

CS 559: Machine Learning Fundamentals and Applications

13th Set of Notes (Recap of Sets 8-12)

Instructor: Philippos Mordohai
Webpage: www.cs.stevens.edu/~mordohai
E-mail: Philippos.Mordohai@stevens.edu
Office: Lieb 215

Outline

- All topics covered in 7th set of Notes (midterm recap)
- **Non-parametric Classification**
- Linear Discriminant Functions
- Support Vector Machines (SVM)
- Ensemble Methods
- Unsupervised Learning
- Other notes

Density Estimation

- Basic idea:
- Probability that a vector x will fall in region R is:

$$P = \int_{\mathcal{R}} p(x') dx' \quad (1)$$

- P is a smoothed (or averaged) version of the density function $p(x)$ if we have a sample of size n ; therefore, the probability that k points fall in R is then:

$$P_k = \binom{n}{k} P^k (1-P)^{n-k} \quad (2)$$

and the expected value for k is:

$$E(k) = nP \quad (3)$$

ML estimation of $P = \theta$

$\text{Max}_{\theta}(P_k / \theta)$ is reached for $\hat{\theta} = \frac{k}{n} \cong P$

Therefore, the ratio k/n is a good estimate for the probability P and hence for the density function p (for large n)

$p(x)$ is continuous and the region R is so small that p does not vary significantly within it, we can write:

$$\int_{\mathfrak{R}} p(x') dx' \cong p(x) V \quad (4)$$

where x is a point within R and V the volume enclosed by R .

$$p_n(x) = (k_n/n)/V_n$$

There are two different ways of obtaining sequences of regions that satisfy the conditions for converging to the true probability given a large number of samples.

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

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

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”

Parzen Windows

- The Parzen-window approach to estimate densities assumes that the region \mathcal{R}_n is a d-dimensional hypercube

$$V_n = h_n^d \text{ (} h_n \text{ : length of the edge of } \mathcal{R}_n \text{)}$$

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

– 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:

$$\mathbf{p}_n(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{i=n} \frac{1}{\mathbf{v}_n} \varphi\left(\frac{\mathbf{x} - \mathbf{x}_i}{\mathbf{h}_n}\right)$$

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

Window Functions

- Conditions for estimating legitimate density function
 - Non-negative $\varphi(x) \geq 0$
 - Integrate to 1

$$\int \varphi(x) dx = 1$$

- In other words, the window function should be a probability density function

Illustration

- The behavior of the Parzen-window method
 - Case where $p(x) \rightarrow N(0, 1)$
 - Let $\varphi(u) = (1/\sqrt{2\pi}) \exp(-u^2/2)$ and $h_n = h_1/\sqrt{n}$ ($n > 1$)
 - (h_1 : known parameter)

Thus:

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

is an average of normal densities centered at the samples x_i

K - 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 grow until it captures k_n samples ($k_n = f(n)$)
 - k_n are called the k_n nearest-neighbors of x
- **Benefits**
 - If density is high near x , the cell will be small which provides a good resolution
 - If density is low, the cell will grow large and stop when higher density regions are reached

We can obtain a family of estimates by setting $k_n = k_1/\sqrt{n}$ and choosing different values for k_1

Estimation of Posterior Probabilities

- **Goal:** estimate $P(\omega_i / \mathbf{x})$ from a set of n labeled samples
- Place a cell of volume V around \mathbf{x} and capture k samples
- k_i samples amongst k turned out to be labeled ω_i then:

$$p_n(\mathbf{x}, \omega_i) = k_i / nV$$

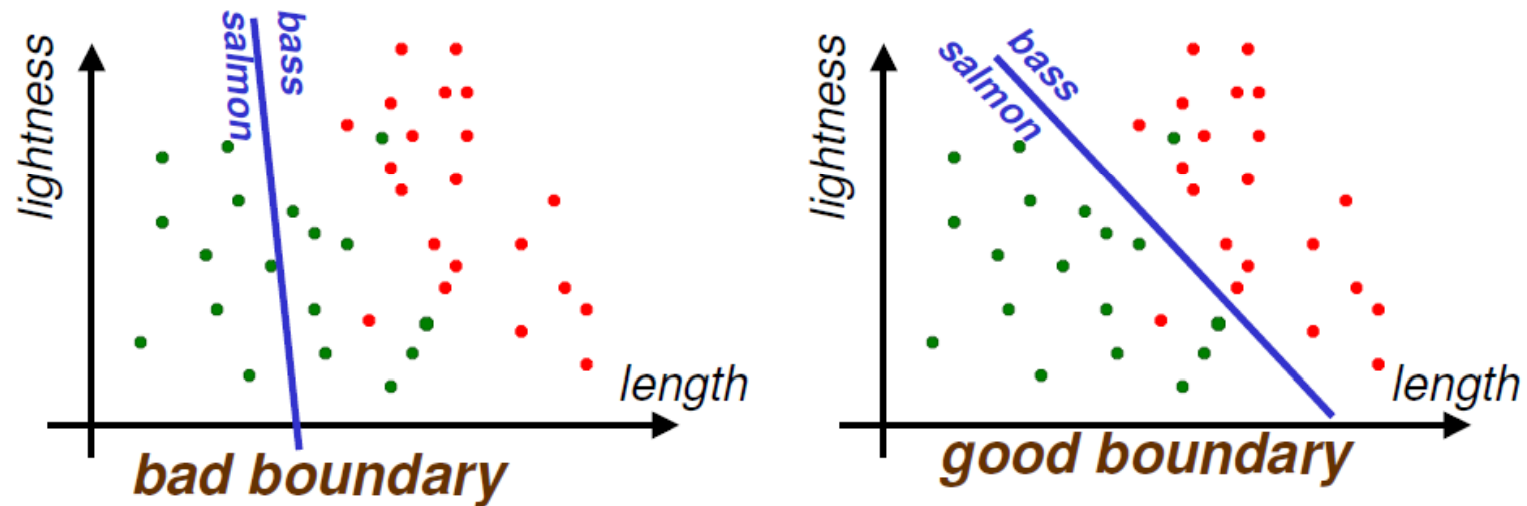
An estimate for $p_n(\omega_i / \mathbf{x})$ is:

$$p_n(\omega_i / \mathbf{x}) = \frac{p_n(\mathbf{x}, \omega_i)}{\sum_{j=1}^c p_n(\mathbf{x}, \omega_j)} = \frac{k_i}{k}$$

Outline

- All topics covered in 7th set of Notes (midterm recap)
- Non-parametric Classification
- **Linear Discriminant Functions**
- Support Vector Machines (SVM)
- Ensemble Methods
- Unsupervised Learning
- Other notes

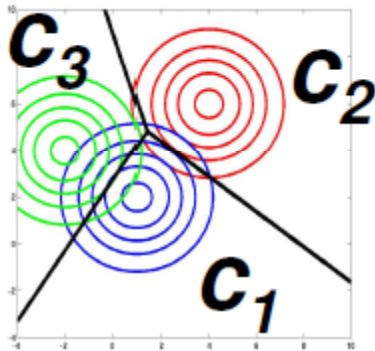
Linear Discriminant Functions: Basic Idea



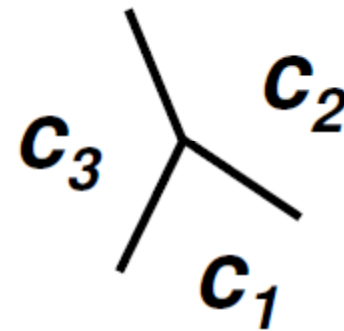
- Have samples from 2 classes x_1, x_2, \dots, x_n
- Assume 2 classes can be separated by a linear boundary $l(q)$ with some unknown parameters q
- Fit the “best” boundary to data by optimizing over parameters q
- What is best?
 - Minimize classification error on training data?
 - Does not guarantee small testing error

Parametric Methods vs. Discriminant Functions

- Assume the shape of density for classes is known $p_1(x|\theta_1), p_2(x|\theta_2), \dots$
- Estimate $\theta_1, \theta_2, \dots$ from data
- Use a Bayesian classifier to find decision regions



- Assume discriminant functions are of known shape $l(\theta_1), l(\theta_2), \dots$, with parameters $\theta_1, \theta_2, \dots$
- Estimate $\theta_1, \theta_2, \dots$ from data
- Use discriminant functions for classification

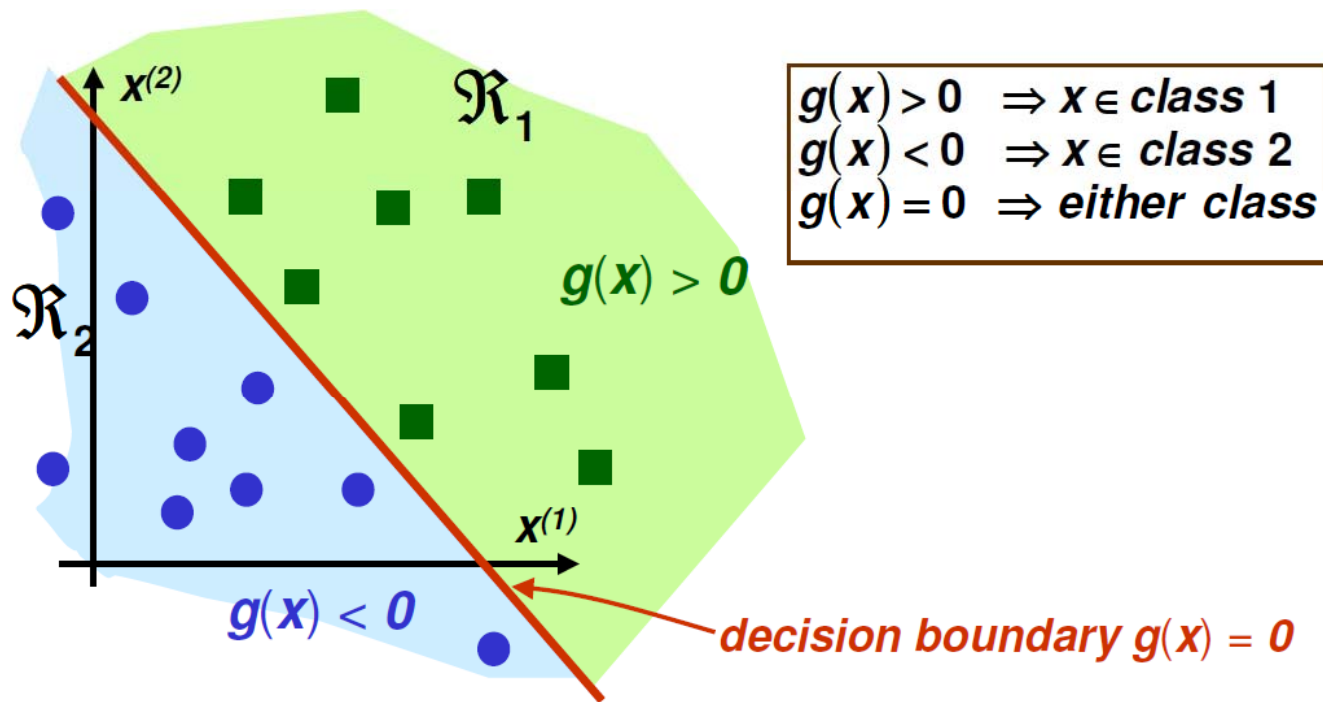


LDF: Two Classes

- A discriminant function is linear if it can be written as

$$g(x) = w^T x + w_0$$

- w is called the weight vector and w_0 is called the bias or threshold



LDF: Multiple Classes

- Suppose we have m classes
- Define m linear discriminant functions

$$g_i(x) = w_i^t x + w_{i0}$$

- Given x , assign to class c_i if
 - $g_i(x) > g_j(x), i \neq j$
- Such a classifier is called 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

LDF: Multiple Classes

- For two contiguous regions R_i and R_j , the boundary that separates them is a portion of the hyperplane H_{ij} defined by:

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

- Thus $w_i - w_j$ is normal to H_{ij}
- The distance from x to H_{ij} is given by:

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

Augmented Feature Vector

- Linear discriminant function: $g(x) = w^t x + w_0$

- Can rewrite it:

$$g(x) = \underbrace{[w_0 \quad w^t]}_{\substack{\text{new weight} \\ \text{vector } a}} \underbrace{\begin{bmatrix} 1 \\ x \end{bmatrix}}_{\substack{\text{new feature} \\ \text{vector } y}} = a^t y = g(y)$$

- y is called the **augmented feature** vector
- Added a dummy dimension to get a completely equivalent new **homogeneous problem**

old problem

$$g(x) = w^t x + w_0$$

$$\begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$$

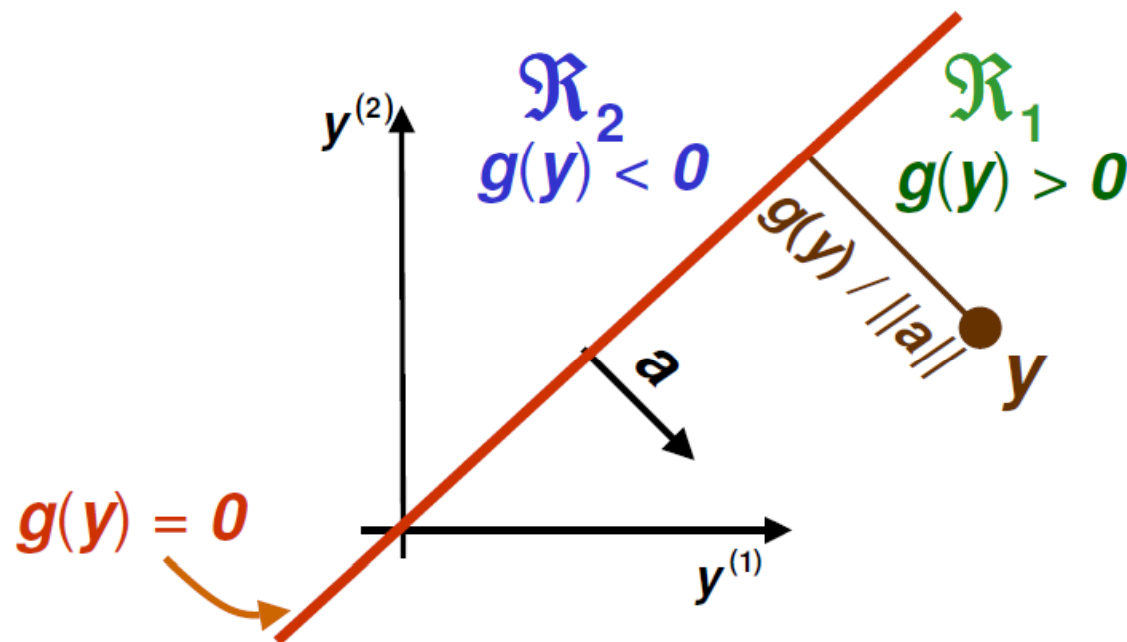
new problem

$$g(y) = a^t y$$

$$\begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}$$

- Feature augmentation is done for simpler notation
- From now on, always assume that we have augmented feature vectors
 - Given samples x_1, \dots, x_n convert them to augmented samples y_1, \dots, y_n by adding a new dimension of value 1

$$y_i = \begin{bmatrix} 1 \\ x_i \end{bmatrix}$$

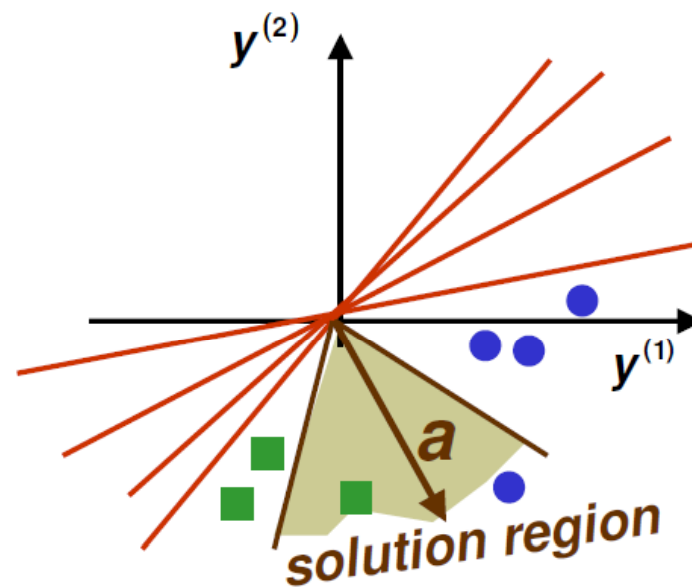


“Normalization”

- Training error is 0 if:
$$\begin{cases} \mathbf{a}^t \mathbf{y}_i > 0 & \forall \mathbf{y}_i \in \mathbf{c}_1 \\ \mathbf{a}^t \mathbf{y}_i < 0 & \forall \mathbf{y}_i \in \mathbf{c}_2 \end{cases}$$
- Equivalently, training error is 0 if:
$$\begin{cases} \mathbf{a}^t \mathbf{y}_i > 0 & \forall \mathbf{y}_i \in \mathbf{c}_1 \\ \mathbf{a}^t (-\mathbf{y}_i) > 0 & \forall \mathbf{y}_i \in \mathbf{c}_2 \end{cases}$$
- This suggests “normalization” (a.k.a. reflection):
 1. Replace all examples from class 2 by:
$$\mathbf{y}_i \rightarrow -\mathbf{y}_i \quad \forall \mathbf{y}_i \in \mathbf{c}_2$$
 2. Seek weight vector \mathbf{a} such that
$$\mathbf{a}^t \mathbf{y}_i > 0 \quad \forall \mathbf{y}_i$$
 - If such \mathbf{a} exists, it is called a separating or solution vector
 - Original samples $\mathbf{x}_1, \dots, \mathbf{x}_n$ can indeed be separated by a line

Solution Region

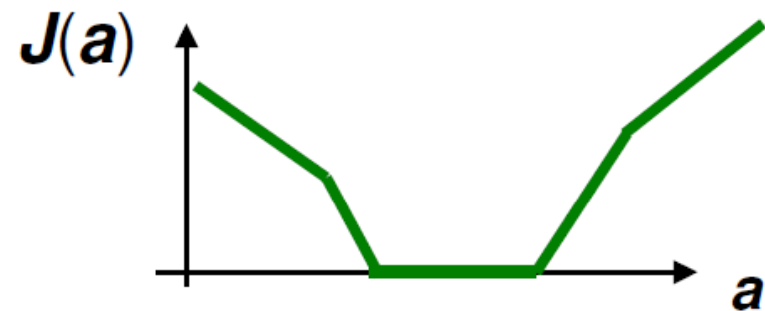
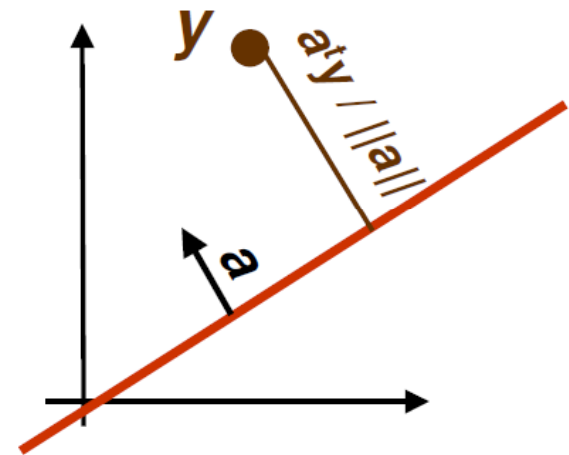
- **Solution region** for \mathbf{a} : set of all possible solutions defined in terms of normal \mathbf{a} to the separating hyperplane



Perceptron Criterion Function

$$J_p(\mathbf{a}) = \sum_{y \in Y_M} (-\mathbf{a}^t \mathbf{y})$$

- If \mathbf{y} is misclassified, $\mathbf{a}^t \mathbf{y} < 0$
- Thus $J_p(\mathbf{a}) > 0$
- $J_p(\mathbf{a})$ is $\|\mathbf{a}\|$ times sum of distances of misclassified examples to decision boundary
- $J_p(\mathbf{a})$ is piecewise linear and thus suitable for gradient descent



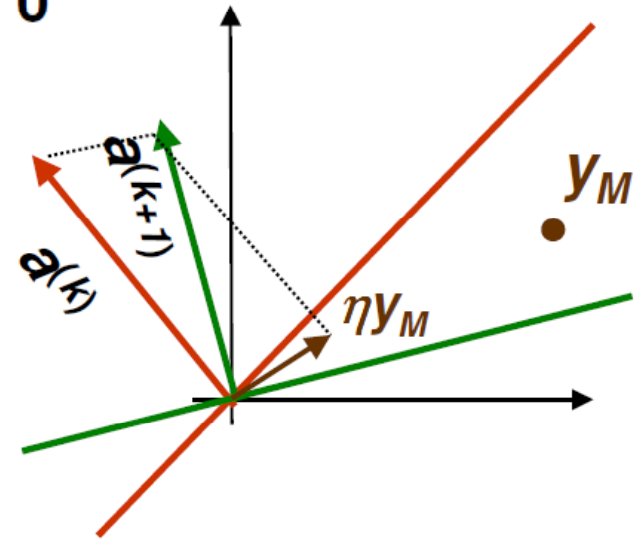
Perceptron Batch Rule

$$J_p(\mathbf{a}) = \sum_{y \in Y_M} (-\mathbf{a}^t \mathbf{y})$$

- Gradient of $J_p(\mathbf{a})$ is: $\nabla J_p(\mathbf{a}) = \sum_{y \in Y_M} (-\mathbf{y})$
 - Y_M are samples misclassified by $\mathbf{a}^{(k)}$
 - It is not possible to solve $\nabla J_p(\mathbf{a}) = \mathbf{0}$ analytically because of Y_M
- Update rule for gradient descent: $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta^{(k)} \nabla J(\mathbf{x})$
- Thus the *gradient decent batch update rule* for $J_p(\mathbf{a})$ is:
$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \eta^{(k)} \sum_{y \in Y_M} \mathbf{y}$$
- It is called batch rule because it is based on all misclassified examples

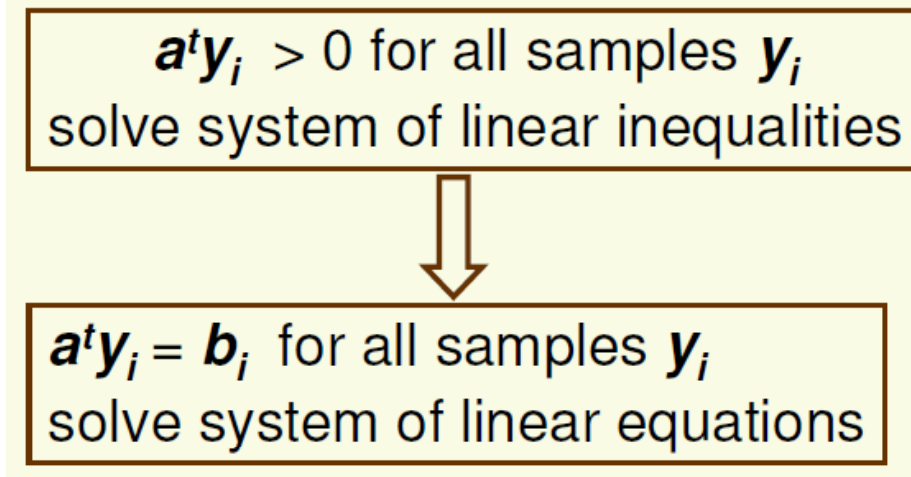
Perceptron Single Sample Rule

- The *gradient decent single sample rule* for $J_p(\mathbf{a})$ is:
$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \eta^{(k)} \mathbf{y}_M$$
 - Note that \mathbf{y}_M is one sample misclassified by $\mathbf{a}^{(k)}$
 - Must have a consistent way of visiting samples
- Geometric Interpretation:
 - \mathbf{y}_M misclassified by $\mathbf{a}^{(k)}$ $(\mathbf{a}^{(k)})^t \mathbf{y}_M \leq 0$
 - \mathbf{y}_M is on the wrong side of decision hyperplane
 - Adding $\eta \mathbf{y}_M$ to \mathbf{a} moves the new decision hyperplane in the right direction with respect to \mathbf{y}_M



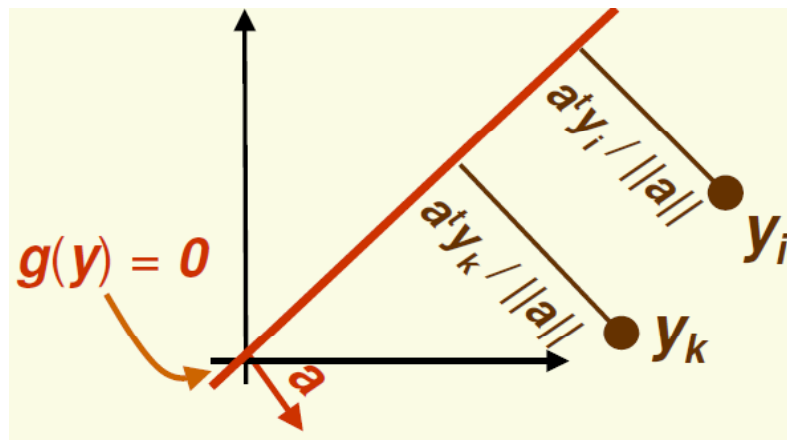
Minimum Squared-Error Procedures

- Idea: convert to easier and better understood problem



- MSE procedure
 - Choose positive constants $\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n$
 - Try to find weight vector \mathbf{a} such that $\mathbf{a}^t \mathbf{y}_i = \mathbf{b}_i$ for all samples \mathbf{y}_i
 - If we can find such a vector, then \mathbf{a} is a solution because the \mathbf{b}_i 's are positive
 - Consider all the samples (not just the misclassified ones)

MSE Margins



- Since we want $\mathbf{a}^T \mathbf{y}_i = b_i$, we expect \mathbf{y}_i to be at distance b_i from the separating hyperplane (normalized by $\|\mathbf{a}\|$)
- Thus b_1, b_2, \dots, b_n give relative expected distances or “margins” of samples from the hyperplane
- Should make b_i small if sample i is expected to be near separating hyperplane, and large otherwise
- In the absence of any additional information, set $b_1 = b_2 = \dots = b_n = 1$

MSE Matrix Notation

- Need to solve n equations
- In matrix form $Y\mathbf{a}=\mathbf{b}$

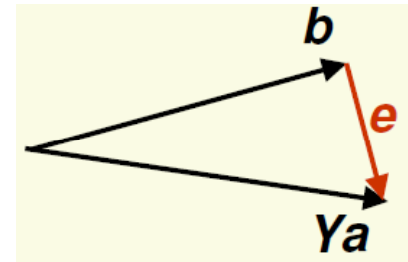
$$\begin{cases} \mathbf{a}^t \mathbf{y}_1 = b_1 \\ \vdots \\ \mathbf{a}^t \mathbf{y}_n = b_n \end{cases}$$

$$\underbrace{\begin{bmatrix} \mathbf{y}_1^{(0)} & \mathbf{y}_1^{(1)} & \dots & \mathbf{y}_1^{(d)} \\ \mathbf{y}_2^{(0)} & \mathbf{y}_2^{(1)} & \dots & \mathbf{y}_2^{(d)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{y}_n^{(0)} & \mathbf{y}_n^{(1)} & \dots & \mathbf{y}_n^{(d)} \end{bmatrix}}_{\mathbf{Y}} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix}}_{\mathbf{a}} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}}_{\mathbf{b}}$$

MSE Criterion Function

- Minimum squared error approach: find \mathbf{a} which minimizes the length of the error vector \mathbf{e}

$$\mathbf{e} = \mathbf{Y}\mathbf{a} - \mathbf{b}$$



- Thus minimize the **minimum squared error criterion** function:

$$J_s(\mathbf{a}) = \|\mathbf{Y}\mathbf{a} - \mathbf{b}\|^2 = \sum_{i=1}^n (\mathbf{a}^t \mathbf{y}_i - b_i)^2$$

- Unlike the perceptron criterion function, we can optimize the minimum squared error criterion function analytically by setting the gradient to 0

Pseudo-Inverse Solution

$$\nabla J_s(\mathbf{a}) = 2\mathbf{Y}^t(\mathbf{Y}\mathbf{a} - \mathbf{b})$$

- Setting the gradient to 0:

$$2\mathbf{Y}^t(\mathbf{Y}\mathbf{a} - \mathbf{b}) = 0 \Rightarrow \mathbf{Y}^t\mathbf{Y}\mathbf{a} = \mathbf{Y}^t\mathbf{b}$$

- The matrix $\mathbf{Y}^t\mathbf{Y}$ is square (it has $d + 1$ rows and columns) and it is often non-singular
- If $\mathbf{Y}^t\mathbf{Y}$ is non-singular, its inverse exists and we can solve for \mathbf{a} uniquely:

$$\mathbf{a} = \boxed{(\mathbf{Y}^t\mathbf{Y})^{-1}\mathbf{Y}^t}\mathbf{b}$$

pseudo inverse of \mathbf{Y}
$$((\mathbf{Y}^t\mathbf{Y})^{-1}\mathbf{Y}^t)\mathbf{Y} = (\mathbf{Y}^t\mathbf{Y})^{-1}(\mathbf{Y}^t\mathbf{Y}) = \mathbf{I}$$

MSE Procedures

- Only guaranteed separating hyperplane if $Y\mathbf{a} > 0$
 - That is if all elements of vector $Y\mathbf{a}$ are positive

$$Y\mathbf{a} = \begin{bmatrix} b_1 + \varepsilon_1 \\ \vdots \\ b_n + \varepsilon_n \end{bmatrix}$$

- where ε may be negative
- If $\varepsilon_1, \dots, \varepsilon_n$ are small relative to b_1, \dots, b_n , then each element of $Y\mathbf{a}$ is positive, and \mathbf{a} gives a separating hyperplane
 - If the approximation is not good, ε_i may be large and negative, for some i , thus $b_i + \varepsilon_i$ will be negative and \mathbf{a} is not a separating hyperplane
- In linearly separable case, least squares solution \mathbf{a} does not necessarily give separating hyperplane

Gradient Descent for MSE

$$J_s(\mathbf{a}) = \|\mathbf{Y}\mathbf{a} - \mathbf{b}\|^2$$

- May wish to find MSE solution by gradient descent:
 1. Computing the inverse of $\mathbf{Y}^t\mathbf{Y}$ may be too costly
 2. $\mathbf{Y}^t\mathbf{Y}$ may be close to singular if samples are highly correlated (rows of \mathbf{Y} are almost linear combinations of each other) computing the inverse of $\mathbf{Y}^t\mathbf{Y}$ is not numerically stable
- The gradient is:

$$\nabla J_s(\mathbf{a}) = 2\mathbf{Y}^t(\mathbf{Y}\mathbf{a} - \mathbf{b})$$

Widrow-Hoff Procedure

$$\nabla J_s(\mathbf{a}) = 2Y^t(Y\mathbf{a} - \mathbf{b})$$

- Thus the update rule for gradient descent is:

$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} - \eta^{(k)} Y^t(Y\mathbf{a}^{(k)} - \mathbf{b})$$

- If $\eta^{(k)} = \eta^{(1)}/k$, then $\mathbf{a}^{(k)}$ converges to the MSE solution \mathbf{a} , that is $Y^t(Y\mathbf{a} - \mathbf{b}) = 0$
- The *Widrow-Hoff procedure* reduces storage requirements by considering single samples sequentially

$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} - \eta^{(k)} \mathbf{y}_i (\mathbf{y}_i^t \mathbf{a}^{(k)} - b_i)$$

Ho-Kashyap Procedure

- If \mathbf{b} is chosen arbitrarily, finding a separating hyperplane is not guaranteed
- Suppose training samples are linearly separable. Then there is \mathbf{a}^s and positive b^s s.t.

$$Y\mathbf{a}^s = b^s > 0$$

- If we knew b^s , we could apply MSE procedure to find the separating hyperplane.
- Find both \mathbf{a}^s and b^s
- Minimize the following criterion function, restricting to positive b :

$$J_{HK}(\mathbf{a}, b) = \|\mathbf{Y}\mathbf{a} - \mathbf{b}\|^2$$
$$J_{HK}(\mathbf{a}^s, b^s) = 0$$

Ho-Kashyap Procedure

$$J_{HK}(\mathbf{a}, \mathbf{b}) = \|\mathbf{Y}\mathbf{a} - \mathbf{b}\|^2$$

- Partial derivatives w.r.t. \mathbf{a} and \mathbf{b}

$$\nabla_{\mathbf{a}} J_{HK} = 2\mathbf{Y}^t(\mathbf{Y}\mathbf{a} - \mathbf{b}) = \mathbf{0}$$

$$\nabla_{\mathbf{b}} J_{HK} = -2(\mathbf{Y}\mathbf{a} - \mathbf{b}) = \mathbf{0}$$

- Use modified gradient descent procedure to find a minimum of $J_{HK}(\mathbf{a}, \mathbf{b})$
 - Alternate two steps until convergence:
 1. Fix \mathbf{b} and minimize $J_{HK}(\mathbf{a}, \mathbf{b})$ with respect to \mathbf{a}
 2. Fix \mathbf{a} and minimize $J_{HK}(\mathbf{a}, \mathbf{b})$ with respect to \mathbf{b}

Ho-Kashyap Procedure

- Step (1) can be performed using the pseudoinverse
- For fixed \mathbf{b} , the minimum of $J_{HK}(\mathbf{a}, \mathbf{b})$ with respect to \mathbf{a} is found by solving

$$2\mathbf{Y}^t(\mathbf{Y}\mathbf{a} - \mathbf{b}) = 0$$

$$\mathbf{a} = (\mathbf{Y}^t\mathbf{Y})^{-1}\mathbf{Y}^t\mathbf{b}$$

Ho-Kashyap Procedure

- Step 2: fix \mathbf{a} and minimize $J_{HK}(\mathbf{a}, \mathbf{b})$ with respect to \mathbf{b}
- We cannot use $\mathbf{b} = Y\mathbf{a}$ because \mathbf{b} has to be positive
- Solution: use modified gradient descent
 - Regular gradient descent rule:
$$\mathbf{b}^{(k+1)} = \mathbf{b}^{(k)} - \eta^{(k)} \nabla_{\mathbf{b}} J(\mathbf{a}^{(k)}, \mathbf{b}^{(k)})$$
 - If any components of the gradient are positive, \mathbf{b} will decrease and may become negative

$$\mathbf{b}^{(k+1)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - 2 * \begin{bmatrix} 2 \\ -3 \\ -2 \end{bmatrix} = \begin{bmatrix} -3 \\ 7 \\ 5 \end{bmatrix}$$

Ho-Kashyap Procedure

- Start with positive \mathbf{b} , follow negative gradient but refuse to decrease any components of \mathbf{b}
 - This can be achieved by setting all the positive components of the gradient to 0

$$\mathbf{b}^{(k+1)} = \mathbf{b}^{(k)} - \eta \frac{1}{2} [\nabla_{\mathbf{b}} \mathbf{J}(\mathbf{a}^{(k)}, \mathbf{b}^{(k)}) - |\nabla_{\mathbf{b}} \mathbf{J}(\mathbf{a}^{(k)}, \mathbf{b}^{(k)})|]$$

- $|\mathbf{v}|$ denotes the vector we get after applying absolute value to all elements of \mathbf{v}
- Not doing steepest descent anymore, but we are still doing descent while ensuring that \mathbf{b} is positive

$$\mathbf{b}^{(k+1)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - 2 * \frac{1}{2} \left[\begin{bmatrix} 2 \\ -3 \\ -2 \end{bmatrix} - \begin{bmatrix} 2 \\ 3 \\ 2 \end{bmatrix} \right] = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ -6 \\ -4 \end{bmatrix} = \begin{bmatrix} 1 \\ 7 \\ 5 \end{bmatrix}$$

Ho-Kashyap Procedure

$$\mathbf{b}^{(k+1)} = \mathbf{b}^{(k)} - \eta \frac{1}{2} [\nabla_b \mathbf{J}(\mathbf{a}^{(k)}, \mathbf{b}^{(k)}) - |\nabla_b \mathbf{J}(\mathbf{a}^{(k)}, \mathbf{b}^{(k)})|]$$

$$\nabla_b \mathbf{J} = -2(\mathbf{Y}\mathbf{a} - \mathbf{b}) = 0$$

- Let $\mathbf{e}^{(k)} = \mathbf{Y}\mathbf{a}^{(k)} - \mathbf{b}^{(k)} = -\frac{1}{2} \nabla \mathbf{J}_b(\mathbf{a}^{(k)}, \mathbf{b}^{(k)})$

- Then
$$\begin{aligned} \mathbf{b}^{(k+1)} &= \mathbf{b}^{(k)} - \eta \frac{1}{2} [-2\mathbf{e}^{(k)} - |2\mathbf{e}^{(k)}|] \\ &= \mathbf{b}^{(k)} + \eta [\mathbf{e}^{(k)} + |\mathbf{e}^{(k)}|] \end{aligned}$$

Ho-Kashyap Procedure

- The final Ho-Kashyap procedure:

(0) Start with arbitrary $\mathbf{a}^{(1)}$ and $\mathbf{b}^{(1)} > \mathbf{0}$, let $k = 1$

repeat steps (1) through (4)

(1) $\mathbf{e}^{(k)} = \mathbf{Y}\mathbf{a}^{(k)} - \mathbf{b}^{(k)}$

(2) Solve for $\mathbf{b}^{(k+1)}$ using $\mathbf{a}^{(k)}$ and $\mathbf{b}^{(k)}$

$$\mathbf{b}^{(k+1)} = \mathbf{b}^{(k)} + \eta [\mathbf{e}^{(k)} + \|\mathbf{e}^{(k)}\| \mathbf{1}]$$

(3) Solve for $\mathbf{a}^{(k+1)}$ using $\mathbf{b}^{(k+1)}$

$$\mathbf{a}^{(k+1)} = (\mathbf{Y}^t \mathbf{Y})^{-1} \mathbf{Y}^t \mathbf{b}^{(k+1)}$$

(4) $k = k + 1$

until $\mathbf{e}^{(k)} \geq \mathbf{0}$ or $k > k_{max}$ or $\mathbf{b}^{(k+1)} = \mathbf{b}^{(k)}$

- For convergence, learning rate should be fixed between $0 < \eta < 1$

NEW

The three termination conditions are not equivalent, see following slides

Ho-Kashyap Procedure

$$\mathbf{b}^{(k+1)} = \mathbf{b}^{(k)} + \eta[\mathbf{e}^{(k)} + |\mathbf{e}^{(k)}|]$$

- What if $\mathbf{e}^{(k)}$ is negative for all components?
 - $\mathbf{b}^{(k+1)} = \mathbf{b}^{(k)}$ and corrections stop

- Write $\mathbf{e}^{(k)}$ out:

$$\mathbf{e}^{(k)} = \mathbf{Y}\mathbf{a}^{(k)} - \mathbf{b}^{(k)} = \mathbf{Y}(\mathbf{Y}^t\mathbf{Y})^{-1}\mathbf{Y}^t\mathbf{b}^{(k)} - \mathbf{b}^{(k)}$$

- Multiply by \mathbf{Y}^t :

$$\mathbf{Y}^t\mathbf{e}^{(k)} = \mathbf{Y}^t(\mathbf{Y}(\mathbf{Y}^t\mathbf{Y})^{-1}\mathbf{Y}^t\mathbf{b}^{(k)} - \mathbf{b}^{(k)}) = \mathbf{Y}^t\mathbf{b}^{(k)} - \mathbf{Y}^t\mathbf{b}^{(k)} = \mathbf{0}$$

- Thus $\mathbf{Y}^t\mathbf{e}^{(k)} = \mathbf{0}$

Ho-Kashyap Procedure

- If $\mathbf{e}^{(k)}$ has no positive components, how do we know that we have found solution?
- Suppose training samples are linearly separable. Then there is \mathbf{a}^s and positive b^s s.t.

$$Y\mathbf{a}^s = \mathbf{b}^s > 0$$

- Multiply both sides by $(\mathbf{e}^{(k)})^t$
$$(\mathbf{e}^{(k)})^t Y\mathbf{a}^s = ((\mathbf{e}^{(k)})^t Y)\mathbf{a}^s = 0 = (\mathbf{e}^{(k)})^t \mathbf{b}^s$$
- Either $\mathbf{e}^{(k)} = \mathbf{0}$ or some of its components are positive (contradiction)
 - All components of \mathbf{b}^s are positive

Ho-Kashyap Procedure

- In the linearly separable case:
 - $\mathbf{e}^{(k)} = \mathbf{0}$, found solution, stop
 - One of components of $\mathbf{e}^{(k)}$ is positive, algorithm continues
- In the nonseparable case:
 - $\mathbf{e}^{(k)}$ will have only negative components eventually, thus found proof of nonseparability
 - No bound on how many iterations needed for the proof of nonseparability

MSE for Multiple Classes

- We still use augmented feature vectors $\mathbf{y}_1, \dots, \mathbf{y}_n$

- Define m linear discriminant functions

$$g_i(\mathbf{y}) = \mathbf{a}_i^t \mathbf{y} \quad i = 1, \dots, m$$

- Given \mathbf{y} , assign to class c_i if:

$$\mathbf{a}_i^t \mathbf{y} \geq \mathbf{a}_j^t \mathbf{y} \quad \forall j \neq i$$

- For each class i , makes sense to seek weight vector \mathbf{a}_i s.t.

$$\begin{cases} \mathbf{a}_i^t \mathbf{y} = 1 & \forall \mathbf{y} \in \text{class } i \\ \mathbf{a}_i^t \mathbf{y} = 0 & \forall \mathbf{y} \notin \text{class } i \end{cases}$$

- If we find such $\mathbf{a}_1, \dots, \mathbf{a}_m$ the training error will be 0

MSE for Multiple Classes

- Stack all b_i as columns in n by c matrix B

$$B = [b_1 \ \dots \ b_n]$$

- Stack all a_i as columns in $d + 1$ by m matrix A

$$A = [a_1 \ \dots \ a_m] = \begin{bmatrix} \text{weights } a_1 \\ \text{weights } a_2 \\ \vdots \\ \text{weights } a_m \end{bmatrix}$$

- m LSE problems can be represented in $YA = B$:

$$\begin{bmatrix} \text{sample from class 1} \\ \text{sample from class 1} \\ \text{sample from class 2} \\ \text{sample from class 3} \\ \text{sample from class 3} \\ \text{sample from class 3} \end{bmatrix} \begin{bmatrix} \text{weights for c1} \\ \text{weights for c2} \\ \text{weights for c3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Y **A** **B**

MSE for Multiple Classes

- Our objective function is:

$$J(\mathbf{A}) = \sum_{i=1}^m \|\mathbf{Y}\mathbf{a}_i - \mathbf{b}_i\|^2$$

- $J(\mathbf{A})$ can be minimized using the pseudoinverse

$$\mathbf{A} = (\mathbf{Y}^t \mathbf{Y})^{-1} \mathbf{Y} \mathbf{B}$$

LDF Summary

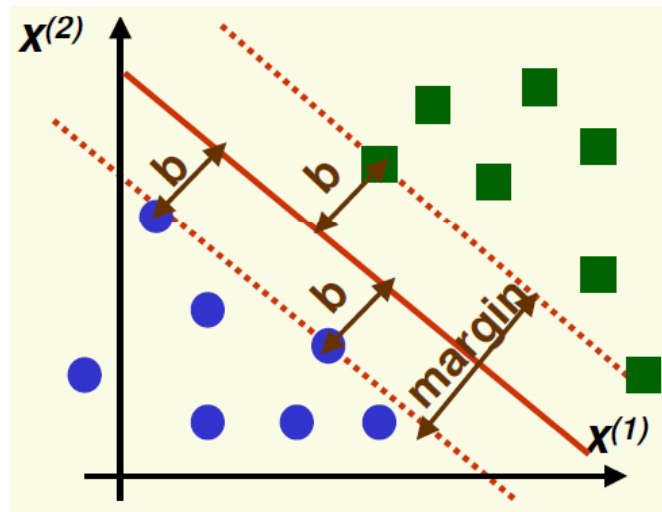
- **Perceptron procedures**
 - Find a separating hyperplane in the linearly separable case,
 - Do not converge in the non-separable case
 - Can force convergence by using a decreasing learning rate, but are not guaranteed a reasonable stopping point
- **MSE procedures**
 - Converge in separable and not separable case
 - May not find separating hyperplane if classes are linearly separable
 - Use pseudoinverse if $Y^T Y$ is not singular and not too large
 - Use gradient descent (Widrow-Hoff procedure) otherwise
- **Ho-Kashyap procedures**
 - Always converge
 - Find separating hyperplane in the linearly separable case
 - More costly

Outline

- All topics covered in 7th set of Notes (midterm recap)
- Non-parametric Classification
- Linear Discriminant Functions
- Support Vector Machines (SVM)
- Ensemble Methods
- Unsupervised Learning
- Other notes

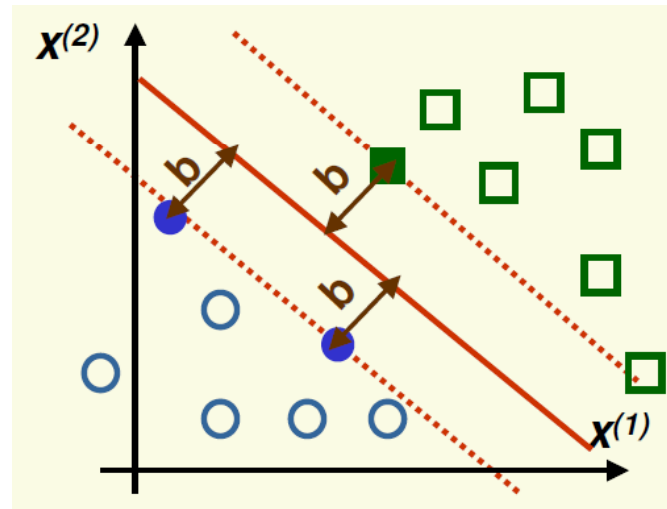
SVM: Linearly Separable Case

- SVM: maximize the margin



- The *margin* is twice the absolute value of distance b of the closest example to the separating hyperplane
- Better generalization (performance on test data)
 - in practice
 - and in theory

SVM: Linearly Separable Case



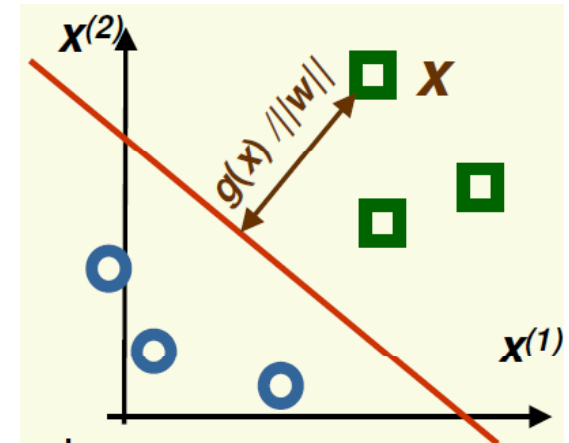
- ***Support vectors*** are the samples closest to the separating hyperplane
 - They are the most difficult patterns to classify
- Optimal hyperplane is completely defined by support vectors
 - Of course, we do not know which samples are support vectors without finding the optimal hyperplane

SVM: Formula for the Margin

$$g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0$$

- Absolute distance between \mathbf{x} and the boundary $g(\mathbf{x}) = 0$

$$\frac{|\mathbf{w}^t \mathbf{x} + w_0|}{\|\mathbf{w}\|}$$



- Distance is unchanged for hyperplane

$$g_1(\mathbf{x}) = \alpha g(\mathbf{x})$$

$$\frac{|\alpha \mathbf{w}^t \mathbf{x} + \alpha w_0|}{\|\alpha \mathbf{w}\|} = \frac{|\mathbf{w}^t \mathbf{x} + w_0|}{\|\mathbf{w}\|}$$

- Let \mathbf{x}_i be an example closest to the boundary. Set

$$|\mathbf{w}^t \mathbf{x}_i + w_0| = 1$$

- Now the largest margin hyperplane is unique

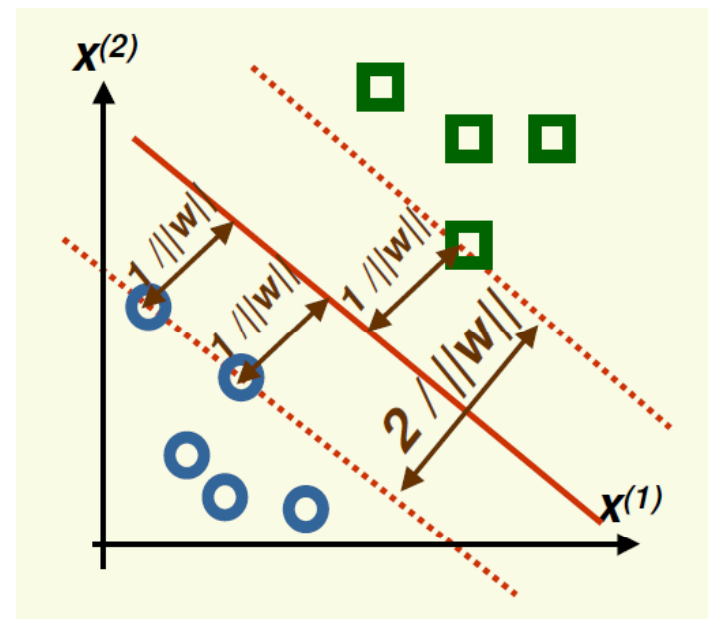
SVM: Formula for the Margin

- For uniqueness, set $|w^T x_i + w_0| = 1$ for any sample x_i closest to the boundary
- The distance from closest sample x_i to $g(x) = 0$ is

$$\frac{|w^T x_i + w_0|}{\|w\|} = \frac{1}{\|w\|}$$

- Thus the margin is

$$m = \frac{2}{\|w\|}$$



Nonlinear SVM

- Can use any linear classifier after lifting data to a higher dimensional space. However we will have to deal with the curse of dimensionality
 - Poor generalization to test data
 - Computationally expensive
- SVM avoids the curse of dimensionality problems
 - Enforcing largest margin permits good generalization
 - It can be shown that generalization in SVM is a function of the margin, independent of the dimensionality
 - Computation in the higher dimensional case is performed only implicitly through the use of *kernel functions*

Nonlinear SVM Step-by-Step

- Start with data x_1, \dots, x_n which live in feature space of dimension d
- Choose kernel $K(x_i, x_j)$ or function $\varphi(x_i)$ which lifts sample x_i to a higher dimensional space
- Find the largest margin linear discriminant function in the higher dimensional space by using quadratic programming package to solve:

$$\begin{aligned} &\text{maximize } L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j z_i z_j K(x_i, x_j) \\ &\text{constrained to } 0 \leq \alpha_i \leq \beta \quad \forall i \quad \text{and} \quad \sum_{i=1}^n \alpha_i z_i = 0 \end{aligned}$$

Nonlinear SVM Step-by-Step

- Weight vector w in the high dimensional space:

$$w = \sum_{x_i \in S} \alpha_i z_i \varphi(x_i)$$

– where S is the set of support vectors

- Linear discriminant function of largest margin in the high dimensional space:

$$g(\varphi(x)) = w^t \varphi(x) = \left(\sum_{x_i \in S} \alpha_i z_i \varphi(x_i) \right)^t \varphi(x)$$

- Non linear discriminant function in the original space:

$$g(x) = \left(\sum_{x_i \in S} \alpha_i z_i \varphi(x_i) \right)^t \varphi(x) = \sum_{x_i \in S} \alpha_i z_i \varphi^t(x_i) \varphi(x) = \sum_{x_i \in S} \alpha_i z_i K(x_i, x)$$

- decide class 1 if $g(x) > 0$, otherwise decide class 2

Nonlinear SVM

- Nonlinear discriminant function

$$g(\mathbf{x}) = \sum_{\mathbf{x}_i \in \mathcal{S}} \alpha_i z_i K(\mathbf{x}_i, \mathbf{x})$$

$$g(\mathbf{x}) = \sum \left[\begin{array}{|l} \text{weight of support} \\ \text{vector } \mathbf{x}_i \end{array} \right] \left[\begin{array}{|l} \mp 1 \end{array} \right] \left[\begin{array}{|l} \text{“inverse distance”} \\ \text{from } \mathbf{x} \text{ to} \\ \text{support vector } \mathbf{x}_i \end{array} \right]$$

most important training samples, i.e. support vectors

$$K(\mathbf{x}_i, \mathbf{x}) = \exp\left(-\frac{1}{2\sigma^2} \|\mathbf{x}_i - \mathbf{x}\|^2\right)$$

SVM Summary

- Advantages:
 - Based on nice theory
 - Excellent generalization properties
 - Objective function has no local minima
 - Can be used to find non linear discriminant functions
 - Complexity of the classifier is characterized by the number of support vectors rather than the dimensionality of the transformed space
- Disadvantages:
 - Tends to be slower than other methods
 - Quadratic programming is computationally expensive

Outline

- All topics covered in 7th set of Notes (midterm recap)
- Non-parametric Classification
- Linear Discriminant Functions
- Support Vector Machines (SVM)
- **Ensemble Methods**
- Unsupervised Learning
- Other notes

Ensemble Methods

- Bagging (Breiman 1994,...)
- Boosting (Freund and Schapire 1995, Friedman et al. 1998,...)
- Random forests (Breiman 2001,...)

Predict class label for unseen data by aggregating a set of predictions (classifiers learned from the training data).

Ensemble Classifiers

- Basic idea: Build different “experts” and let them vote
- Advantages:
 - Improve predictive performance
 - Different types of classifiers can be directly included
 - Easy to implement
 - Not too much parameter tuning
- Disadvantages:
 - The combined classifier is not transparent (black box)
 - Not a compact representation

Bagging

- Training
 - Given a dataset S , at each iteration i , a training set S_i is sampled with replacement from S (i.e. bootstrapping)
 - A classifier C_i is learned for each S_i
- Classification: given an unseen sample X
 - Each classifier C_i returns its class prediction
 - The bagged classifier H counts the votes and assigns the class with the most votes to X

Bagging

- Bagging works because it reduces variance by voting/averaging
 - In some pathological hypothetical situations the overall error might increase
 - Usually, the more classifiers the better
- Problem: we only have one dataset
- Solution: generate new ones of size n by bootstrapping, i.e. sampling with replacement
- Can help a lot if data is noisy

Boosting

- Idea: given a set of weak learners, run them multiple times on (reweighted) training data, then let learned classifiers vote
- At each iteration t :
 - Weight each training example by how incorrectly it was classified
 - Learn a hypothesis - h_t
 - Choose a strength for this hypothesis - α_t
- Final classifier: weighted combination of weak learners

Learning from Weighted Data

- Sometimes not all data points are equal
 - Some data points are more equal than others
- Consider a weighted dataset
 - $D(i)$ - weight of i^{th} training example (x_i, y_i)
 - Interpretations:
 - i^{th} training example counts as $D(i)$ examples
 - If I were to “resample” data, I would get more samples of “heavier” data points
- Now, in all calculations the i^{th} training example counts as $D(i)$ “examples”

The AdaBoost Algorithm

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = 1/m$.

For $t = 1, \dots, T$:

- Train base learner using distribution D_t .
- Get base classifier $h_t : X \rightarrow \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final classifier:

$$H(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right).$$

The AdaBoost Algorithm

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = 1/m$.

For $t = 1, \dots, T$:

- Train base learner using distribution D_t .
- Get base classifier $h_t : X \rightarrow \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

$$\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i]$$

$$\epsilon_t = \frac{1}{\sum_{i=1}^m D_t(i)} \sum_{i=1}^m D_t(i) \delta(h_t(x_i) \neq y_i)$$

Multi-class AdaBoost

- Assume $y \in \{1, \dots, k\}$
- Direct approach (AdaBoost.M1):

$$h_t : X \rightarrow Y$$

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

$$H_{\text{final}}(x) = \arg \max_{y \in Y} \sum_{t: h_t(x)=y} \alpha_t$$

- can prove same bound on error if $\forall t : \epsilon_t \leq 1/2$

Reducing to Binary Problems

- Say possible labels are $\{a,b,c,d,e\}$
- Each training example replaced by five $\{-1,+1\}$ labeled examples

$$x, c \rightarrow \begin{cases} (x, a), -1 \\ (x, b), -1 \\ (x, c), +1 \\ (x, d), -1 \\ (x, e), -1 \end{cases}$$

AdaBoost.MH

- Formally $h_t : X \times Y \rightarrow \{-1, +1\}$ (or \mathbb{R}) Used to be $X \rightarrow \{-1, +1\}$

$$D_{t+1}(i, y) = \frac{D_t(i, y)}{Z_t} \cdot \exp(-\alpha_t v_i(y) h_t(x_i, y))$$

where $v_i(y) = \begin{cases} +1 & \text{if } y_i = y \\ -1 & \text{if } y_i \neq y \end{cases}$

$$H_{\text{final}}(x) = \arg \max_{y \in Y} \sum_t \alpha_t h_t(x, y)$$

Can prove that $\text{training error}(H_{\text{final}}) \leq \frac{k}{2} \cdot \prod Z_t$

Summary

- Linear weak learners can drive training error to zero if points are “well-separated”
- Can show that if a gap exists between positive and negative points, generalization error converges to zero
- Results apply to any weak learner based on linear classifiers (neural networks, decision trees)
- Main drawback: does not allow overlapping distributions (suffers under uniform noise)

Advantages and Disadvantages

Advantages:

A general meta-algorithm - use any 'reasonable' weak learner

Single parameter to be tuned (# iterations) - in principle

Fast and easy to program

Theoretical performance guarantees

Difficulties:

Not clear how to incorporate prior knowledge effectively

Regularization often essential - best strategy unclear

The best choice of weak learner is not obvious

Decision boundaries generated using parallel-split based methods often very rugged

NEW: for CS 599 Final

- No need to know over-fitting specifically for boosting
- Also no need to know margin analysis for boosting
- Need to be able to make connection with margins in SVMs

Random Forests

- The forest consists of N trees constructed by randomly selecting dimensions of feature vector along which to split
- Class prediction:
 - Each tree votes for a class; the predicted class C for an observation is the plurality, $\max_C \sum_k [f_k(\mathbf{x}, \mathcal{T}) = C]$

Outline

- All topics covered in 7th set of Notes (midterm recap)
- Non-parametric Classification
- Linear Discriminant Functions
- Support Vector Machines (SVM)
- Ensemble Methods
- **Unsupervised Learning**
- Other notes

Supervised vs. Unsupervised Learning

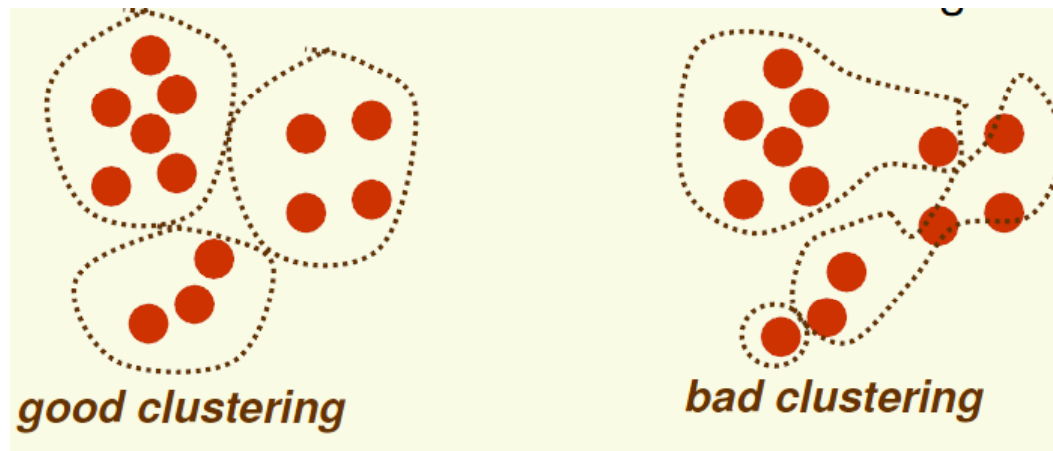
- Up to now we considered **supervised learning** scenarios, where we are given
 1. samples x_1, \dots, x_n
 2. class labels for all samples
 - This is also called learning with teacher, since the correct answer (the true class) is provided
- In **unsupervised learning** scenarios, we are only given
 1. samples x_1, \dots, x_n
 - This is also called learning without teacher, since the correct answer is not provided
 - Do not split data into training and test sets

Why Unsupervised Learning?

- Unsupervised learning is harder
 - How do we know if results are meaningful? No answer (labels) is available
 - Let the expert look at the results (external evaluation)
 - Define an objective function on clustering (internal evaluation)
- We nevertheless need it because
 1. Labeling large datasets is very costly (speech recognition, object detection in images)
 - Sometimes can label only a few examples by hand
 2. May have no idea what/how many classes there are (data mining)
 3. May want to use clustering to gain some insight into the structure of the data before designing a classifier
 - Clustering as data description

What we need for Clustering

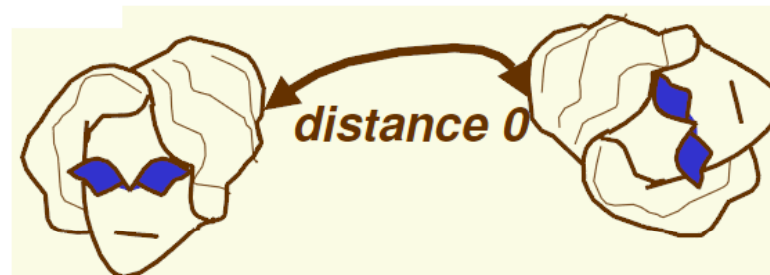
1. Proximity measure, *either*
 - similarity measure $s(x_i, x_k)$: large if x_i, x_k are similar
 - dissimilarity(or distance) measure $d(x_i, x_k)$: small if x_i, x_k are similar
2. Criterion function to evaluate a clustering



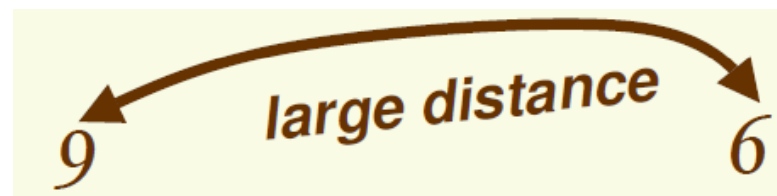
3. Algorithm to compute clustering
 - For example, by optimizing the criterion function

Proximity Measures

- A good proximity measure is VERY application dependent
 - Clusters should be invariant under the transformations “natural” to the problem
 - For example for object recognition, we should have invariance to rotation



- For character recognition, no invariance to rotation



Distance Measures

- Euclidean distance

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{k=1}^d (\mathbf{x}_i^{(k)} - \mathbf{x}_j^{(k)})^2}$$

- translation invariant

- Manhattan (city block) distance

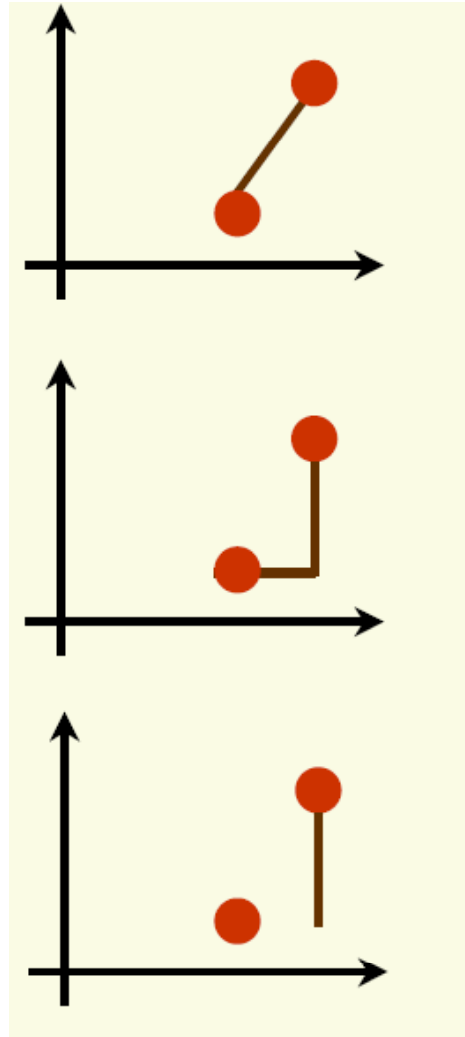
$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^d |\mathbf{x}_i^{(k)} - \mathbf{x}_j^{(k)}|$$

- approximation to Euclidean distance, cheaper to compute

- Chebyshev distance

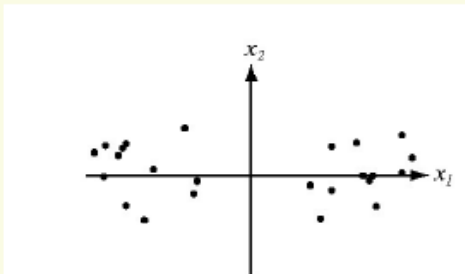
$$d(\mathbf{x}_i, \mathbf{x}_j) = \max_{1 \leq k \leq d} |\mathbf{x}_i^{(k)} - \mathbf{x}_j^{(k)}|$$

- approximation to Euclidean distance, cheapest to compute

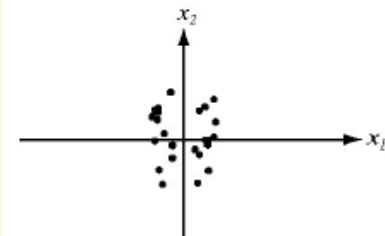


Feature Scaling

- Old problem: how to choose appropriate relative scale for features?
 - [length (in meters or cms?), weight (in grams or kgs?)]
- In supervised learning, can normalize to zero mean unit variance with no problems
- In clustering this is more problematic
- ***If variance in data is due to cluster presence, then normalizing features is not a good thing***



before normalization



after normalization

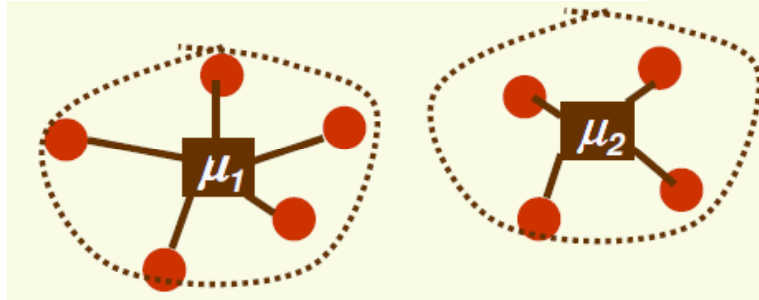
SSE Criterion Function

- Let n_i be the number of samples in D_i , and define the mean of samples in D_i

$$\mu_i = \frac{1}{n_i} \sum_{x \in D_i} x$$

- Then the sum-of-squared errors criterion function (to minimize) is:

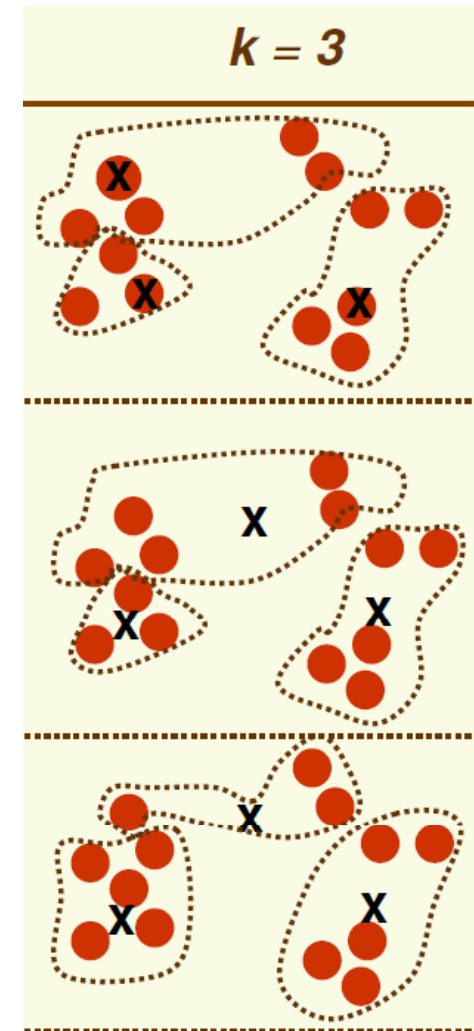
$$J_{SSE} = \sum_{i=1}^c \sum_{x \in D_i} \|x - \mu_i\|^2$$



- Note that the number of clusters, c , is fixed

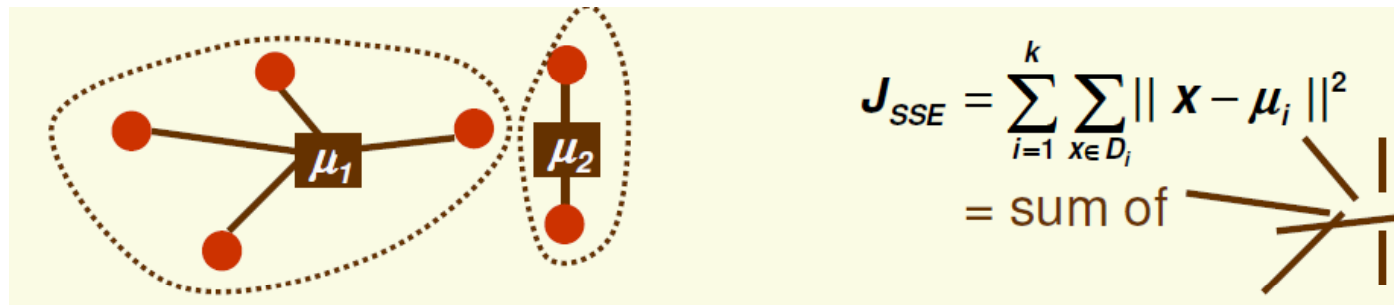
K-means Clustering

1. Initialize
 - Pick k cluster centers arbitrarily
 - Assign each example to closest center
2. Compute sample means for each cluster
3. Reassign all samples to the closest mean
4. If clusters changed at step 3, go to step 2

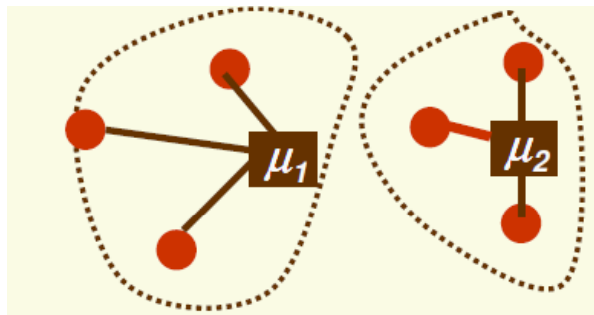


K-means Clustering

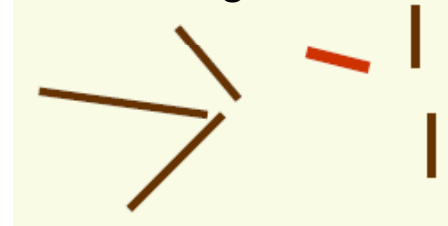
- Consider steps 2 and 3 of the algorithm
 1. compute sample means for each cluster



2. reassign all samples to the closest mean

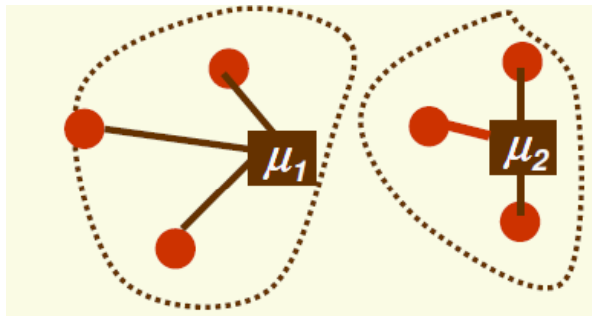


If we represent clusters by their old means, the error has gotten smaller

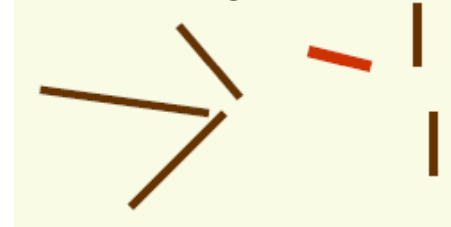


K-means Clustering

2. reassign all samples to the closest mean



If we represent clusters by their old means, the error has gotten smaller

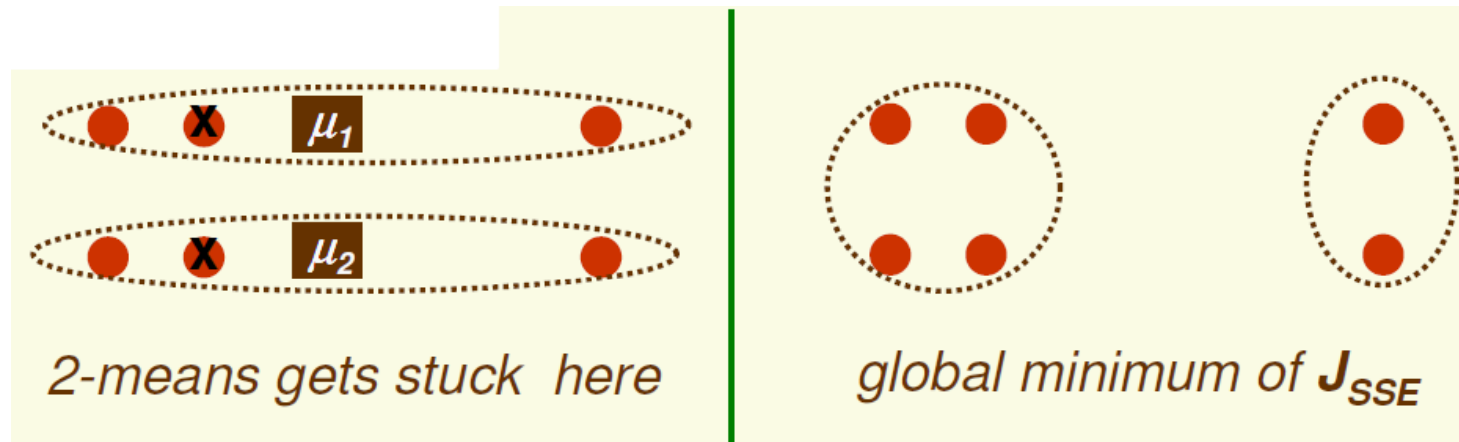


- However we represent clusters by their new means, and the mean always results in the smallest sum of squared distances

$$\frac{\partial}{\partial \mathbf{z}} \sum_{x \in D_i} \frac{1}{2} \|\mathbf{x} - \mathbf{z}\|^2 = \frac{\partial}{\partial \mathbf{z}} \sum_{x \in D_i} \frac{1}{2} (\|\mathbf{x}\|^2 - 2\mathbf{x}^t \mathbf{z} + \|\mathbf{z}\|^2) = \sum_{x \in D_i} (-\mathbf{x} + \mathbf{z}) = 0$$
$$\Rightarrow \mathbf{z} = \frac{1}{n_i} \sum_{x \in D_i} \mathbf{x}$$

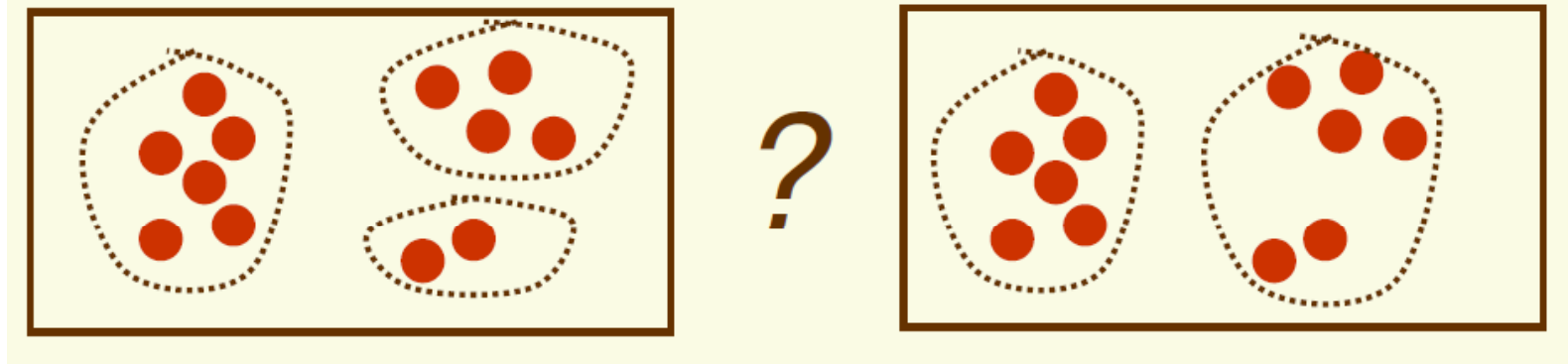
K-means Clustering

- Proved that by repeating steps 2 and 3, the objective function is reduced
 - Found a “smart “ move which decreases the objective function
- Thus k-means converges after a finite number of iterations of steps 2 and 3
- However k-means is not guaranteed to find a global minimum



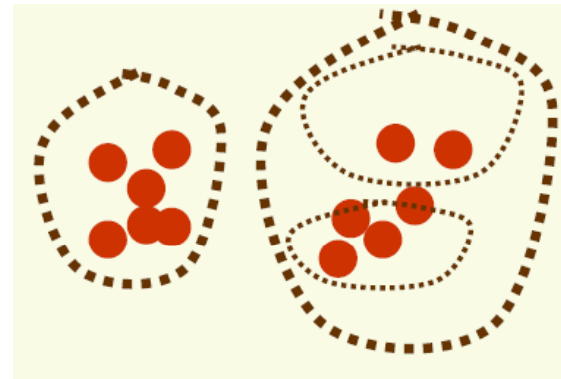
Hierarchical Clustering

- Up to now considered flat clustering



- For some data, hierarchical clustering is more appropriate than “flat” clustering

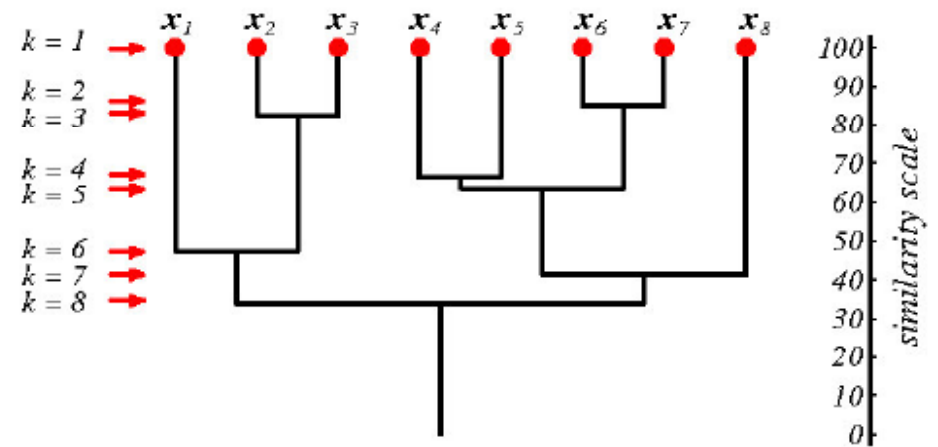
- Hierarchical clustering



Hierarchical Clustering: Dendrogram

- The preferred way to represent a hierarchical clustering is a dendrogram

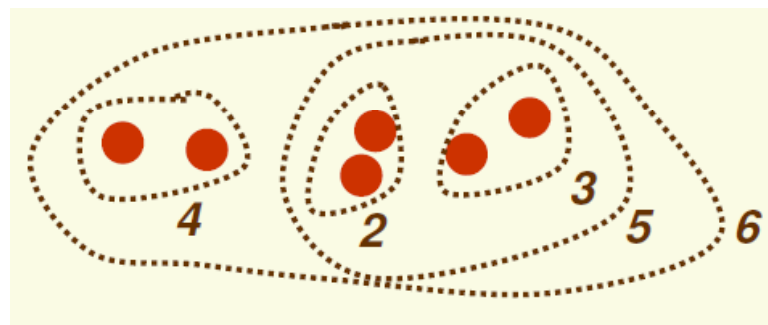
- Binary tree
- Level k corresponds to partitioning with $n-k+1$ clusters
- If k clusters required, use clustering from level $n-k+1$



- If samples are in the same cluster at level k , they stay in the same cluster at higher levels
- The dendrogram typically shows the similarity of grouped clusters

Hierarchical Clustering

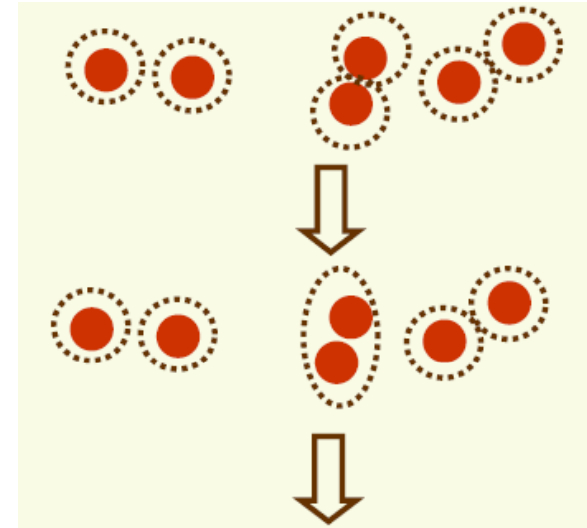
- Algorithms for hierarchical clustering can be
- divided into two types:
 1. Agglomerative (bottom up) procedures
 - Start with n singleton clusters
 - Form hierarchy by merging most similar clusters



2. Divisive (top down) procedures
 - Start with all samples in one cluster
 - Form hierarchy by splitting the “worst” clusters

Agglomerative Hierarchical Clustering

initialize with each example in singleton cluster
while there is more than 1 cluster
 1. find 2 nearest clusters
 2. merge them



- Four common ways to measure cluster distance

1. minimum distance $d_{\min}(D_i, D_j) = \min_{x \in D_i, y \in D_j} \|x - y\|$

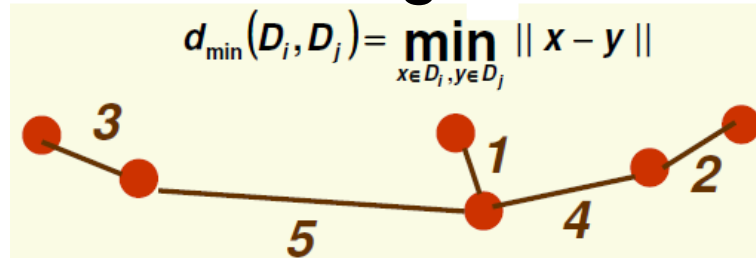
2. maximum distance $d_{\max}(D_i, D_j) = \max_{x \in D_i, y \in D_j} \|x - y\|$

3. average distance $d_{\text{avg}}(D_i, D_j) = \frac{1}{n_i n_j} \sum_{x \in D_i} \sum_{y \in D_j} \|x - y\|$

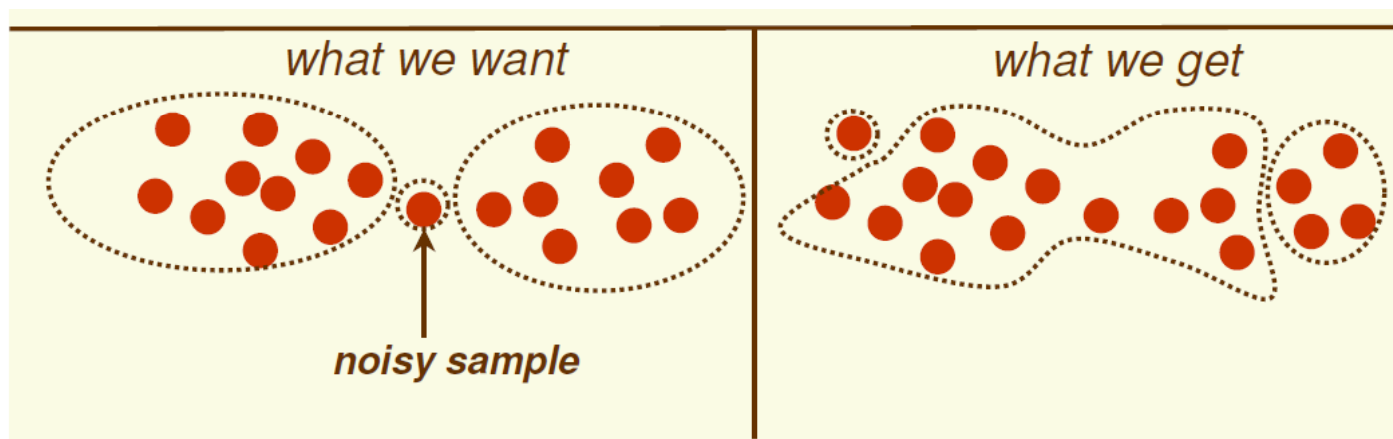
4. mean distance $d_{\text{mean}}(D_i, D_j) = \|\mu_i - \mu_j\|$

Single Linkage or Nearest Neighbor

- Agglomerative clustering with minimum distance



- Generates minimum spanning tree
- Encourages growth of elongated clusters
- Disadvantage: very sensitive to noise

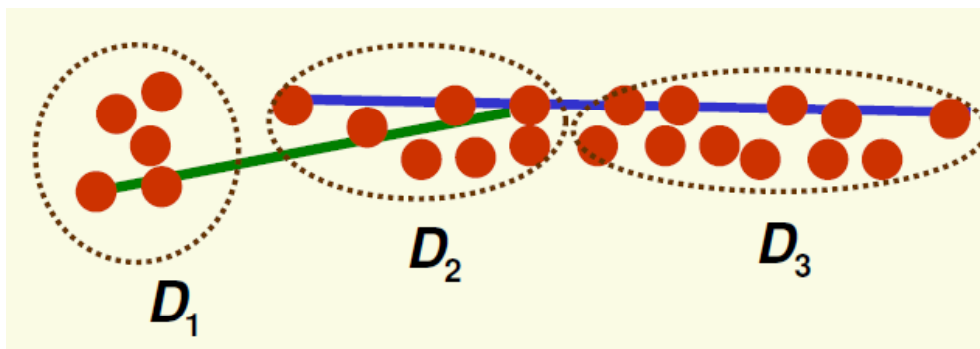


Complete Linkage or Farthest Neighbor

- Agglomerative clustering with maximum distance

$$d_{\max}(D_i, D_j) = \max_{x \in D_i, y \in D_j} \|x - y\|$$

- Encourages compact clusters
- Does not work well if elongated clusters are present



- $d_{\max}(D_1, D_2) < d_{\max}(D_2, D_3)$
- thus D_1 and D_2 are merged instead of D_2 and D_3

Average and Mean Agglomerative Clustering

- Agglomerative clustering is more robust under the average or the mean cluster distance

$$d_{avg}(D_i, D_j) = \frac{1}{n_i n_j} \sum_{x \in D_i} \sum_{y \in D_j} \|x - y\|$$

$$d_{mean}(D_i, D_j) = \|\mu_i - \mu_j\|$$

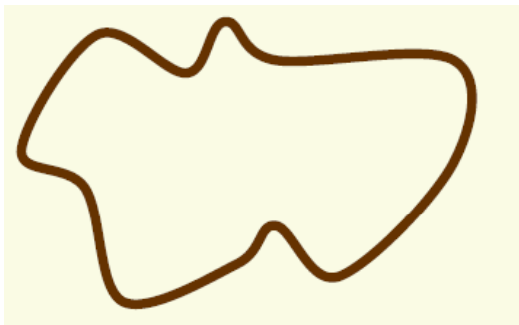
- Mean distance is cheaper to compute than the average distance
- Unfortunately, there is not much to say about agglomerative clustering theoretically, but it does work reasonably well in practice

Agglomerative vs. Divisive

- Agglomerative is faster to compute, in general
- Divisive may be less “blind” to the global structure of the data

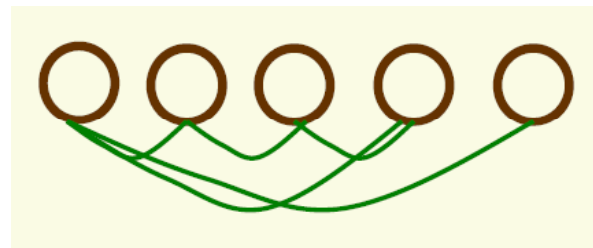
Divisive

- when taking the first step (split), it has access to all the data; can find the best possible split in 2 parts



Agglomerative

- when taking the first step (merge), it does not consider the global structure of the data, only looks at pairwise structure



NEW: End of Material

- The material for the final ends at slide 52 of Week 12

Outline

- All topics covered in 7th set of Notes (midterm recap)
- Non-parametric Classification
- Linear Discriminant Functions
- Support Vector Machines (SVM)
- Ensemble Methods
- Unsupervised Learning
- Other notes

Overfitting

- Prediction error: probability of test pattern not in class with max posterior (true)
- Training error: probability of test pattern not in class with max posterior (estimated)
- Classifier optimized w.r.t. training error
 - Training error: optimistically biased estimate of prediction error

Overfitting

Overfitting: a learning algorithm overfits the training data if it outputs a solution w when exists another solution w' such that:

$$\text{error}_{\text{train}}(w) < \text{error}_{\text{train}}(w')$$

AND

$$\text{error}_{\text{true}}(w') < \text{error}_{\text{true}}(w)$$

Cross-Validation

- In practice
- Available data => training and validation
- Train on the training data
- Test on the validation data
- k-fold cross validation:
 - Data randomly separated into k groups
 - Each time k-1 groups used for training and one as testing