Neural Networks

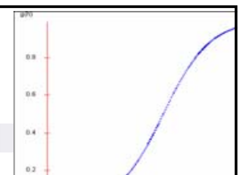
Machine Learning – 10701/15781

Carlos Guestrin, XOXO

Carnegie Mellon University

February 14th, 2007

Logistic regression



- P(Y|X) represented by:

$$P(Y=1 \mid x, W) = \frac{1}{1+e^{-(w_0+\sum_i w_i x_i)}}$$
$$= g(w_0 + \sum_i w_i x_i)$$

- Learning rule – MLE:

Compute derivative

$$\frac{\partial \ell(W)}{\partial w_i} = \sum_j x_i^j [y^j - P(Y^j = 1 \mid x^j, W)]$$
$$= \sum_j x_i^j [y^j - g(w_0 + \sum_i w_i x_i^j)]$$

learning rate

$$\boxed{w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \sum_j x_i^j \delta^j}$$

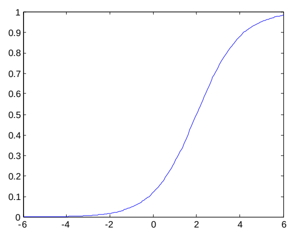
truth   prediction

$$\delta^j = y^j - g(w_0 + \sum_i w_i x_i^j)$$

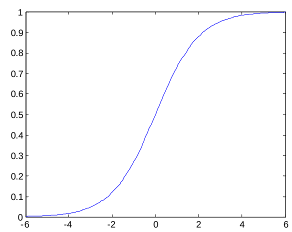
Sigmoid

$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}$$

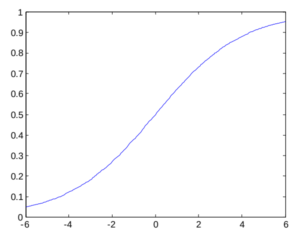
w_0=2, w_1=1



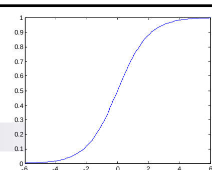
w_0=0, w_1=1



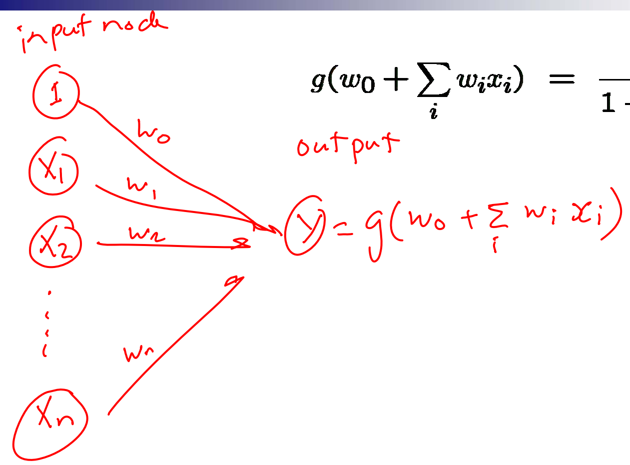
w_0=0, w_1=0.5



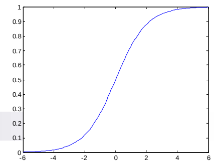
Perceptron as a graph



$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}$$



Linear perceptron classification region



$$g(w_0 + \sum_i w_i x_i) = \frac{1}{1 + e^{-(w_0 + \sum_i w_i x_i)}}$$

+

$w_0 + \sum w_i x_i > 0$

$w_0 + \sum w_i x_i = 0$

−

$w_0 + \sum_i w_i x_i < 0$

Optimizing the perceptron

$\frac{\partial}{\partial w_i} f(g(w)) = \frac{\partial f(g(w))}{\partial g} \cdot \frac{\partial g}{\partial w_i}$

- Trained to minimize sum-squared error

$$\ell(W) = \frac{1}{2}\sum_{j=1}^{m}[\underbrace{y^j}_{\text{truth}} - \underbrace{g(w_0 + \sum_i w_i x_i^j)}_{\text{prediction}}]^2$$

squared

datapoints

$$\frac{\partial \ell(W)}{\partial w_k} = \frac{\partial}{\partial w_k}\left[\frac{1}{2}\sum_j [y_j - g(w_0 + \sum_i w_i x_i^j)]^2\right]$$

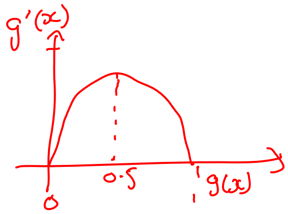
$$= \frac{1}{2}\sum_j \frac{\partial}{\partial w_k}[y_j - g(w_0 + \sum_i w_i x_i^j)]^2$$

Derivative of sigmoid

$$\frac{\partial \ell(W)}{\partial w_{\mu}} = -\sum_j \overbrace{[y^j - g(w_0 + \sum_i w_i x_i^j)]}^{\text{diff truth prediction}} \overbrace{x_{\mu}^j}^{\text{magnitude of feature}} \overbrace{g'(w_0 + \sum_i w_i x_i^j)}^{\text{derivative of logistic function}}$$

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

$$g'(x) = \frac{\partial g(x)}{dx} = g(x)\,(1-g(x))$$



derivative large near decision boundary

$$\frac{\partial \ell}{\partial w_{\mu}} = -\sum_j \left[y^j - g(w_0 + \sum_i w_i x_i^j)\right] x_{\mu}^j \quad g'(w_0 + \Sigma_i x_i^j)$$

The perceptron learning rule

learning rate strength feature

$$\boxed{w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \sum_j x_i^j \delta^j}$$

pushes me towards right answer

push away from boundary

$$\delta^j = [y^j - g(w_0 + \sum_i w_i x_i^j)]\, g^j (1 - g^j)$$

$$g^j = g(w_0 + \sum_i w_i x_i^j)$$

both unhappy when classification is wrong

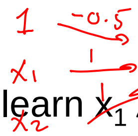
- Compare to MLE:

$$\boxed{w_i \leftarrow w_i + \eta \sum_j x_i^j \delta^j} \qquad \delta^j = [y^j - g(w_0 + \sum_i w_i x_i^j)]$$

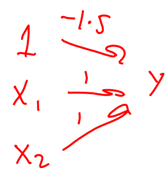
Percepton, linear classification, Boolean functions

$$H(x) = \begin{cases} w_0 + \sum_i x_i w_i \geq 0 \Rightarrow \text{true} \\ w_0 + \sum_i w_i x_i < 0 \Rightarrow \text{false} \end{cases}$$

- Can learn $x_1 \vee x_2 = y$

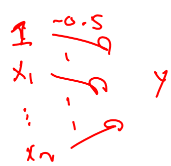


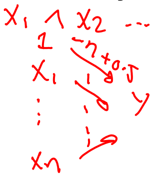
- Can learn $x_1 \wedge x_2 = y$



- Can learn any conjunction or disjunction

$x_1 \vee x_2 \vee \dots \vee x_n$ $\qquad$ $x_1 \wedge x_2 \cdots \wedge x_n$



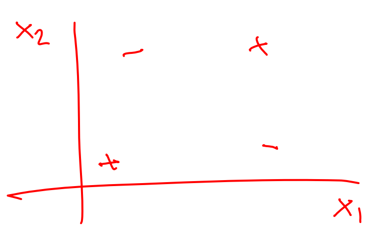


Percepton, linear classification, Boolean functions

- Can learn majority $\qquad \sum_i x_i > n/2$

- Can perceptrons do everything?

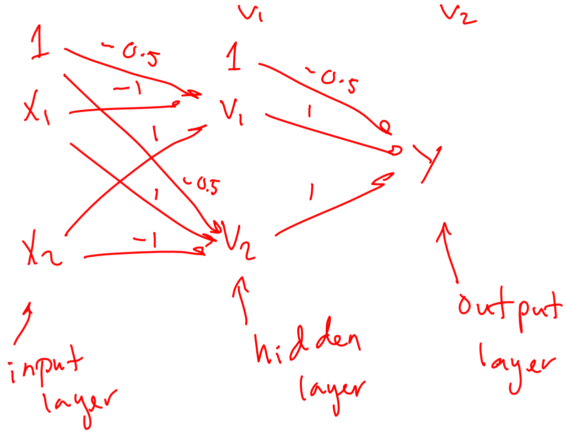
XOR $\equiv y = x_1 \wedge \neg x_2 \vee \neg x_1 \wedge x_2$



Going beyond linear classification

- Solving the XOR problem

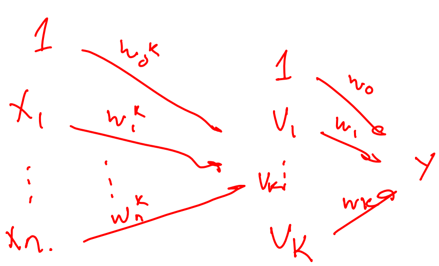
$$Y = \underbrace{\neg X_1 \wedge X_2}_{V_1} \vee \underbrace{X_1 \wedge \neg X_2}_{V_2}$$



Hidden layer

- Perceptron: $\underline{out(\mathbf{x})} = g(w_0 + \sum_i w_i x_i)$
- 1-hidden layer:

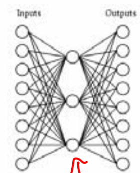
$$out(\mathbf{x}) = g\left(w_0 + \sum_k w_k \underbrace{g(w_0^k + \sum_i w_i^k x_i)}_{V_K}\right)$$



Example data for NN with hidden layer

8 bits

8 binary outputs



3 hidden binary features

A target function:

Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

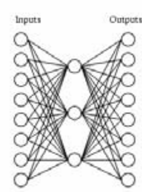
Can this be learned??

Learned weights for hidden layer

A network:

activation of V_1, V_2, V_3

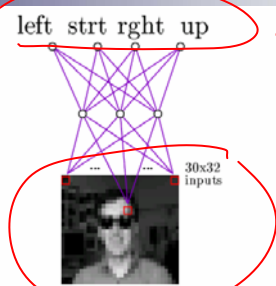
correspond to binary encoding



Learned hidden layer representation:

Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

NN for images



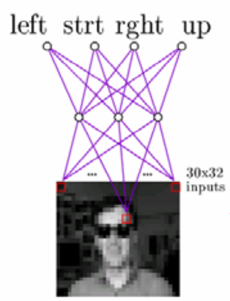
output

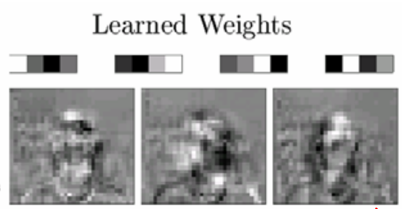
inputs

Typical input images

90% accurate learning head pose, and recognizing 1-of-20 faces

Weights in NN for images





weights for hidden nodes

white → positive
gray → zero
black → negative

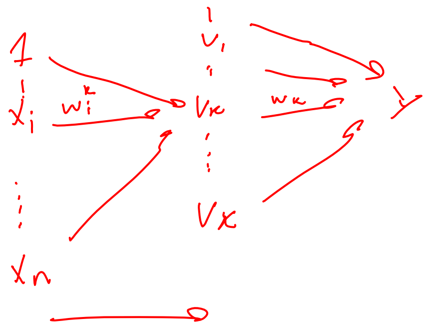
Typical input images

Forward propagation for 1-hidden layer - Prediction

- 1-hidden layer:

w's are given
want to classify input x

$$\underset{y}{out(\mathbf{x})} = g\left(w_0 + \sum_k w_k \underbrace{g(w_0^k + \sum_i w_i^k x_i)}_{v_k}\right)$$



Gradient descent for 1-hidden layer – Back-propagation: Computing $\frac{\partial \ell(W)}{\partial w_k}$

Dropped w_0 to make derivation simpler

$$\ell(W) = \frac{1}{2}\sum_j [y^j - out(\mathbf{x}^j)]^2$$

$$out(\mathbf{x}) = g\left(\overbrace{\sum_{k'} w_{k'} g(\sum_{i'} w_{i'}^{k'} x_{i'})}^{f}\right)$$

$$\frac{\partial out}{\partial w_k} = \frac{\partial f}{\partial w_k} \cdot \frac{\partial g}{\partial f}$$

$$g' = g \cdot (1-g)$$

$$\frac{\partial \ell(W)}{\partial w_k} = \sum_{j=1}^m -[y_j - out(\mathbf{x}^j)]\frac{\partial out(\mathbf{x}^j)}{\partial w_k}$$

$$\frac{\partial out(x^j)}{\partial w_k} = \underbrace{g\left(\sum_{i'} w_{i'}^k x_{i'}\right)}_{v_k} \underbrace{g'\left(\sum_{k'} w_{k'} g\left(\sum_{i'} w_{i'}^{k'} x_{i'}\right)\right)}_{out(x)\,(1-out(x))}$$

$$= v_k \, out(x)\,(1-out(x))$$

Gradient descent for 1-hidden layer – Back-propagation: Computing $\frac{\partial \ell(W)}{\partial w_i^k}$

Dropped w_0 to make derivation simpler

$$\ell(W) = \frac{1}{2}\sum_j [y^j - out(\mathbf{x}^j)]^2$$

$$out(\mathbf{x}) = g\left(\sum_{k'} w_{k'} g(\sum_{i'} w_{i'}^{k'} x_{i'})\right)$$

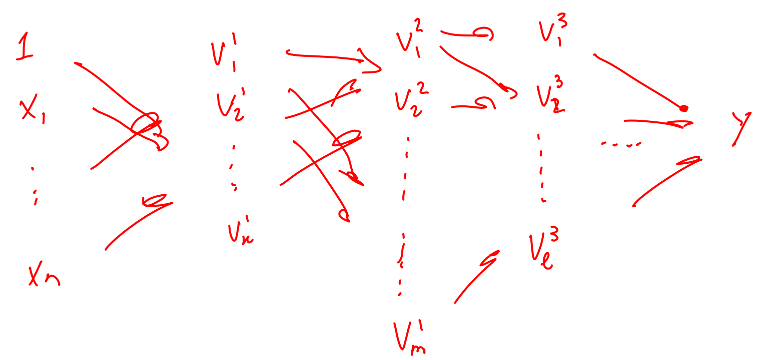
$$\frac{\partial}{\partial w_i^k} g(f(h)) = g'(f(h)) \cdot \frac{\partial f(h)}{\partial h} \cdot \frac{\partial h}{\partial w_i^k}$$

$$\frac{\partial \ell(W)}{\partial w_i^k} = \sum_{j=1}^{m} -[y - out(\mathbf{x}^j)] \frac{\partial out(\mathbf{x}^j)}{\partial w_i^k}$$

$$\frac{\partial out(x)}{\partial w_i^k} = \underbrace{x_i}_{x_i} \cdot \underbrace{g'(\sum_{i'} w_{i'}^k x_{i'})}_{v_k(1-v_k)} \; \underbrace{g'(\sum_{k'} w_{k'} g(\sum_{i'} w_{i'}^{k'} x_{i'}))}_{out(x)(1-out(x))}$$

$$= \underline{x_i \, v_k(1-v_k) \; out(x)(1-out(x))}$$

Multilayer neural networks



Forward propagation – prediction

- Recursive algorithm
- Start from input layer
- Output of node V_k with parents $U_1, U_2, \ldots$:

$$V_k = g\left(\sum_i w_i^k U_i\right)$$

Back-propagation – learning

- Just gradient descent!!!
- Recursive algorithm for computing gradient
- For each example
  - Perform forward propagation
  - Start from output layer
  - Compute gradient of node V_k with parents $U_1, U_2, \ldots$
  - Update weight w_i^k

Many possible response functions

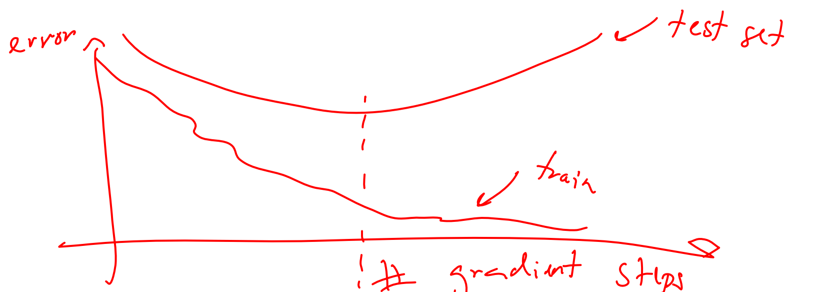
- Sigmoid $g(w_0 + \sum_i w_i x_i)$
- Linear $w_0 + \sum_i w_i x_i$
- Exponential $e^{w_0 + \sum_i w_i x_i}$
- Gaussian $e^{-\frac{(w_0 + \sum_i w_i x_i)^2}{\sigma^2}}$
- … Threshold
  ⋮

Convergence of backprop

- Perceptron leads to convex optimization
  - Gradient descent reaches **global minima**
- Multilayer neural nets **not convex**
  - Gradient descent gets stuck in local minima
  - Hard to set learning rate
  - Selecting number of hidden units and layers = fuzzy process
  - NNs falling in disfavor in last few years
  - We'll see later in semester, *kernel trick* is a good alternative
  - Nonetheless, neural nets are one of the most used ML approaches

~~Training set error~~ Overfitting

- Neural nets represent complex functions
  - Output becomes more complex with gradient steps



Stop early to avoid overfitting

Overfitting

- Output fits training data "too well"
  - Poor test set accuracy
- Overfitting the training data
  - Related to bias-variance tradeoff
  - One of central problems of ML
- Avoiding overfitting?
  - More training data
  - Regularization, regularize w.r.t. w^2 (typically)
  - Early stopping momentum, ...

What you need to know about neural networks

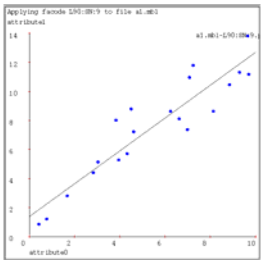
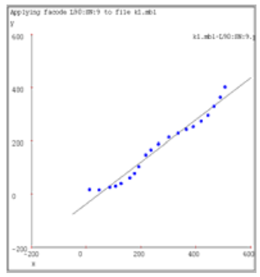
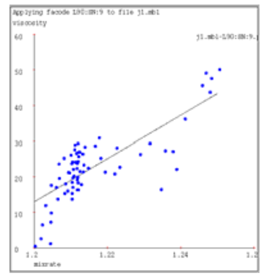
- Perceptron:
  - Representation
  - Perceptron learning rule
  - Derivation
- Multilayer neural nets
  - Representation
  - Derivation of backprop
  - Learning rule
- Overfitting
  - Definition
  - Training set versus test set
  - Learning curve

Instance-based Learning

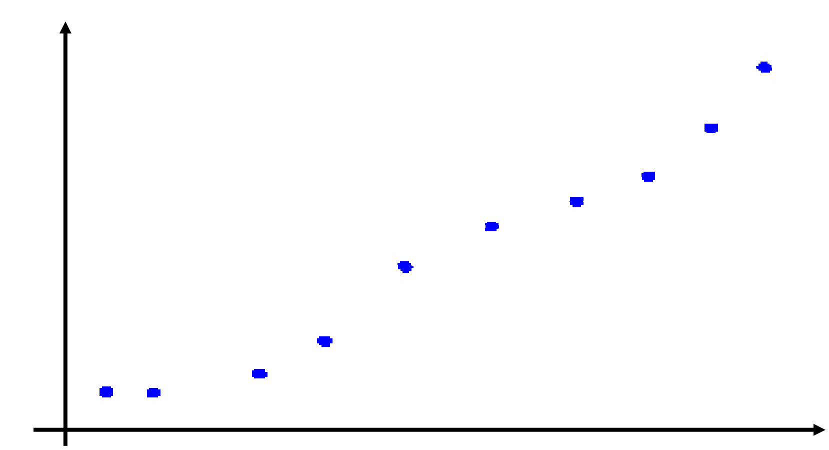
Machine Learning – 10701/15781
Carlos Guestrin
Carnegie Mellon University

February 14^{th}, 2007

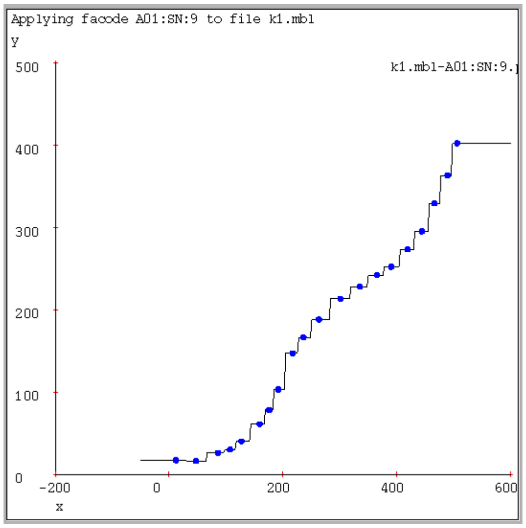
Why not just use Linear Regression?

Using data to predict new data



Nearest neighbor



Univariate 1-Nearest Neighbor

Given datapoints (x_1,y_1) (x_2,y_2)..(x_N,y_N),where we assume $y_i=f(x_i)$ for some unknown function f.

Given query point x_q, your job is to predict $\hat{y} \approx f(x_q)$

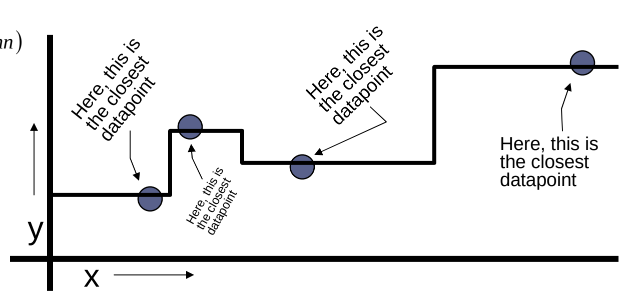
Nearest Neighbor:

1. Find the closest x_i in our set of datapoints

$$i(nn) = \underset{i}{\operatorname{argmin}} \left| x_i - x_q \right|$$

2. Predict $\hat{y} = y_{i(nn)}$

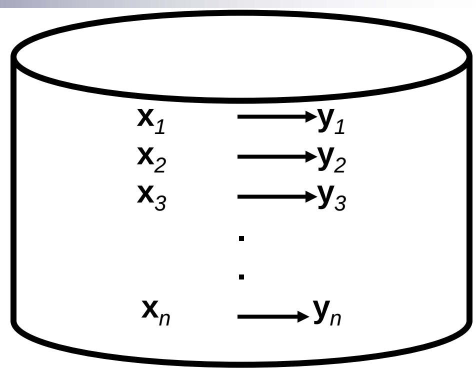
Here's a dataset with one input, one output and four datapoints.



1-Nearest Neighbor is an example of…. **Instance-based learning**

A function approximator that has been around since about 1910.

To make a prediction, search database for similar datapoints, and fit with the local points.



Four things make a memory based learner:

- A distance metric
- How many nearby neighbors to look at?
- A weighting function (optional)
- How to fit with the local points?

1-Nearest Neighbor

Four things make a memory based learner:

1. *A distance metric*
   Euclidian (and many more)
2. *How many nearby neighbors to look at?*
   One
3. *A weighting function (optional)*
   Unused
4. *How to fit with the local points?*
   Just predict the same output as the nearest neighbor.

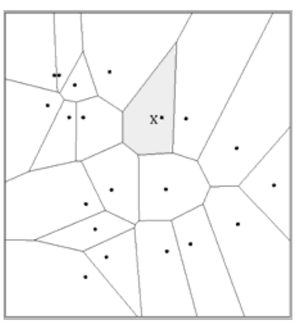
Multivariate 1-NN examples

Regression

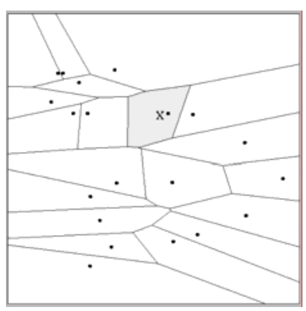
Classification

Multivariate distance metrics

Suppose the input vectors x1, x2, …xn are two dimensional:
$\mathbf{x}_1 = (x_{11}, x_{12})$, $\mathbf{x}_2 = (x_{21}, x_{22})$, …$\mathbf{x}_N = (x_{N1}, x_{N2})$.
One can draw the nearest-neighbor regions in input space.



$Dist(\mathbf{x}_i,\mathbf{x}_j) = (x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2$



$Dist(\mathbf{x}_i,\mathbf{x}_j) = (x_{i1} - x_{j1})^2 + (3x_{i2} - 3x_{j2})^2$

The relative scalings in the distance metric affect region shapes.