Nearest Neighbor and Linear Discriminant Functions (perceptrons)

Biometrics

CSE 190-a

Lecture 8

Announcements

- HW1 assigned, due thursday
- Most of last lecture was on the blackboard.

Non-Parametric Density Estimation

- Given a collection of *n* samples, estimate the probability density.
  - Parzen Windows
  - K-th nearest neighbor
- Main ideas:
  1. As number of samples *(n)* approaches infinity, estimated density should approach true density
  2. Approximated density should be "reasonable" for finite *n.*

Three necessary conditions should apply if we want $p_n(x)$ to converge to $p(x$

$$1)\ \lim_{n\to\infty} V_n = 0$$
$$2)\ \lim_{n\to\infty} k_n = \infty$$
$$3)\ \lim_{n\to\infty} k_n / n = 0$$

There are two different ways of obtaining sequences of regions that satisfy these conditions:

(a) Shrink an initial region where $V_n = 1/\sqrt{n}$ and show that

$$p_n(x) \underset{n\to\infty}{\rightarrow} p(x)$$

This is called "the Parzen-window estimation method"

(b) Specify k_n as some function of n, such as $k_n = \sqrt{n}$; the volume V_n is grown until it encloses k_n neighbors of x. This is called "the k_n-nearest neighbor estimation method"

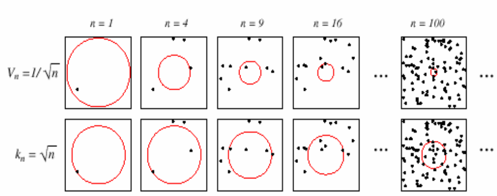


FIGURE 4.2. There are two leading methods for estimating the density at a point, here at the center of each square. The one shown in the top row is to start with a large volume centered on the test point and shrink it according to a function such as $V_n = 1/\sqrt{n}$. The other method, shown in the bottom row, is to decrease the volume in a data-dependent way, for instance letting the volume enclose some number $k_n = \sqrt{n}$ of sample points. The sequences in both cases represent random variables that generally converge and allow the true density at the test point to be calculated. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Parzen Windows

- Parzen-window approach to estimate densities assume that the region R_n is a d-dimensional hypercube

$$V_n = h_n^d \quad (h_n : \text{length of the edge of } \Re_n)$$

Let $\varphi(u)$ be the following window function :

$$\varphi(u) = \begin{cases} 1 & |u_j| \leq \frac{1}{2} \quad j = 1,\dots,d \\ 0 & \text{otherwise} \end{cases}$$

- $\varphi((x-x_i)/h_n)$ is equal to unity if x_i falls within the hypercube of volume V_n centered at x and equal to zero otherwise.

6

- The number of samples in this hypercube is:

$$k_n = \sum_{i=1}^{i=n} \varphi\left(\frac{x - x_i}{h_n}\right)$$

By substituting k_n in equation (7), we obtain the following estimate:

$$p_n(x) = \frac{1}{n}\sum_{i=1}^{i=n} \frac{1}{V_n}\varphi\left(\frac{x - x_i}{h_n}\right)$$

$P_n(x)$ estimates $p(x)$ as an average of functions of x and the samples (x_i) $(i = 1,\ldots,n)$. These functions φ can be general!

7

- Parzen Window Example
  - Draw samples from a Normal distribution, *N(0,1)*
  - Let $\varphi(u) = (1/\sqrt{(2\pi)}\, \exp(-u^2/2)$
    $h_n = h_1/\sqrt{n}$ $(n>1)$

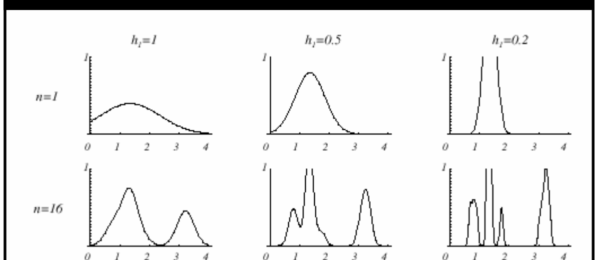
Thus:

$$p_n(x) = \frac{1}{n}\sum_{i=1}^{i=n} \frac{1}{h_n}\varphi\left(\frac{x - x_i}{h_n}\right)$$

is an average of normal densities centered at the samples x_i.

8

- Case where $p(x) = \lambda_1.U(a,b) + \lambda_2.T(c,d)$ (unknown density) (mixture of a uniform and a triangle density)



9

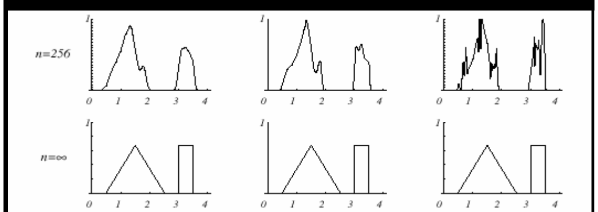


FIGURE 4.7. Parzen-window estimates of a bimodal distribution using different window widths and numbers of samples. Note particularly that the $n = \infty$ estimates are the same (and match the true distribution), regardless of window width. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

10

Probabalistic Neural network contains Parzen Window

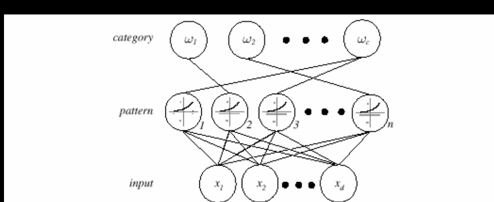


FIGURE 4.9. A probabilistic neural network (PNN) consists of d input units, n pattern units, and c category units. Each pattern unit forms the inner product of its weight vector and the normalized pattern vector $\mathbf{x}$ to form $z = \mathbf{w}^t\mathbf{x}$, and then it emits $\exp[(z-1)/\sigma^2]$. Each category unit sums such contributions from the pattern unit connected to it. This ensures that the activity in each of the category units represents the Parzen-window density estimate using a circularly symmetric Gaussian window of covariance $\sigma^2\mathbf{I}$, where $\mathbf{I}$ is the $d \times d$ identity matrix. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

11

- K_n - Nearest neighbor estimation
  - Goal: a solution for the problem of the unknown "best" window function
    - Let the cell volume be a function of the training data
    - Center a cell about x and let it grows until it captures k_n samples $(k_n = f(n))$
    - k_n are called the k_n nearest-neighbors of x

  2 possibilities can occur:

    - Density is high near x; therefore the cell will be small which provides a good resolution
    - Density is low; therefore the cell will grow large and stop until higher density regions are reached

  We can obtain a family of estimates by setting $k_n = \sqrt{n}$

12

Illustration

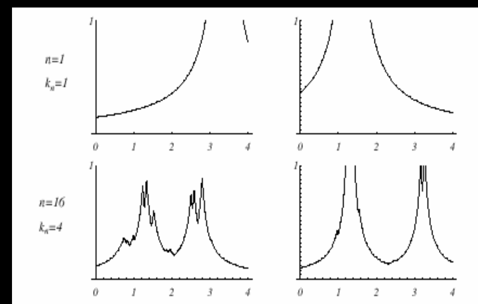
For $n = 1$ and $k_n = \sqrt{n} = 1$; the estimate becomes:

$P_n(x) = k_n / n.V_n = 1 / V_1 = 1 / 2|x-|$

Yikes! Well not so good as the probability goes to infinity at x_1 but at least we do not have holes in the density!

Things get better as n gets bigger! And we still don't have holes in the density even for higher dimensions!

13



14

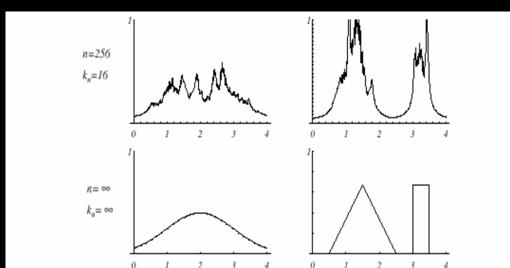


FIGURE 4.12. Several k-nearest-neighbor estimates of two unidimensional densities: a Gaussian and a bimodal distribution. Notice how the finite n estimates can be quite "spiky." From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

15

- **The nearest –neighbor rule**
  - Let $D_n = \{x_1, x_2, \ldots, x_n\}$ be a set of n labeled prototypes
  - Let $x' \in D_n$ be the closest prototype to a test point x then the nearest-neighbor rule for classifying x is to assign it the label associated with x'
  - The nearest-neighbor rule leads to an error rate greater than the minimum possible: the Bayes rate
  - If the number of prototype is large (unlimited), the error rate of the nearest-neighbor classifier is never worse than twice the Bayes rate (it can be demonstrated!)
  - If $n \to \infty$, it is always possible to find x' sufficiently close so that:

$$P(\omega_i \mid x') \cong P(\omega_i \mid x)$$

16

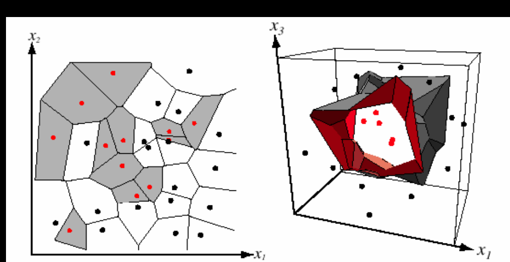


FIGURE 4.13. In two dimensions, the nearest-neighbor algorithm leads to a partitioning of the input space into Voronoi cells, each labeled by the category of the training point it contains. In three dimensions, the cells are three-dimensional, and the decision boundary resembles the surface of a crystal. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

17

- The k – nearest-neighbor rule
  - Goal: Classify x by assigning it the label most frequently represented among the k nearest samples and use a voting scheme

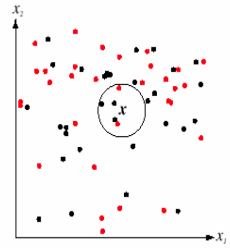


FIGURE 4.15. The k-nearest-neighbor query starts at the test point **x** and grows a spherical region until it encloses k training samples, and it labels the test point by a majority vote of these samples. In this $k = 5$ case, the test point **x** would be labeled the category of the black points. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Whitening Transform

See blackboard

Linear Discriminant Functions (Sections 5.1-5.2)

-

Linear discriminant functions and decisions surfaces

- Definition

  It is a function that is a linear combination of the components of x

  $$g(x) = w^t x + w_0 \qquad (1)$$

  where w is the weight vector and w_0 the bias

- A two-category classifier with a discriminant function of the form (1) uses the following rule:

  Decide ω_1 *if* $g(x) > 0$ and ω_2 if $g(x) < 0$

  $\Leftrightarrow$ Decide ω_1 *if* $w^t x > -w_0$ and ω_2 otherwise

  If $g(x) = 0 \Rightarrow x$ is assigned to either class

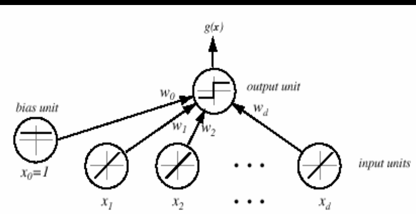


FIGURE 5.1. A simple linear classifier having d input units, each corresponding to the values of the components of an input vector. Each input feature value x_i is multiplied by its corresponding weight w_i; the effective input at the output unit is the sum all these products, $\sum w_i x_i$. We show in each unit its effective input-output function. Thus each of the d input units is linear, emitting exactly the value of its corresponding feature value. The single bias unit unit always emits the constant value 1.0. The single output unit emits a +1 if $\mathbf{w}^t\mathbf{x} + w_0 > 0$ or a −1 otherwise. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

- The equation $g(x) = 0$ defines the decision surface that separates points assigned to the category ω_1 from points assigned to the category ω_2

- When $g(x)$ is linear, the decision surface is a hyperplane

- Algebraic measure of the distance from x to the hyperplane (interesting result!)

24

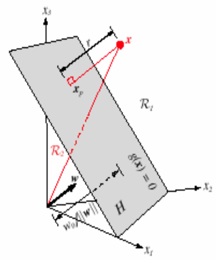


FIGURE 5.2. The linear decision boundary H, where $g(\mathbf{x}) = \mathbf{w}^t\mathbf{x}+w_0 = 0$, separates the feature space into two half-spaces $\mathcal{R}_1$ (where $g(\mathbf{x}) > 0$) and $\mathcal{R}_2$ (where $g(\mathbf{x}) < 0$). From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

25

$$x = x_p + \frac{r.w}{\|w\|} \quad (\text{since } w \text{ is colinear with } x - x_p \text{ and } \frac{w}{\|w\|} = 1)$$

$$\text{since } g(x) = 0 \text{ and } w^t.w = \|w\|^2$$

$$\text{therefore } r = \frac{g(x)}{\|w\|}$$

$$\text{in particular } d(0, H) = \frac{w_0}{\|w\|}$$

- In conclusion, a linear discriminant function divides the feature space by a hyperplane decision surface
- The orientation of the surface is determined by the normal vector w and the location of the surface is determined by the bias

26

- **The multi-category case**
  - We define c linear discriminant functions

$$g_i(x) = w_i^t x + w_{i0} \qquad i = 1,\ldots,c$$

and assign x to ω_i if $g_i(x) > g_j(x)$ $\forall j \neq i$; in case of ties, the classification is undefined

  - In this case, the classifier is a "linear machine"
  - A linear machine divides the feature space into c decision regions, with $g_i(x)$ being the largest discriminant if x is in the region R_i
  - For a two contiguous regions R_i and R_j; the boundary that separates them is a portion of hyperplane H_{ij} defined by:

$$g_i(x) = g_j(x)$$
$$\Leftrightarrow (w_i - w_j)^t x + (w_{i0} - w_{j0}) = 0$$

  - $w_i - w_j$ is normal to H_{ij} and

$$d(x, H_{ij}) = \frac{g_i - g_j}{\|w_i - w_j\|}$$

27

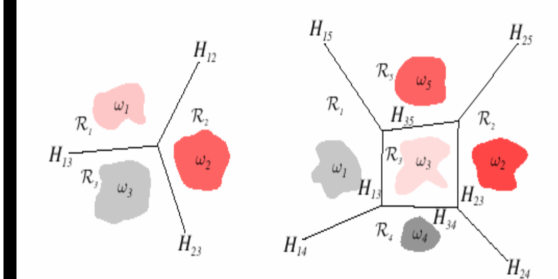
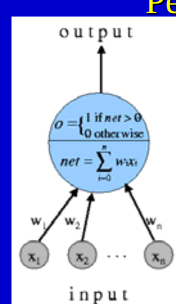


FIGURE 5.4. Decision boundaries produced by a linear machine for a three-class problem and a five-class problem. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

28

- It is easy to show that the decision regions for a linear machine are convex, this restriction limits the flexibility and accuracy of the classifier

Perceptron



Linear, threshold units

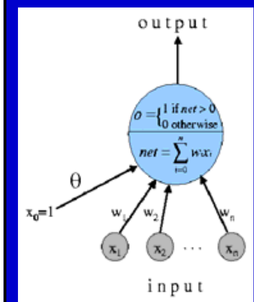
$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_1x_1 + \cdots + w_nx_n > \theta \\ -1 & \text{otherwise.} \end{cases}$$

X_i : inputs

W_i : weights

θ : threshold

Perceptron



The threshold can be easily forced to 0 by introducing an additional weight input $W_0 = \theta$

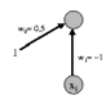
$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

How powerful is a perceptron?

Threshold = 0

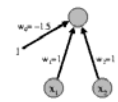
Inverter

input x1	output
0	1
1	0



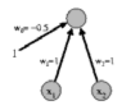
Boolean AND

input x1	input x2	ouput
0	0	0
0	1	0
1	0	0
1	1	1



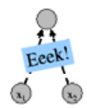
Boolean OR

input x1	input x2	ouput
0	0	0
0	1	1
1	0	1
1	1	1



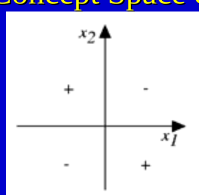
Boolean XOR

input x1	input x2	ouput
0	0	0
0	1	1
1	0	1
1	1	0

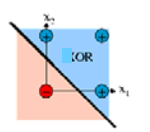


31

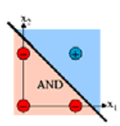
Concept Space & Linear Separability



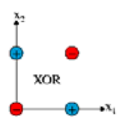
Linear Separability



Linear Separability



Linear Separability



32

Training Perceptron

Perceptron Training Rule

$$\underbrace{w_i}_{\text{new weight}} \leftarrow \underbrace{w_i}_{\text{old weight}} + \underbrace{\Delta w_i}_{\text{increment}}$$

$$\underbrace{\Delta w_i}_{\text{increment}} = \underbrace{\eta}_{\text{step size}} (\underbrace{t}_{\text{target}} - \underbrace{o}_{\text{perceptron output}}) \underbrace{x_i}_{\text{input}}$$

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

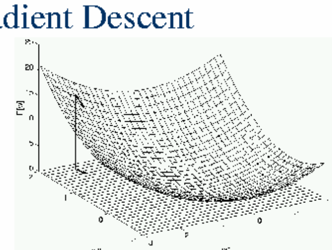
- Learn w_i's that minimize squared error

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

D = training data

34

Gradient Descent



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

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

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

Gradient Descent

- To find the best direction in the feature space we compute the gradient of E with respect to each of the components of $\vec{w}$

$$\nabla E(\vec{w}) \equiv [\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n}]$$

- This vector specifies the direction the produces the steepest increase in E;
- We want to modify $\vec{w}$ in the direction of $-\nabla E(\vec{w})$

$$\vec{w} = \vec{w} + \Delta\vec{w}$$

- Where:

$$\Delta\vec{w} = -R\,\nabla E(\vec{w})$$

Batch Learning

- Initialize each w_i to small random value
- Repeat until termination:

  $\Delta w_i = 0$

  For each training example d do

  $o_d \leftarrow \sigma(\Sigma_i\, w_i\, x_{i,d})$

  $\Delta w_i \leftarrow \Delta w_i + \eta\,(t_d - o_d)\; o_d\;(1\text{-}o_d)\; x_{i,d}$

  $w_i \leftarrow w_i + \Delta w_i$

Incremental (Online) Learning

- Initialize each w_i to small random value
- Repeat until termination:

  For each training example d do

  $\Delta w_i = 0$

  $o_d \leftarrow \Sigma_i\, w_i\, x_{i,d}$

  $\Delta w_i \leftarrow \Delta w_i + \eta\,(t_d - o_d)\; o_d\;(1\text{-}o_d)\; x_{i,d}$

  $w_i \leftarrow w_i + \Delta w_i$

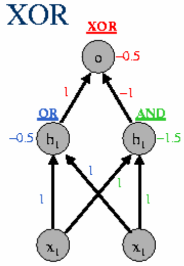
Summary: Single Layer Networks

- Variety of update rules
  - Multiplicative $\Delta w_i = R(t_d - o_d)x_{id}$
  - 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

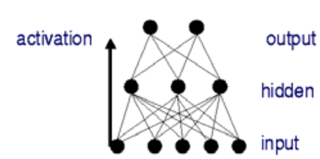
input x1	input x2	ouput
0	0	0
0	1	1
1	0	1
1	1	0



2-layer Neural Net

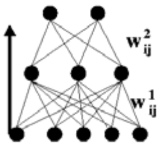
Multi-Layer Neural Network

- Multi-layer networks can represent arbitrary functions, but building effective learning methods for such network was thought to be difficult.
- Networks are composed of an input layer, hidden layer(s) and output layer. Activation is feed-forward from input to output



Multi-Layer Neural Network

- Patterns of activation are presented at the inputs and the resulting activation of the outputs is computed.
- The values of the weights determine the function computed. A network with one hidden layer is sufficient to represent every Boolean function. With real weights every real valued function can be approximated with a single hidden later.



Basic Unit in Multi-Layer Neural Network

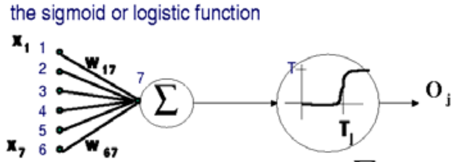
- Linear Unit: $o_j = \vec{w} \bullet \vec{x}$ multiple layers of linear functions produce linear functions. We want to represent nonlinear functions
- Threshold units $o_j = \mathrm{sgn}(\vec{w} \bullet \vec{x} - T)$ are not differentiable, hence unsuitable for gradient descent
- Us a non-linear, differentiable output function such as the sigmoid (or logistic) function

$$O_j = \frac{1}{1 + e^{-(w \bullet x - T_j)}}$$

43

Model Neuron (Logistic)

- Us a non-linear, differentiable output function such as the sigmoid or logistic function



- Net input to a unit is defined as: $\mathrm{net}_j = \sum w_{ij} \bullet x_i$
- Output of a unit is defined as:

$$O_j = \frac{1}{1 + e^{-(\mathrm{net}_j - T_j)}}$$

Representational Power

- The Backpropagation version presented is for networks with a single hidden layer,

But:

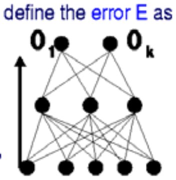
- Any Boolean function can be represented by a two layer network (simulate a two layer AND-OR network)
- Any bounded continuous function can be approximated with arbitrary small error by a two layer network. Sigmoid functions provide a set of basis function from which arbitrary function can be composed.
- Any function can be approximated to arbitrary accuracy by a three layer network.

Backpropagation Learning Rule

- Since there are multiple output units, we define the error E as the sum over all the network output units.

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in K} (t_{kd} - o_{kd})^2$$

where D is the set of training examples, K is the set of output units



(1,0,1,0,0)
(1,0,1,1,1)

This can be used to derive the (global) learning rule which performs gradient descent in the weight space in an attempt to minimize the error function.

$$\Delta w_{ij} = -R \frac{\partial E}{\partial w_{ij}}$$