - e.g., robot designer lacks omniscience
- Learning is necessary in dynamic environments
  - Robot can adapt to changes in environment not foreseen at design time
- Learning is essential for robot programming by demonstration
  - Say adios to traditional robot programming based on physics equations which require lots of assumptions and seldom work well in noisy real-world environments

Types of Learning

- Unsupervised learning: discover patterns in input data (no outputs or labels given)
  E.g., principal component analysis (PCA), clustering
- Supervised learning: correct answers/outputs for each input provided
  E.g., linear regression, backprop neural networks
- Reinforcement learning and Markov Decision Process (MDPs): occasional rewards (or punishments) given
  E.g., Q learning (for details, see: Russell & Norvig, Artificial Intelligence: A Modern Approach, 3rd ed.)

Unsupervised Learning Example 1

Principal Component Analysis (PCA)



Data covariance matrix

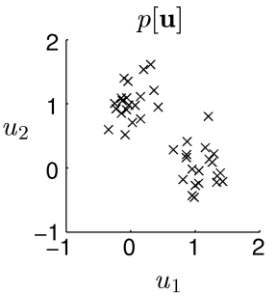
$Av = \lambda v$

v_1 is eigenvector of A with largest eigenvalue λ_1
v_2 is eigenvector of A with smaller eigenvalue λ_2

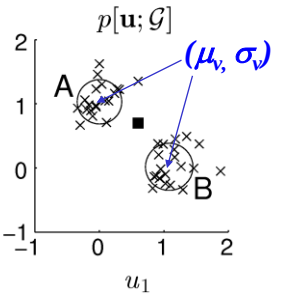
Unsupervised Learning Example 2

K-Means Clustering and Probability Density Estimation

Data



Mixture of Gaussians Model



$$p[\mathbf{u};G] = \sum_v p[\mathbf{u} \mid v;G]p[v;G]$$

Learn parameters $\boldsymbol{G} = (\mu_v, \sigma_v, \gamma_v)$

(for details, see: Russell & Norvig, Artificial Intelligence: A Modern Approach, 3rd ed.)

Supervised learning

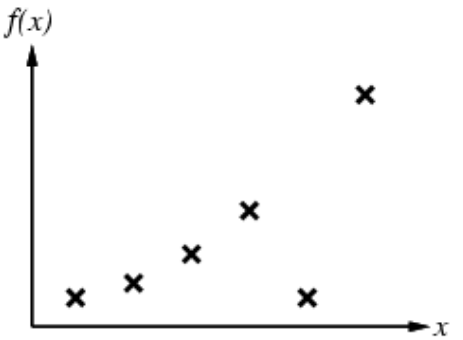
Goal: Learn a function from examples

f is the target function. Input data are pairs (*x*, *f(x)*)

Problem: learn a function ("hypothesis") *h*
such that $h \approx f$ (*h* approximates *f* as best as possible)
given a training set of (input,output) pairs

Supervised learning example

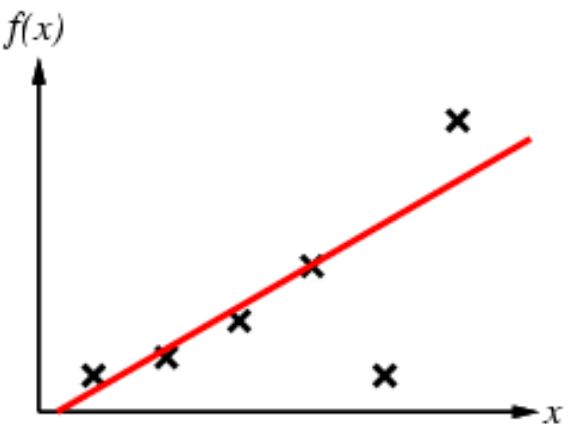
- Construct *h* to agree with *f* on training set
  h is consistent if it agrees with *f* on all training examples
- E.g., regression (curve fitting):



× = Input data point $(x,(f(x))$

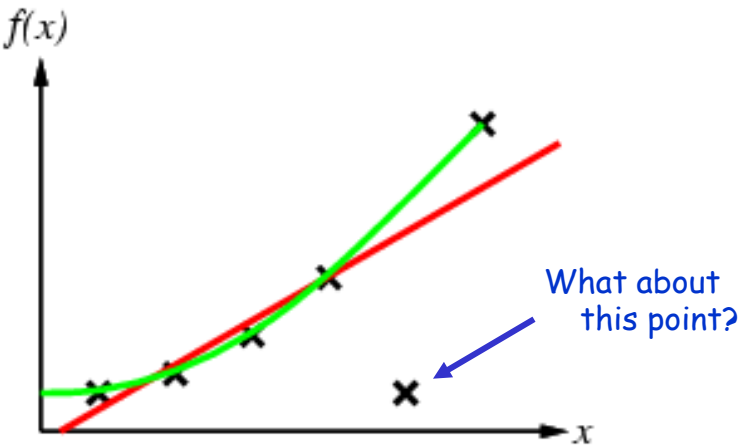
Supervised learning example

h = Straight line?



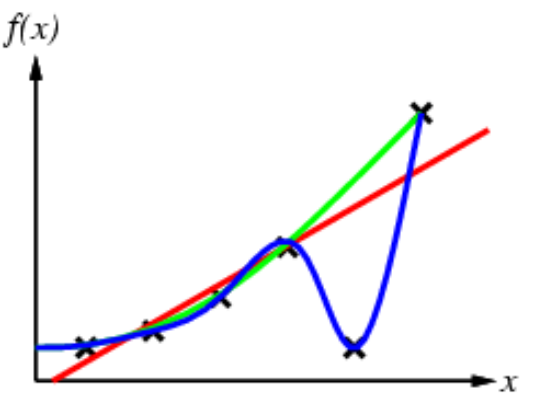
Supervised learning example

What about a quadratic function?



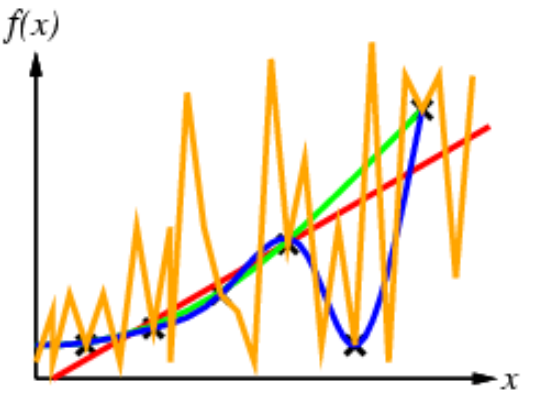
Supervised learning example

Finally, a function that satisfies all!

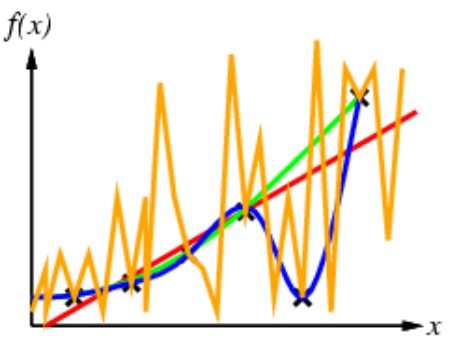


Supervised learning example

But so does this one…



Ockham's razor principle

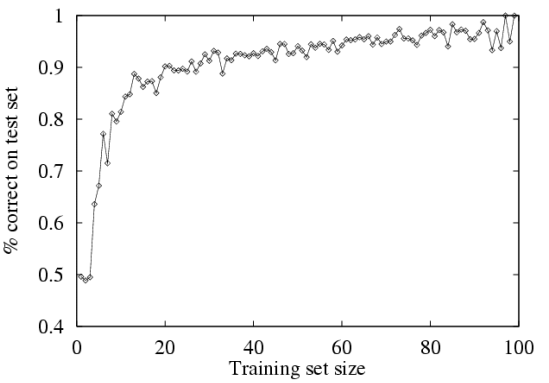


- Ockham's razor: prefer the simplest hypothesis consistent with data
  - Related to KISS principle ("keep it simple stupid")
  - Smooth blue function preferable over wiggly yellow one
  - If noise known to exist in this data, even linear might be better (the lowest x might be due to noise)

Performance Evaluation

- How do we know that the learned $h \approx f$?
- Answer: Try h on a new test set of examples

Learning curve = % correct on test set as a function of training set size



13

Generalization

- How do we know the classifier function we have learned is good?

  Look at generalization error on test data
  - Method 1: Split available data into training vs test sets (the "hold out" method)
  - Method 2: Cross-Validation

14

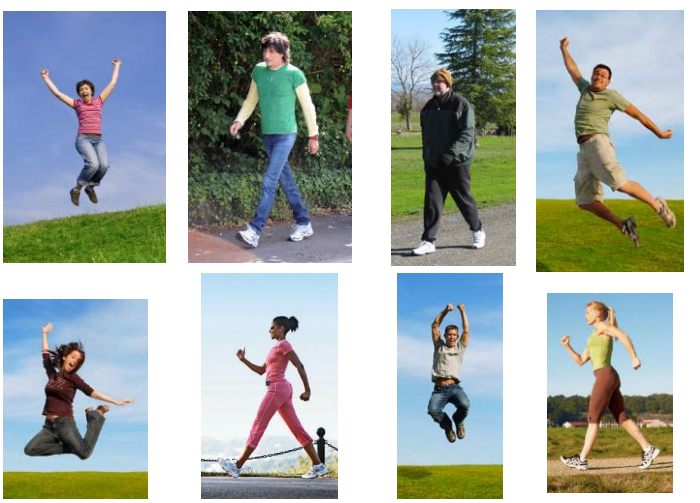
Cross-validation

- K-fold cross-validation:
  - Divide data into k subsets of equal size
  - Train learning algorithm K times, leaving out one of the subsets. Compute error on left-out subset
  - Report average error over all subsets
- Leave-1-out cross-validation:
  - Train on all but 1 data point, test on that data point; repeat for each point
  - Report average error over all points

15

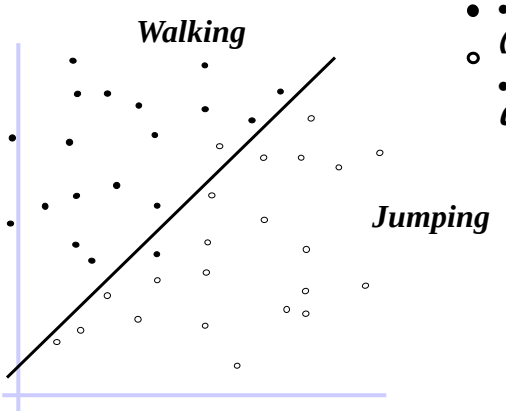
Example Problem: Action Classification

How do we build a classifier to distinguish between walking and jumping?



16

Supervised Learning as Classification



- •*denotes output of +1 (for Walking)*
- •*denotes output of -1 (or 0) (for Jumping)*

Goal: Find a separating boundary (line in this case)

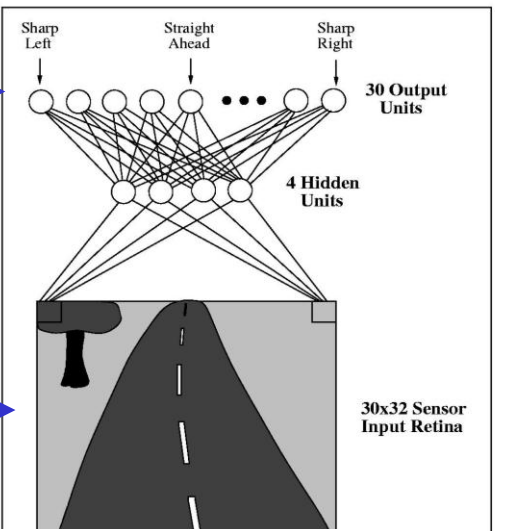
Example 2: Learning to Drive

Can a human train a robot to drive?

Supervised Learning as Regression

Get steering angle from a human driver

Training Output:
$\mathbf{d} = (d_1\ d_2 \ldots d_{30})$



Get current camera image

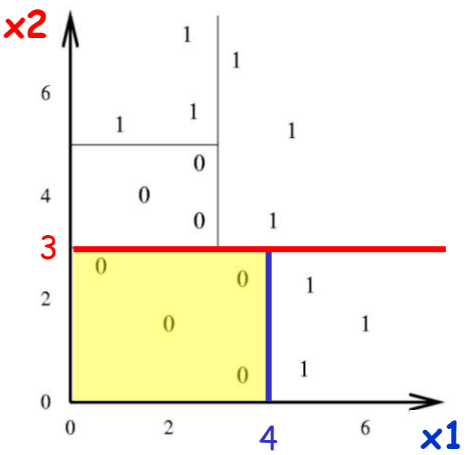
Training Input $\mathbf{u} = (u_1\ u_2 \ldots u_{960})$ = image pixels

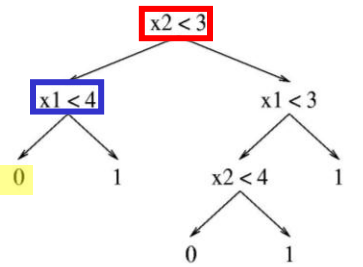
Supervised Learning

- Two Primary Tasks
  - **1. Classification**
    - Inputs $u_1, u_2, \ldots$ and discrete classes $C_1, C_2, \ldots, C_k$
    - Training examples: (u_1, C_2), (u_2, C_7), etc.
    - Learn the mapping from an arbitrary input to its class
    - Example: Inputs = video or joint angle data, output classes = walking or jumping
  - **2. Regression**
    - Inputs $u_1, u_2, \ldots$ and continuous outputs $v_1, v_2, \ldots$
    - Training examples: (input, desired output) pairs
    - Learn to map an arbitrary input to its corresponding output
    - Example: Teaching a robot how to drive
      Input = road image, output = steering angle

Classification Techniques
Decision Trees

Decision trees divide the feature space into axis-parallel rectangles, and label each rectangle with one of the K classes.



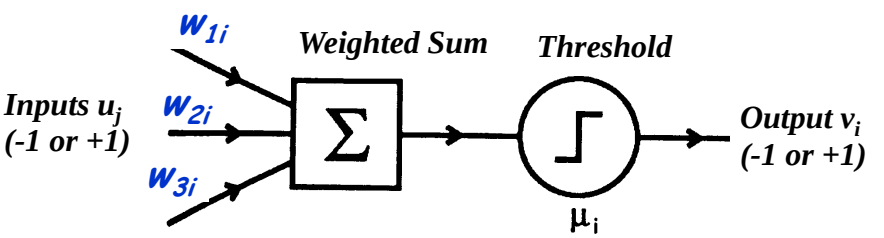


Decision Tree

(for details, see: Russell & Norvig, Artificial Intelligence: A Modern Approach, 3rd ed.)

Classification using Neurons

- Artificial neuron or "Perceptron":
  - m binary inputs (-1 or 1), 1 output (-1 or 1)
  - Connection weights w_{ji}
  - Threshold μ_i



$$v_i = sign(\sum_j w_{ji} u_j - \mu_i)$$

sign(x) = 1 if x > 0 and -1 if x ≤ 0

Perceptrons and Classification

- Consider a single-layer perceptron
  Weighted sum forms a *linear hyperplane*

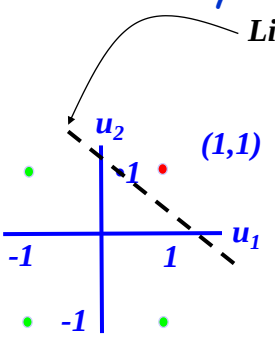
$$\sum_j w_{ji} u_j - \mu_i = 0$$

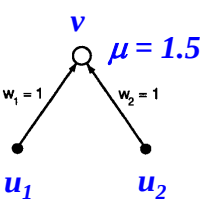
  Everything *on one side* of this hyperplane is in class 1 (output = +1) and everything *on other side* is class 2 (output = -1)
- Any function that is linearly separable can be computed by a perceptron

Linear Separability

Example: AND is linearly separable

u_1	u_2	AND
-1	-1	-1
1	-1	-1
-1	1	-1
1	1	1



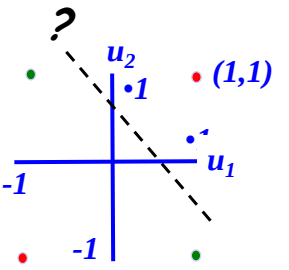


v = 1 iff $u_1 + u_2 - 1.5 > 0$

Similarly for OR and NOT

What about the XOR function?

u_1	u_2	XOR
-1	-1	1
1	-1	-1
-1	1	-1
1	1	1



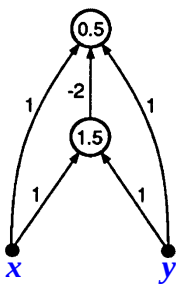
Can a perceptron separate the +1 outputs from the -1 outputs?

25

How do we deal with linear inseparability?

Multilayer Perceptrons

- Removes limitations of single-layer networks
  Can solve XOR
- Example: Two-layer perceptron that computes XOR



Backpropagation learning algorithm (to be discussed later) can be used to train multi-layered networks given (input, output) pairs

27

Back to Linear Separability

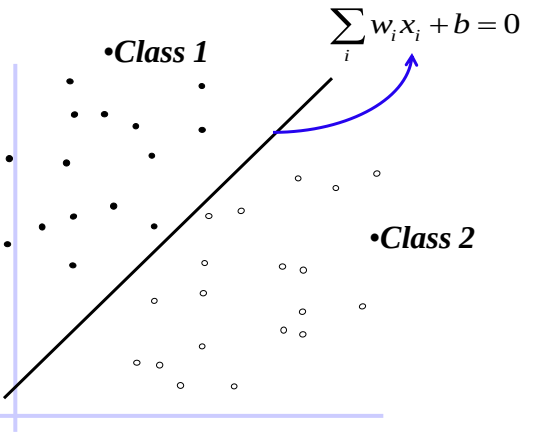
Recall: Weighted sum in perceptron forms a *linear hyperplane*

$$\sum_i w_i x_i + b = 0$$

Due to threshold function, everything *on one side* of this hyperplane is labeled as class 1 (output = +1) and everything *on other side* is labeled as class 2 (output = -1)

28

Separating Hyperplane

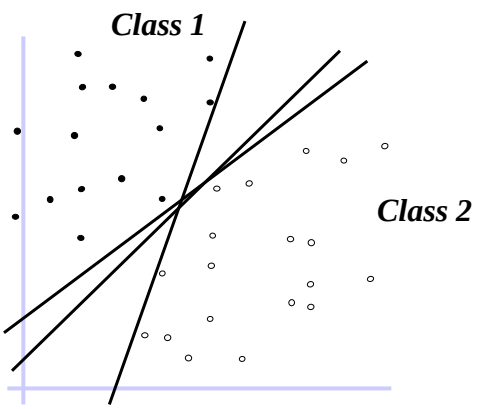


- ***denotes +1 output***
- ***denotes -1 output***

Need to choose w and b based on training data

Separating Hyperplanes

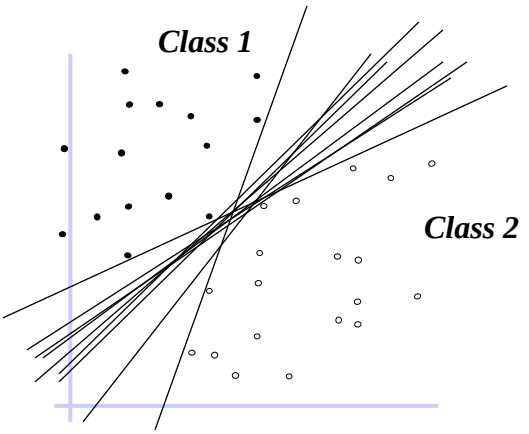
Different choices of w and b give different hyperplanes



- denotes +1 output
- denotes -1 output

(This and next few slides adapted from Andrew Moore's)

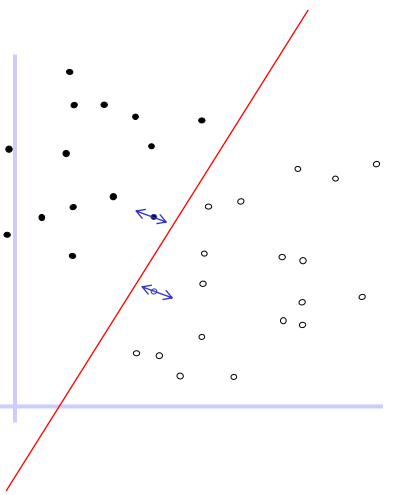
Which hyperplane is best?



- denotes +1 output
- denotes -1 output

31

How about the one right in the middle?

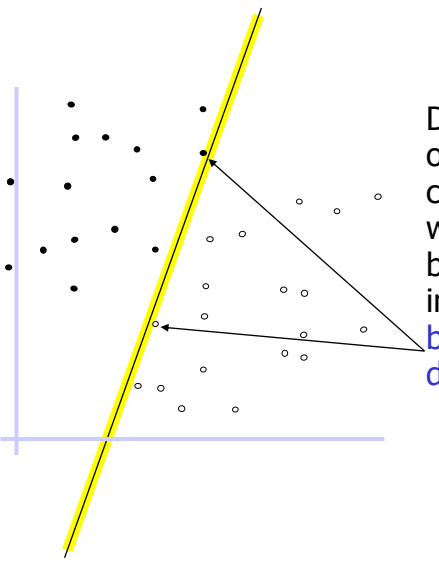


Intuitively, this boundary seems good

Avoids misclassification of new test points if they are generated from the same distribution as training points

32

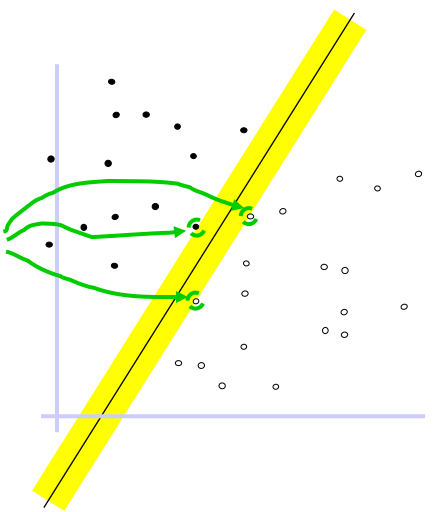
Margin



Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

33

Maximum Margin and Support Vector Machine

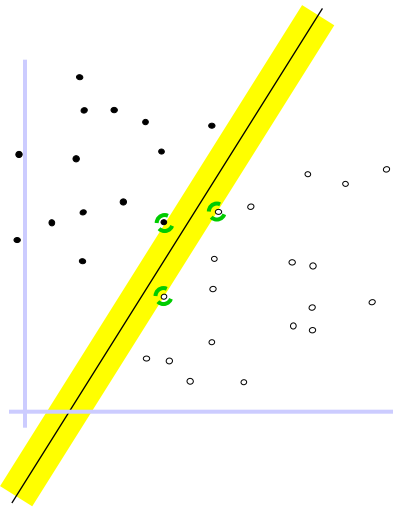


Support Vectors are those datapoints that the margin pushes up against

The maximum margin classifier is called a Support Vector Machine (in this case, a Linear SVM or LSVM)

34

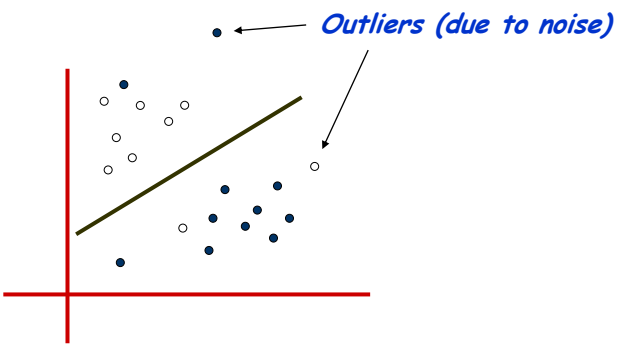
Why Maximum Margin?



- Robust to small perturbations of data points near boundary
- There exists theory showing this is best for generalization to new points
- Empirically works great

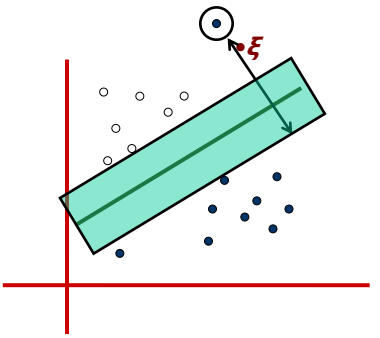
35

What if data is not linearly separable?



36

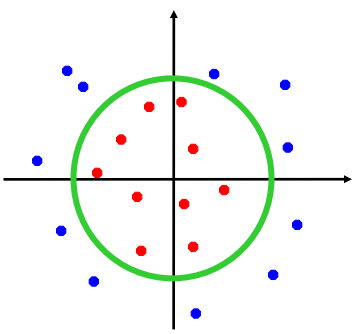
Approach 1: Soft Margin SVMs



- Allow errors ξ_i (deviations from margin)
- Trade off margin with errors.

37

What if data is not linearly separable: Other ideas?

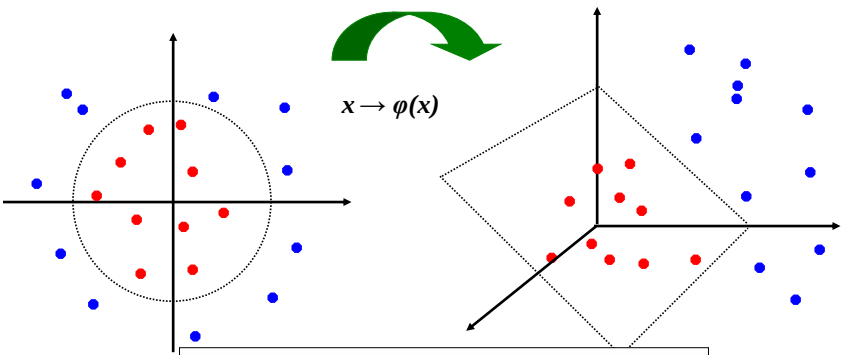


Can we do something to the inputs?

38

What if data is not linearly separable?

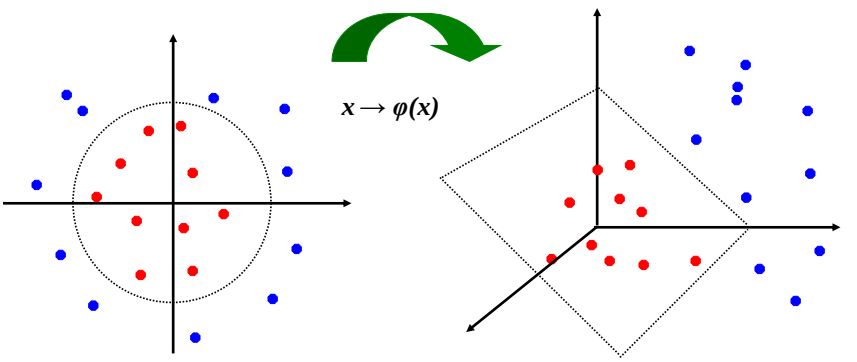
Approach 2: Map original input space to higher-dimensional feature space; use linear classifier in higher-dim. space



$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$

39

Problem: High dimensional space



- Computation in high-dimensional feature space can be costly
- The high dimensional projection function φ(x) may be too complicated to compute
- Kernel trick to the rescue!

40

The Kernel Trick

SVM maximizes a quadratic function:

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

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

Insight:

The data points only appear as inner product

No need to compute high-dimensional $\varphi(x)$ explicitly! Just replace inner product $x_i \cdot x_j$ with a kernel function $K(x_i, x_j) = \varphi(x_i) \cdot \varphi(x_j)$

E.g., Gaussian kernel

$K(x_i, x_j) = \exp(-||x_i - x_j||^2 / 2\sigma^2)$

E.g., Polynomial kernel

$K(x_i, x_j) = (x_i \cdot x_j + 1)^d$

41

An Example for $\phi(.)$ and K(.,.)

- Suppose $\phi(.)$ is given as follows

$$\phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

- An inner product in the feature space is

$$\langle \phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}), \phi(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}) \rangle = (1 + x_1y_1 + x_2y_2)^2$$

- If we define the kernel function as follows, there is no need to compute $\phi(.)$ explicitly

$$K(\mathbf{x}, \mathbf{y}) = (1 + x_1y_1 + x_2y_2)^2$$

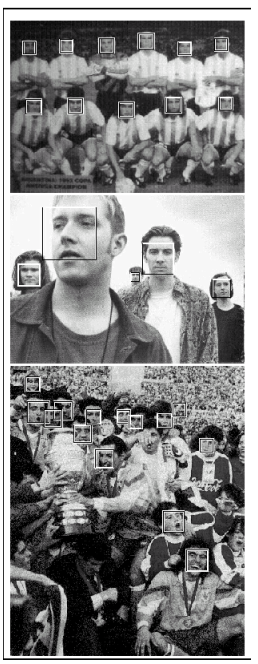
- This use of kernel function to avoid computing $\phi(.)$ explicitly is known as the kernel trick

42

Summary: Steps for Classification using SVMs

- Prepare the data matrix
- Select the kernel function to use
- Select parameters of the kernel function
  - You can use the values suggested by the SVM software, or use cross-validation
- Execute the training algorithm and obtain the parameters α_i
- Classify new data using the learned parameters

43



Face Detection using SVMs

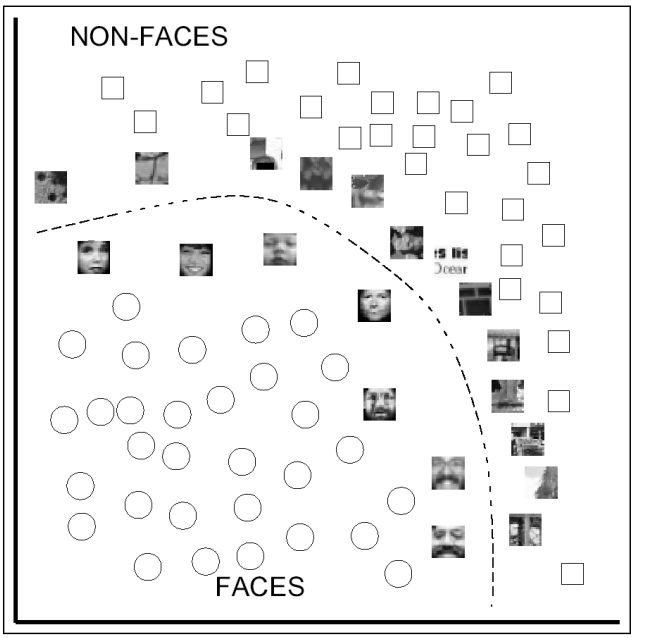
	Test Set A		Test Set B	
	Detect Rate	False Alarms	Detect Rate	False Alarms
SVM	97.1 %	4	74.2%	20
Sung *et al.*	94.6 %	2	74.2%	11

Kernel used: Polynomial of degree 2

(*Osuna, Freund, Girosi, 1998*)

44

Support Vectors



45

K-Nearest Neighbors

- A simple non-parametric non-linear classification algorithm
- Idea:

  Look around you to see how your neighbors classify data

  Classify a new data-point according to a *majority vote* of your k nearest neighbors

46

Distance Metric

- How do we measure what it means to be a neighbor (what is "close")?
- Appropriate distance metric depends on the problem
- Examples:

  discrete (e.g., strings): Hamming distance

  $d(x_1,x_2)$ = # features on which x_1 and x_2 differ

  continuous (e.g., vectors over reals): Euclidean distance

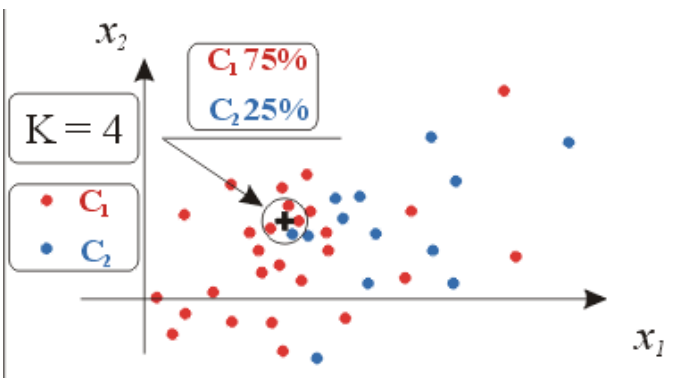
  $d(x_1,x_2)$ = $|| x_1-x_2 ||$ = square root of sum of squared differences between corresponding elements of data vectors

47

Example

Input Data: 2-D points (x_1,x_2)

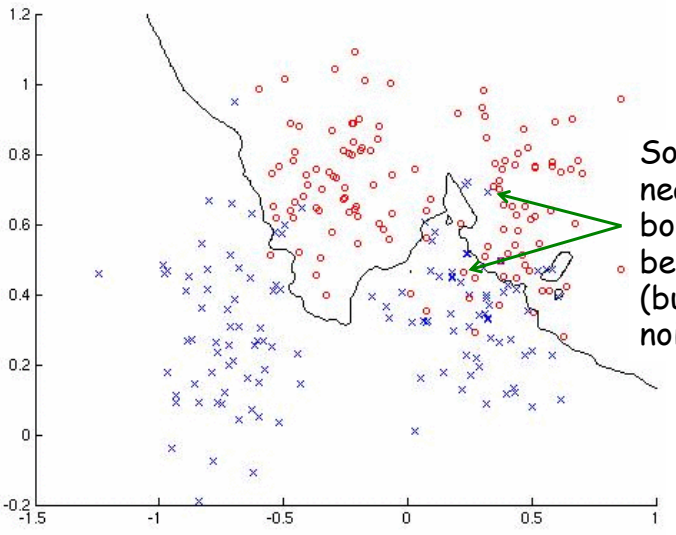
Two classes: C_1 and C_2. ***New Data Point +***



K = 4: Look at 4 nearest neighbors of + 3 are in C_1, so classify + as C_1

48

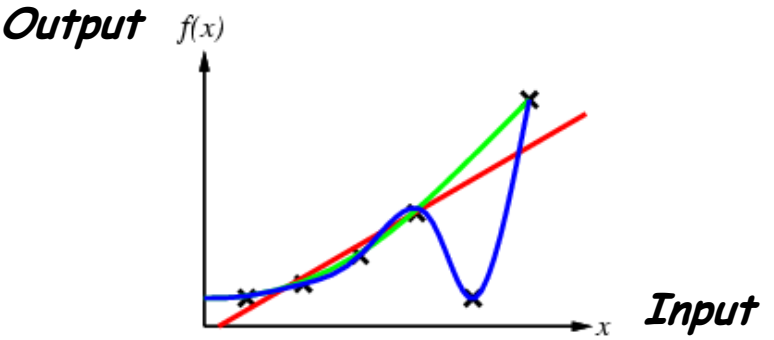
Decision Boundary using K-NN



Some points near the boundary may be misclassified (but maybe noise)

49

What if we want to learn continuous-valued functions?



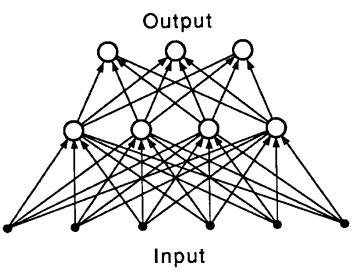
Example: Learning to Drive

Need to map sensor readings to continuous motor commands (this is the problem of regression)

51

Regression using Networks

- We want networks that can learn a function
  Network maps real-valued inputs to real-valued output
  Idea: Given data, *minimize errors* between network's output and desired output by changing weights

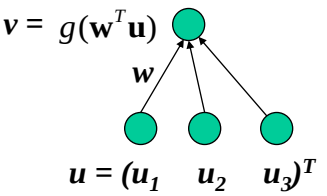


Continuous output values → Can't use binary threshold units anymore

To minimize errors, a differentiable output function is desirable

52

Sigmoidal Networks



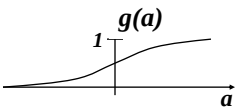
Output

Input nodes

The most common activation function:

Sigmoid function:

$$g(a) = \frac{1}{1+e^{-\beta a}}$$



Non-linear "squashing" function: Squashes input to be between 0 and 1. The parameter β controls the slope.

Gradient-Descent Learning ("Hill-Climbing")

Given training examples (u^m, d^m) (m = 1, …, N), define an error function (cost function or "energy" function)

$$E(\mathbf{w}) = \frac{1}{2}\sum_m (d^m - v^m)^2$$

where $v^m = g(\mathbf{w}^T \mathbf{u}^m)$

Gradient-Descent Learning ("Hill-Climbing")

Would like to change w so that *E*(w) is minimized

Gradient Descent: Change w in proportion to -d*E*/dw (why?)

$$\mathbf{w} \rightarrow \mathbf{w} - \varepsilon \frac{dE}{d\mathbf{w}}$$

$$\frac{dE}{d\mathbf{w}} = -\sum_m (d^m - v^m) \frac{dv^m}{d\mathbf{w}} = -\sum_m (d^m - v^m) g'(\mathbf{w}^T \mathbf{u}^m) \mathbf{u}^m$$

Derivative of sigmoid

55

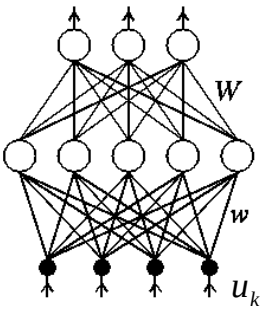
But wait....

- Delta rule tells us how to modify the connections from input to output (one layer network)
  - One layer networks are not that interesting (remember XOR?)
- What if we have multiple layers?

56

Learning Multilayer Networks

$$v_i = g(\sum_j W_{ji}\, g(\sum_k w_{kj} u_k))$$



- Start with random weights W, w
- Given input u, network produces output v
- Find W and w that minimize total squared output error over all output units (labeled i):

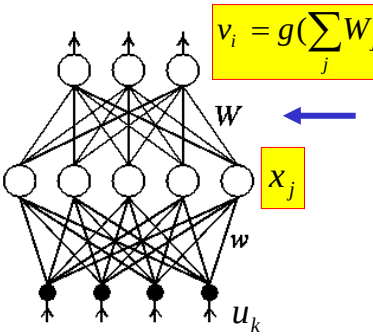
$$E(\mathbf{W},\mathbf{w}) = \frac{1}{2}\sum_i (d_i - v_i)^2$$

57

Backpropagation Learning : Output Weights

$$E(\mathbf{W},\mathbf{w}) = \frac{1}{2}\sum_i (d_i - v_i)^2$$

$$v_i = g(\sum_j W_{ji} x_j)$$



Learning rule for hidden-output weights W:

$$W_{ji} \rightarrow W_{ji} - \varepsilon \frac{dE}{dW_{ji}} \qquad \{\textit{gradient descent}\}$$

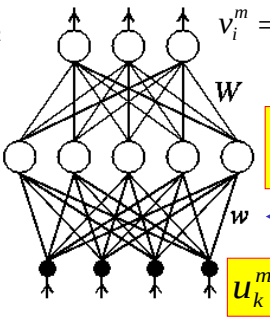
$$\frac{dE}{dW_{ji}} = -(d_i - v_i)\, g'(\sum_j W_{ji} x_j)\, x_j \quad \{\textit{delta rule}\}$$

58

Backpropagation: Hidden Weights

$$E(\mathbf{W},\mathbf{w}) = \frac{1}{2}\sum_i (d_i - v_i)^2$$

$$v_i^m = g(\sum_j W_{ji} x_j)$$



$$x_j^m = g(\sum_k w_{kj} u_k^m)$$

u_k^m

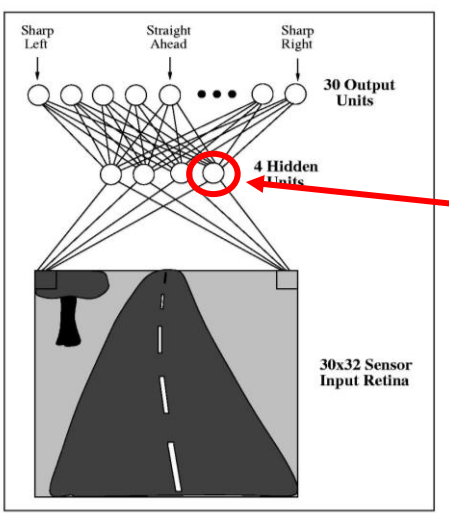
Learning rule for input-hidden weights w:

$$w_{kj} \rightarrow w_{kj} - \varepsilon \frac{dE}{dw_{kj}} \qquad \text{But}: \frac{dE}{dw_{kj}} = \frac{dE}{dx_j} \cdot \frac{dx_j}{dw_{kj}} \quad \{\textit{chain rule}\}$$

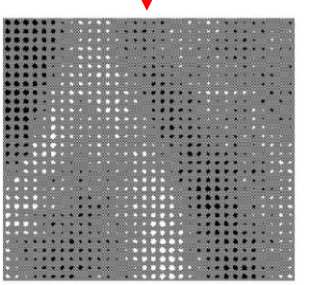
$$\frac{dE}{dw_{kj}} = \left[-\sum_{m,i} (d_i^m - v_i^m) g'(\sum_j W_{ji} x_j^m) W_{ji} \right] \cdot \left[g'(\sum_k w_{kj} u_k^m) u_k^m \right]$$

59

Teaching a robotic vehicle to drive

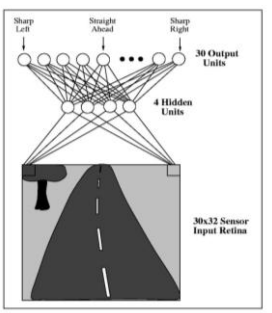


One of the learned "road features" w_i



60

ALVINN (Autonomous Land Vehicle in a Neural Network)



CMU Navlab

- Trained using human driver + camera images
- After learning:
  - Drove up to 70 mph on highway
  - Up to 22 miles without intervention
  - Drove cross-country largely autonomously

(Pomerleau, 1992)

Demos: Learning Pole Balancing and Backing up a Truck from a Human Teacher

(courtesy of Keith Grochow, CSE 599)

- Neural network learns to balance a pole on a cart
  - System:
    - 4 state variables: x_{cart}, v_{cart}, θ_{pole}, v_{pole}
    - 1 input: Force on cart
  - Backprop Network:
    - Input: State variables
    - Output: New force on cart
- NN learns to back a truck into a loading dock
  - System (Nyugen and Widrow, 1989):
    - State variables: x_{cab}, y_{cab}, θ_{cab}
    - 1 input: new $\theta_{steering}$
  - Backprop Network:
    - Input: State variables
    - Output: Steering angle $\theta_{steering}$