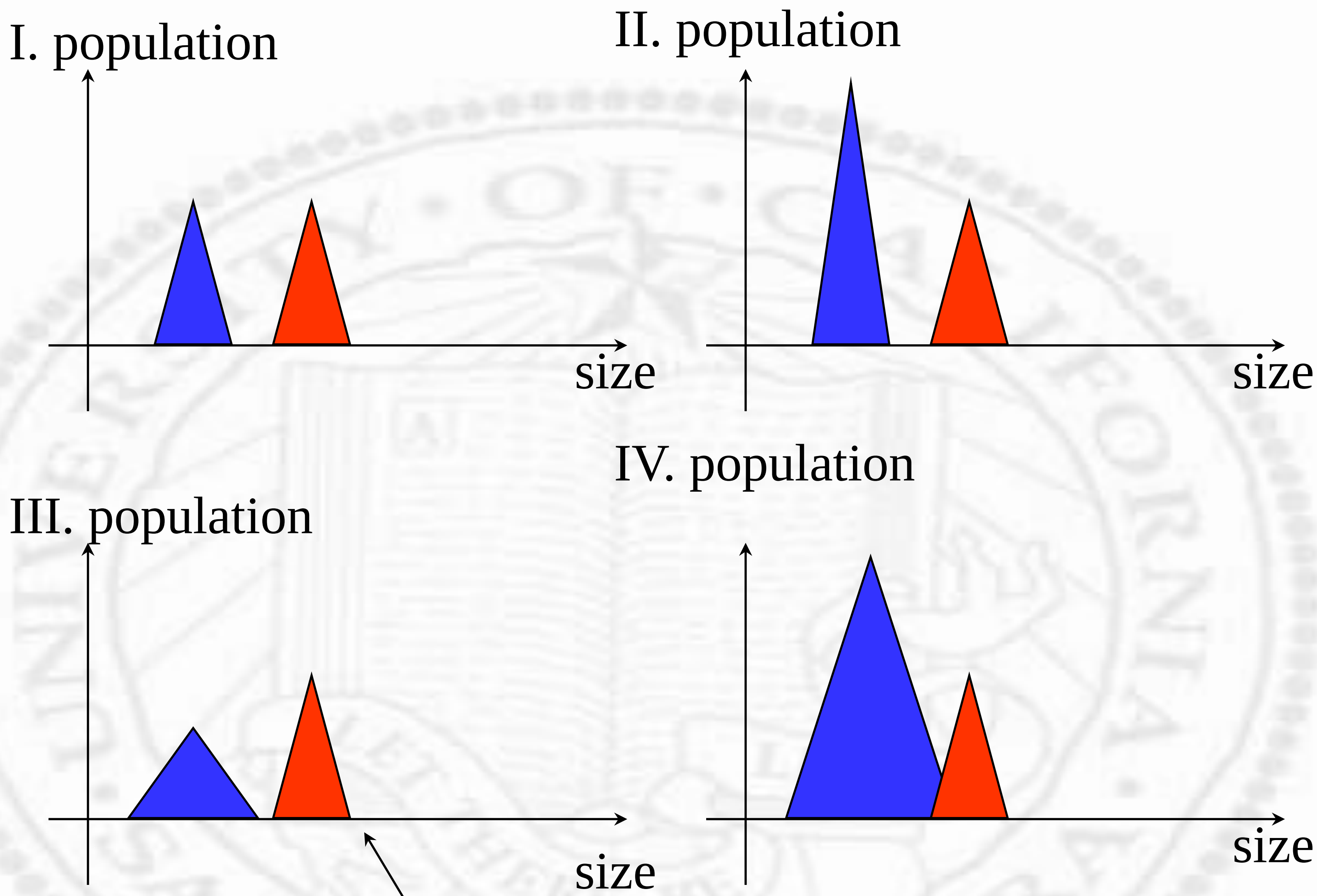
# Linear *Discriminant Functions*

# Linear *Discriminant Functions*

- So far, concentrate on *density* functions
  - with a known parametric form
  - shape of the function directly
- Here, learn the *discriminant* functions
  - surface separating different clusters
  - what type of surfaces?
  - *linear* (easiest!) functions (hyperplanes)




$$\frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{1}{2}\frac{|x-\bar{x}|^2}{\sigma^2}}$$

# *Case I: same prior, same deviation*

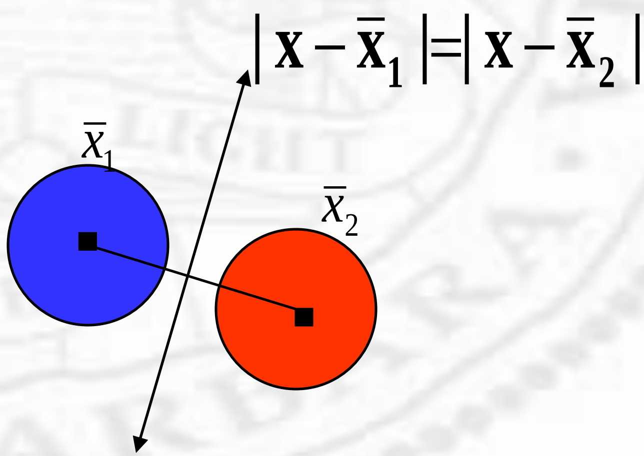
- Decision boundary is planar
- In the middle of the two cluster

$$P(\varpi_1 \mid \mathbf{x}) = P(\varpi_2 \mid \mathbf{x})$$

$$P(\varpi_1)\, p(\mathbf{x} \mid \varpi_1) = P(\varpi_2)\, p(\mathbf{x} \mid \varpi_2)$$

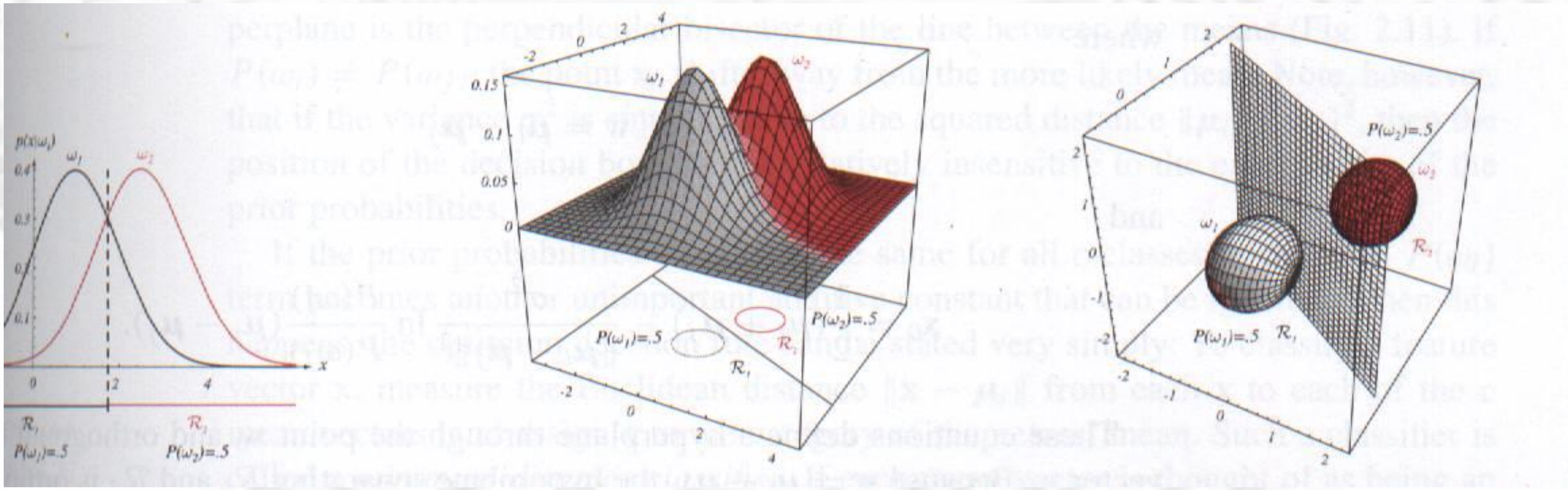
$$P \frac{1}{(2\pi)^{1/2}\sigma} e^{-\frac{1}{2}\frac{|\mathbf{x}-\bar{\mathbf{x}}_1|^2}{\sigma^2}} = P \frac{1}{(2\pi)^{1/2}\sigma} e^{-\frac{1}{2}\frac{|\mathbf{x}-\bar{\mathbf{x}}_2|^2}{\sigma^2}}$$

$$|\mathbf{x}-\bar{\mathbf{x}}_1| = |\mathbf{x}-\bar{\mathbf{x}}_2|$$



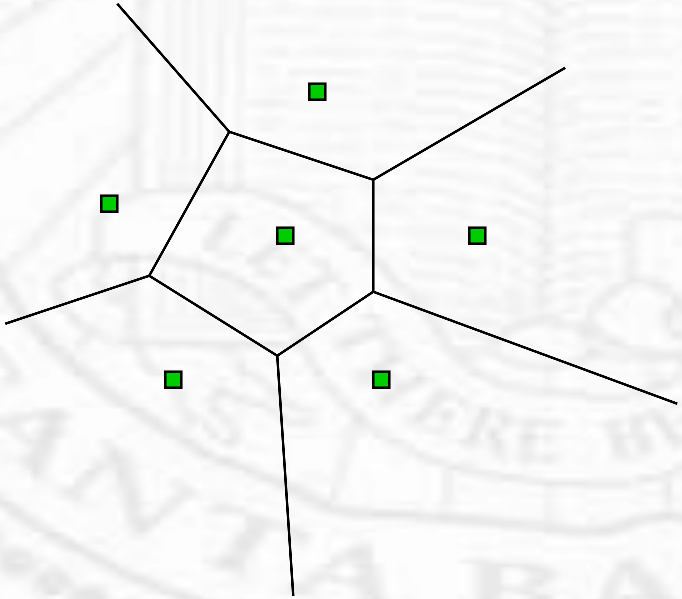
# *Case 1.A*

- The partition plane is perpendicular to the line connecting two means
  - Scalar case
  - Covariance matrices are the same and are diagonal with the same variance in all features $\mathbf{\Sigma} = \sigma^2 \mathbf{I}$



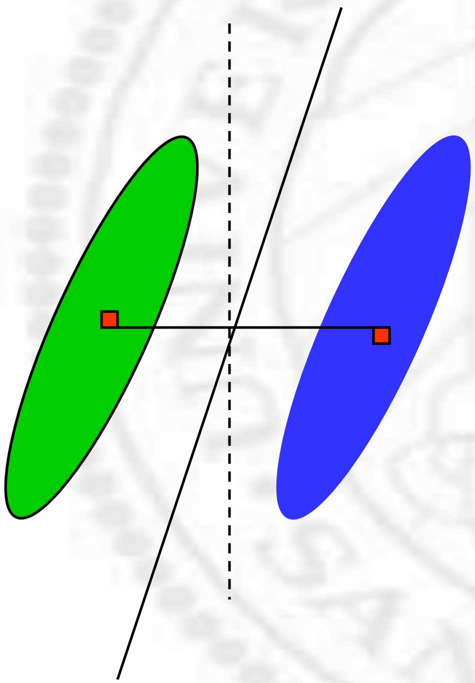
# Case I.A: same prior, same deviation

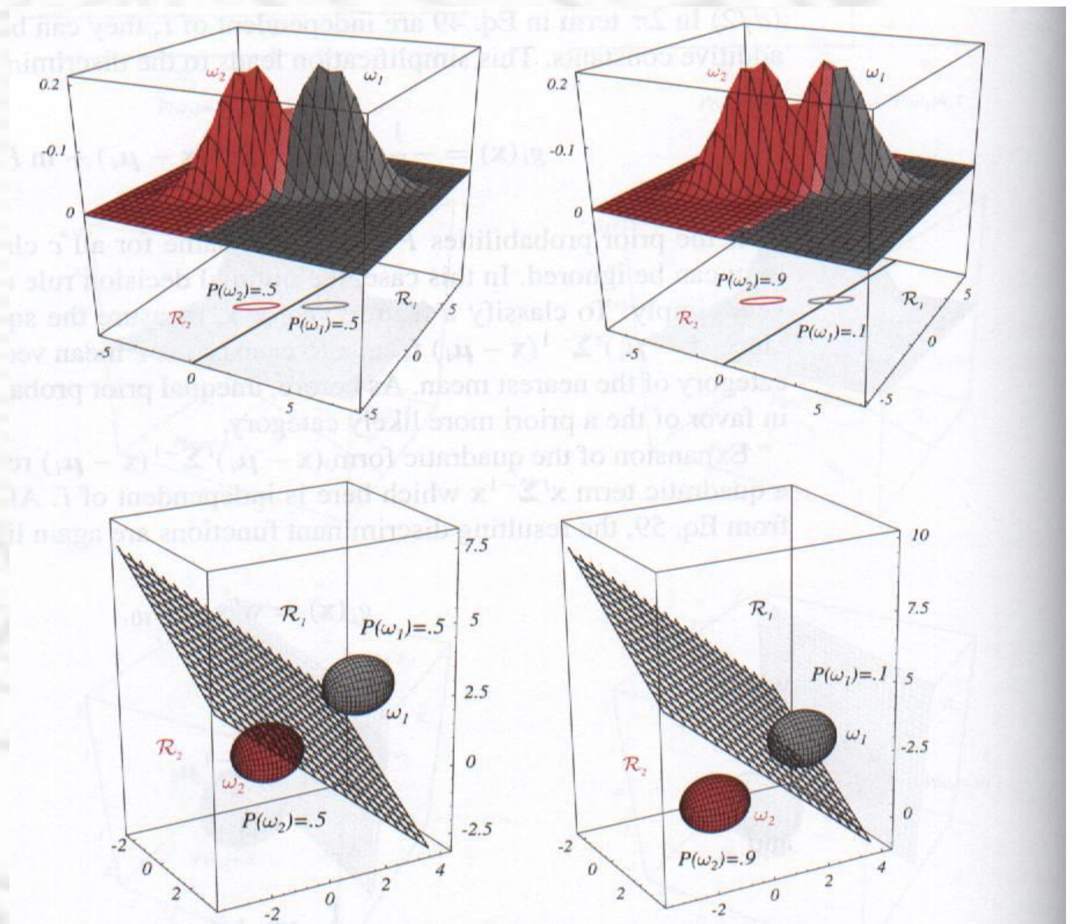
- Even with multiple classes, if they all have the same prior and the same deviation, then
  - the decision boundaries form a Vonoroi diagram, or Bayes rule is a minimum Euclidean distance classifier



# Case 1.B

- The partition plane is not perpendicular to the line connecting two means
  - Same (but general) covariance matrices





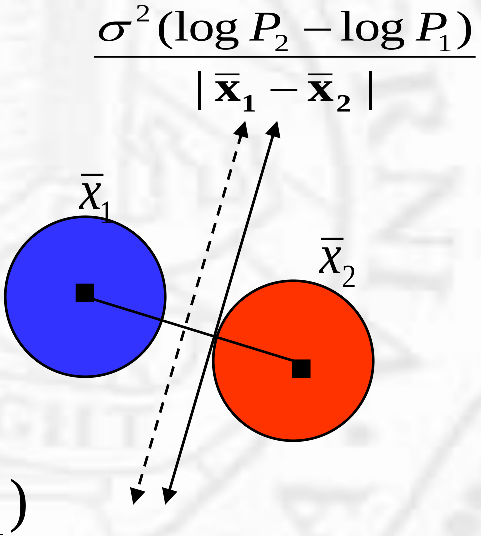
# *Case II: different prior, same deviation*

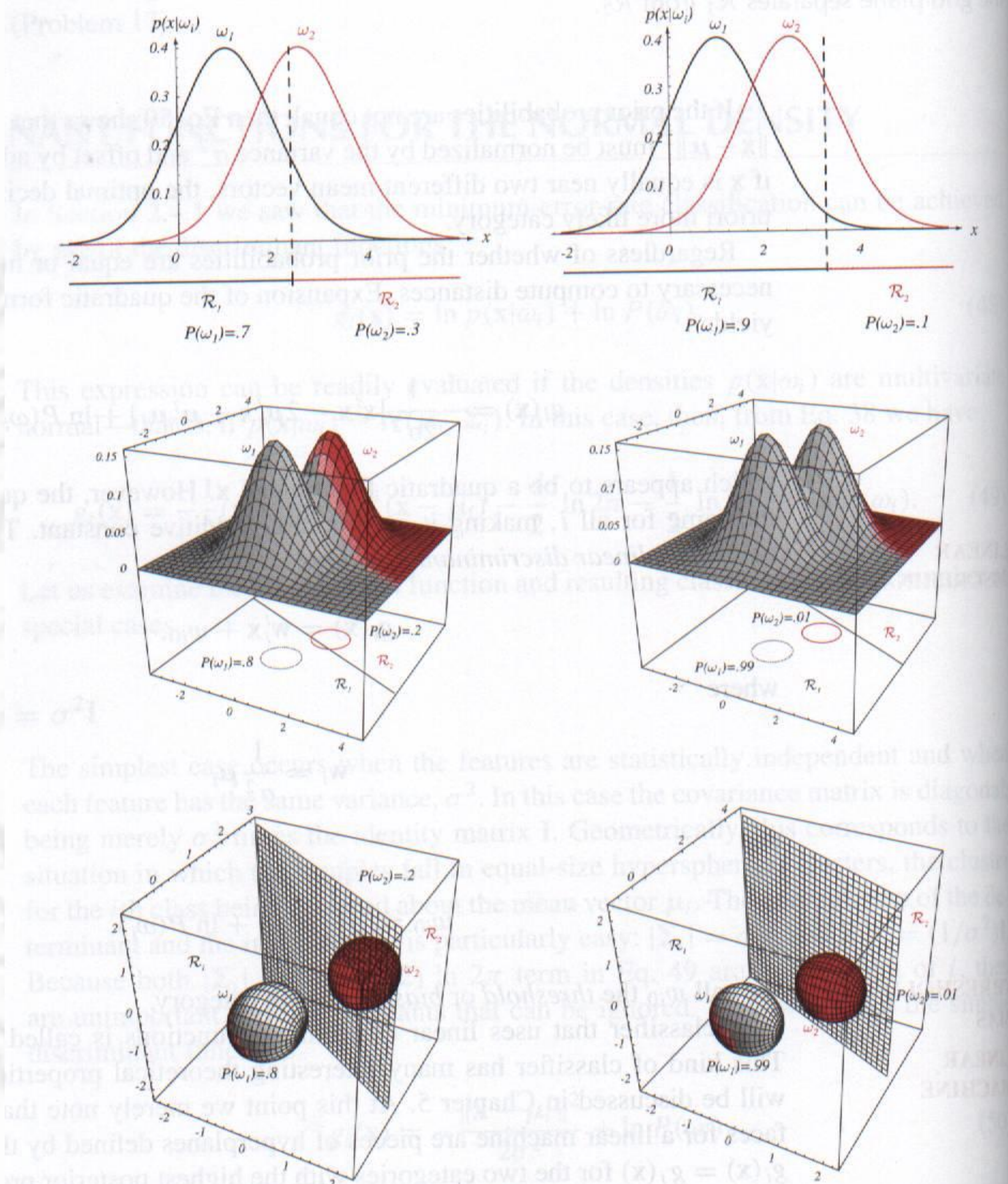
- Decision boundary is still planar
- At $$\frac{1}{2}(\bar{\mathbf{x}}_1 + \bar{\mathbf{x}}_2) + \frac{\sigma^2(\log P_2 - \log P_1)}{|\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2|}$$

$$P_1 \frac{1}{(2\pi)^{1/2}\sigma} e^{-\frac{1}{2}\frac{|\mathbf{x}-\bar{\mathbf{x}}_1|^2}{\sigma^2}} = P_2 \frac{1}{(2\pi)^{1/2}\sigma} e^{-\frac{1}{2}\frac{|\mathbf{x}-\bar{\mathbf{x}}_2|^2}{\sigma^2}}$$

$$\log P_1 - \frac{1}{2}\frac{|\mathbf{x}-\bar{\mathbf{x}}_1|^2}{\sigma^2} = \log P_2 - \frac{1}{2}\frac{|\mathbf{x}-\bar{\mathbf{x}}_2|^2}{\sigma^2}$$

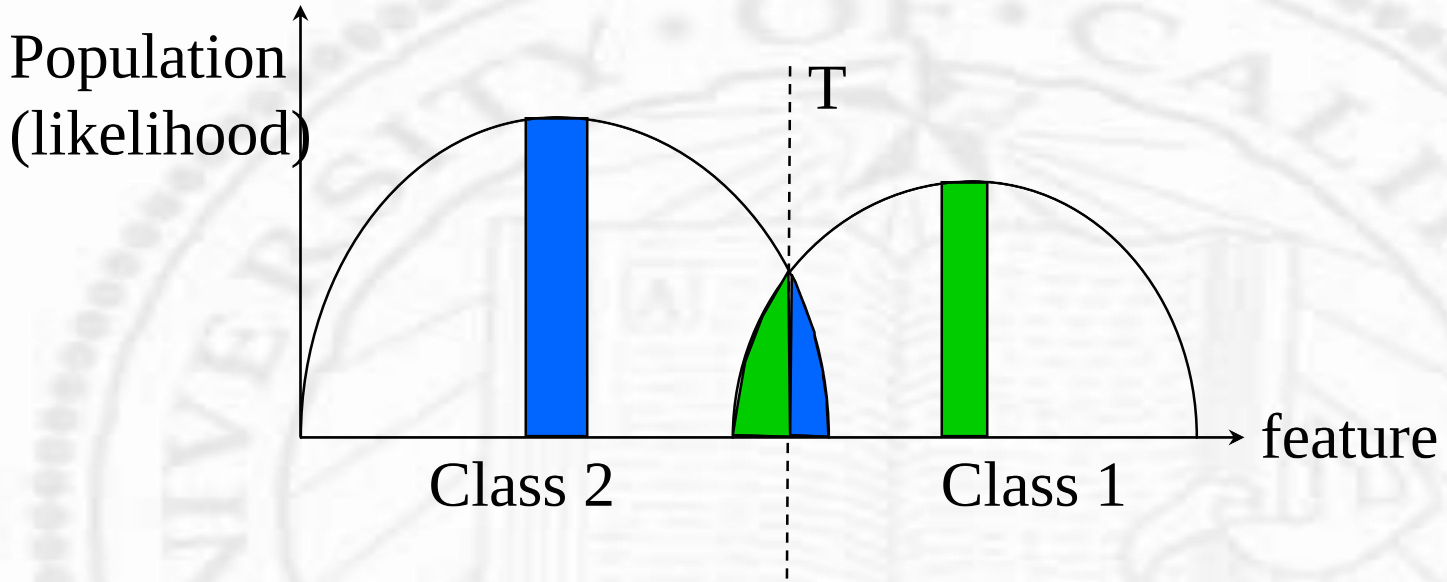
$$(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)\mathbf{x} = \frac{1}{2}(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)(\bar{\mathbf{x}}_1 + \bar{\mathbf{x}}_2) + \sigma^2(\log P_2 - \log P_1)$$





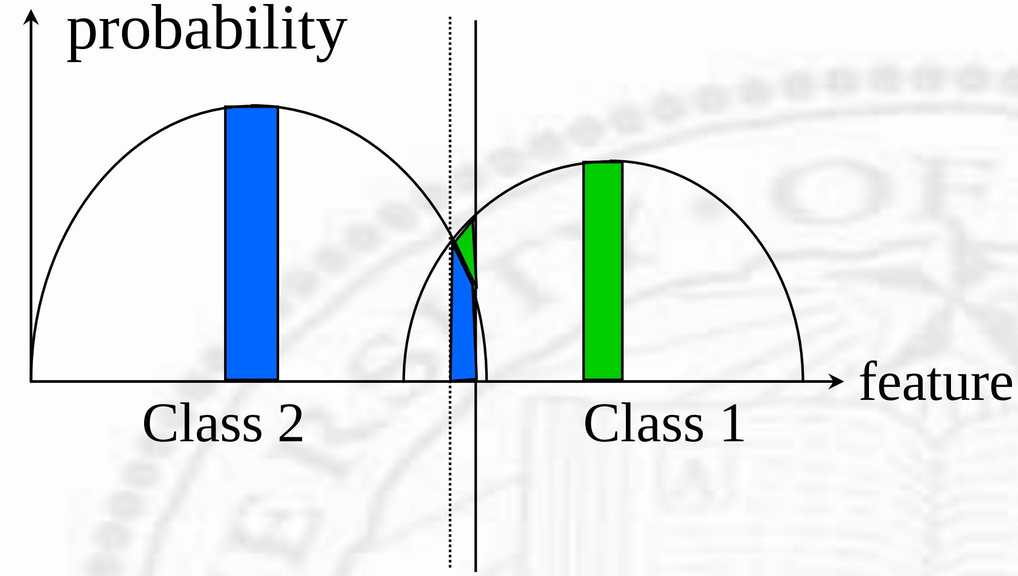
p(x|ωi)
ω1
ω2
x
R1
R2
P(ω1)=.7
P(ω2)=.3
P(ω1)=.9
P(ω2)=.1
P(ω1)=.8
P(ω2)=.2
P(ω1)=.99
P(ω2)=.01

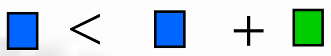
# *Graphical Interpretation in 1D*



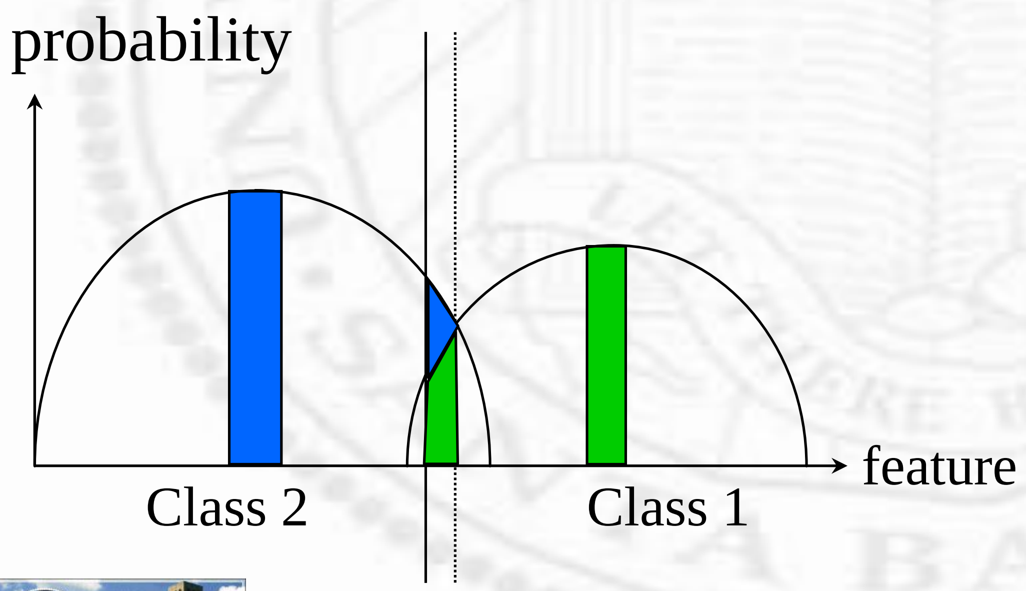

Class 1 misclassified as Class 2

Class 2 misclassified as Class 1



More class 1 misclassification



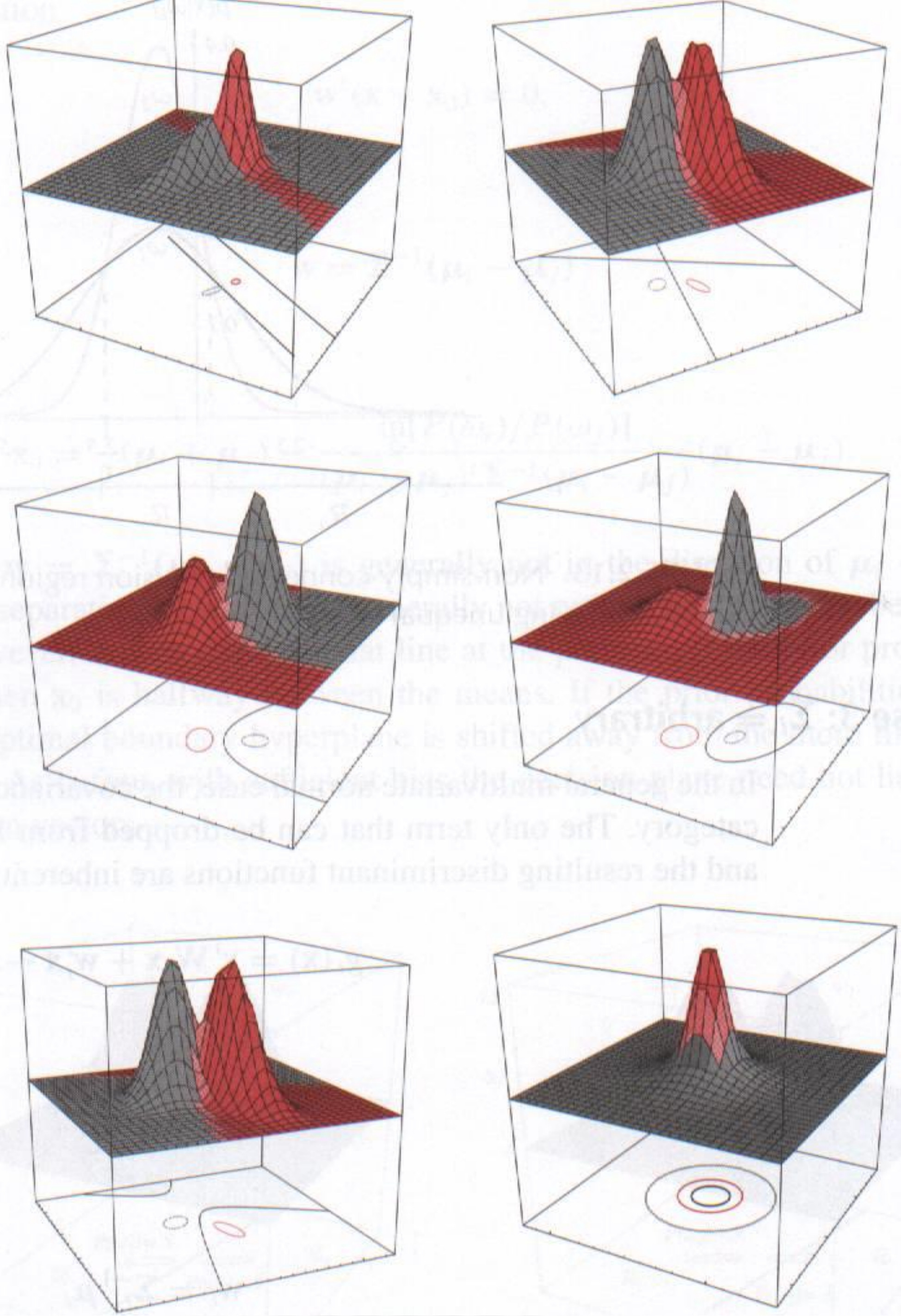

More class 2 misclassification

# *Case III & IV: same or different prior, different deviation*

- Decision boundary is no longer planar

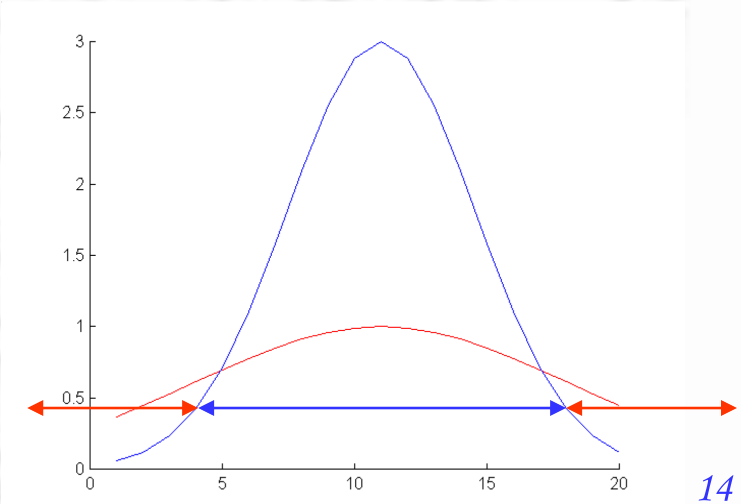
$$P\frac{1}{(2\pi)^{1/2}\sigma_1}e^{-\frac{1}{2}\frac{|\mathbf{x}-\bar{\mathbf{x}}_1|^2}{\sigma_1{}^2}} = P\frac{1}{(2\pi)^{1/2}\sigma_2}e^{-\frac{1}{2}\frac{|\mathbf{x}-\bar{\mathbf{x}}_2|^2}{\sigma_2{}^2}}$$

$$-n\log\sigma_1 - \frac{1}{2}\frac{|\mathbf{x}-\bar{\mathbf{x}}_1|^2}{\sigma_1{}^2} = -n\log\sigma_2 - \frac{1}{2}\frac{|\mathbf{x}-\bar{\mathbf{x}}_2|^2}{\sigma_2{}^2}$$



# Lessons

- The decision boundaries in general are NOT linear or planar
- Even with a single feature and a Gaussian distribution the boundary can be complicated
- That said,
  - planar boundaries can be used to approximate curved, disjoint boundaries  (a **lot** more on this later), “massage” the classifier
  - Features can also be “massaged”
- They are mathematically more tractable



# Two-category case

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + \mathbf{w}_0$$

$$\varpi_1 \quad g(\mathbf{x}) > 0$$

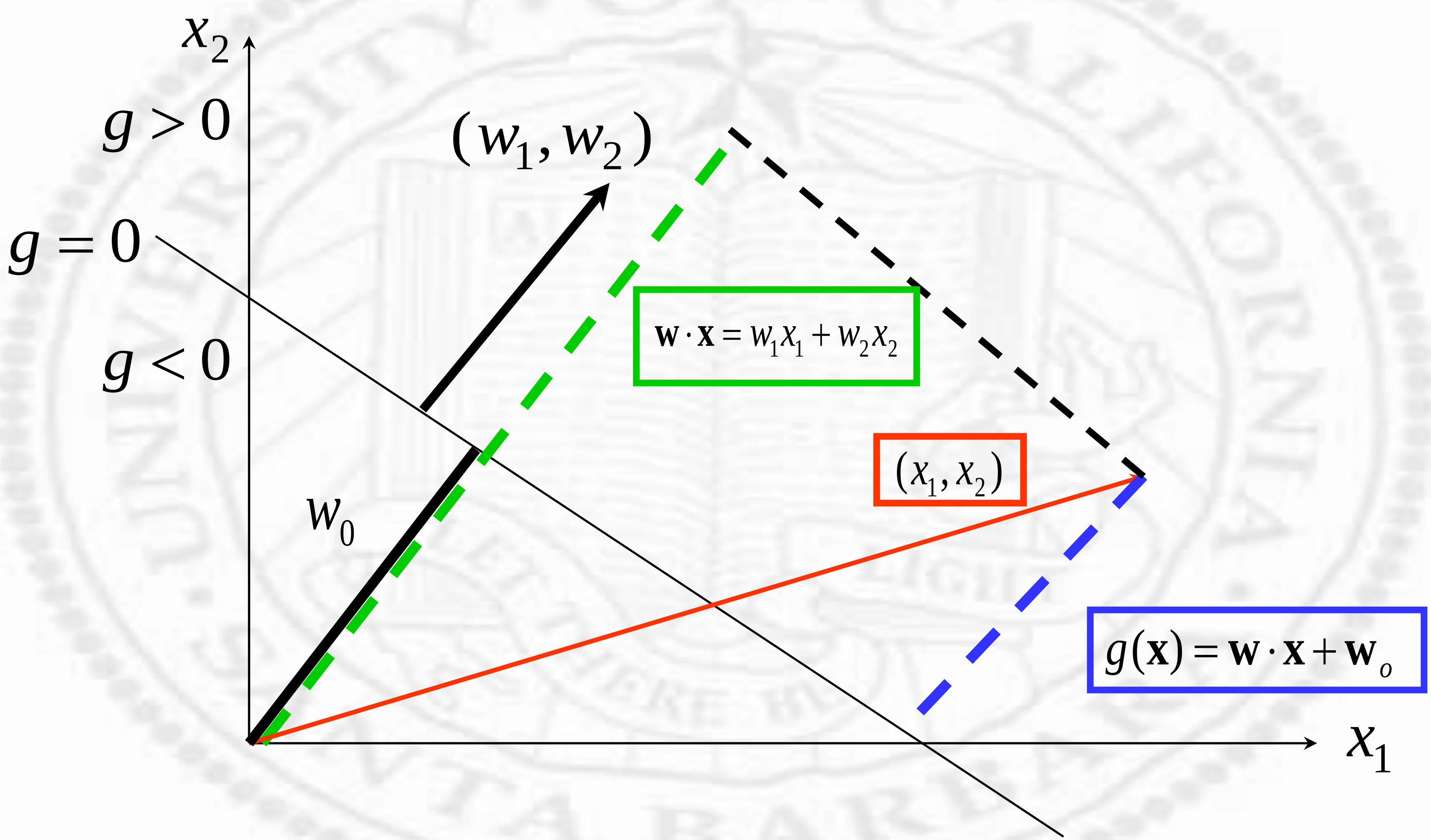
$$\varpi_2 \quad g(\mathbf{x}) < 0$$

$$\mathbf{w} \quad weightvector$$

$$\mathbf{w}_0 \quad threshold\,weight$$

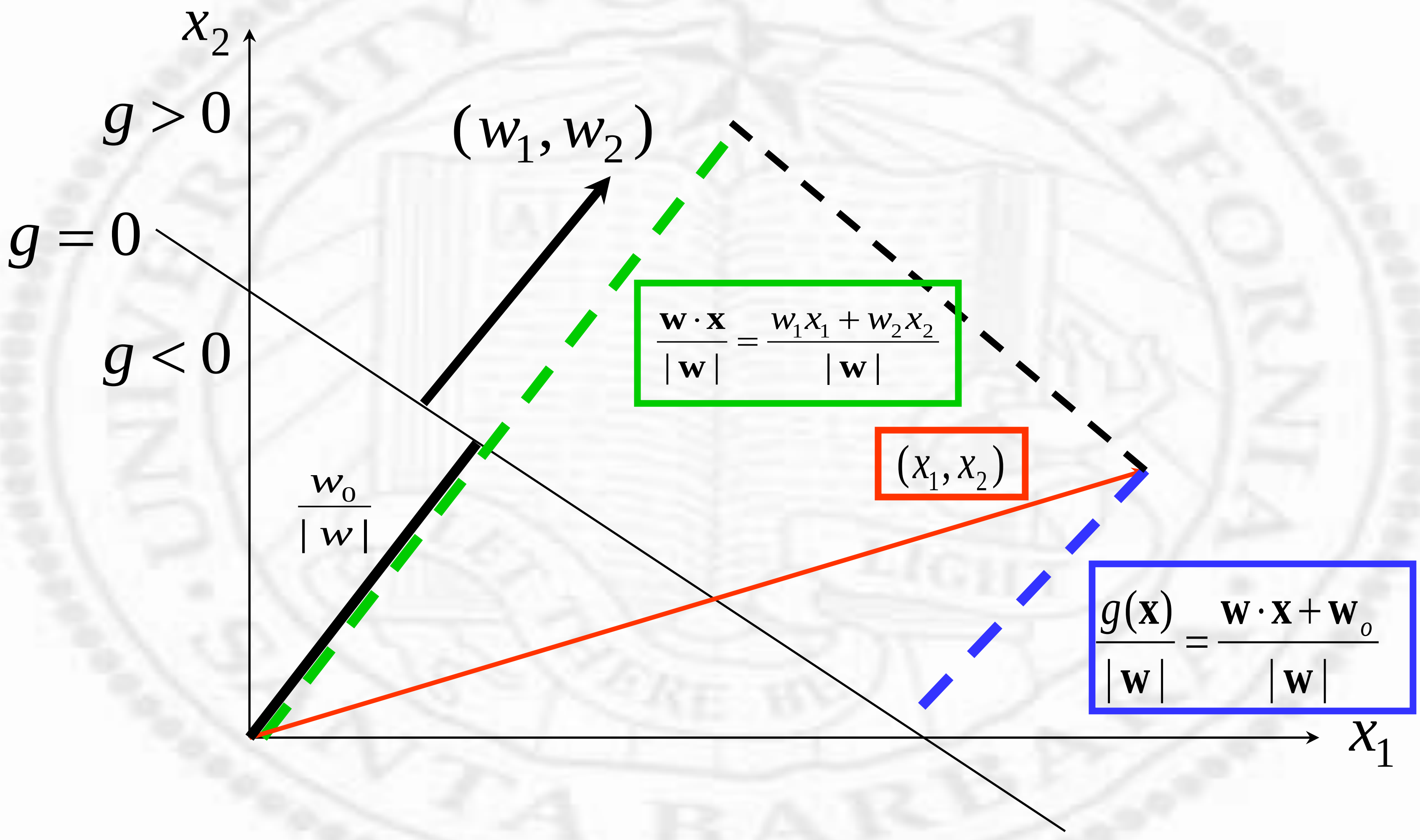
# Decision surface (Hyperplane)

$$g(x_1, x_2) = \mathbf{w} \cdot \mathbf{x} + \mathbf{w}_0 = w_1 x_1 + w_2 x_2 + w_0 \qquad |\mathbf{w}| = 1$$



# Decision surface (Hyperplane)

$$g(x_1, x_2) = \mathbf{w} \cdot \mathbf{x} + \mathbf{w}_0 = w_1 x_1 + w_2 x_2 + w_0$$



# Training Procedure

- Two-category case
  - Use n tagged samples $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ to determine the discriminant function

$$\mathbf{w}^{\mathbf{t}}\mathbf{x}_i + \mathbf{w}_o > 0 \quad \mathbf{x}_i \in \varpi_1$$

$$\mathbf{w}^{\mathbf{t}}\mathbf{x}_j + \mathbf{w}_o < 0 \quad \mathbf{x}_j \in \varpi_2$$

---

$$\mathbf{w}^{\mathbf{t}}\mathbf{x}_i + w_0 \times 1 > 0 \qquad \mathbf{x}_i \in \varpi_1$$

$$\mathbf{w}^{\mathbf{t}}(-\mathbf{x}_j) + w_0 \times (-1) > 0 \quad \mathbf{x}_j \in \varpi_2$$

---

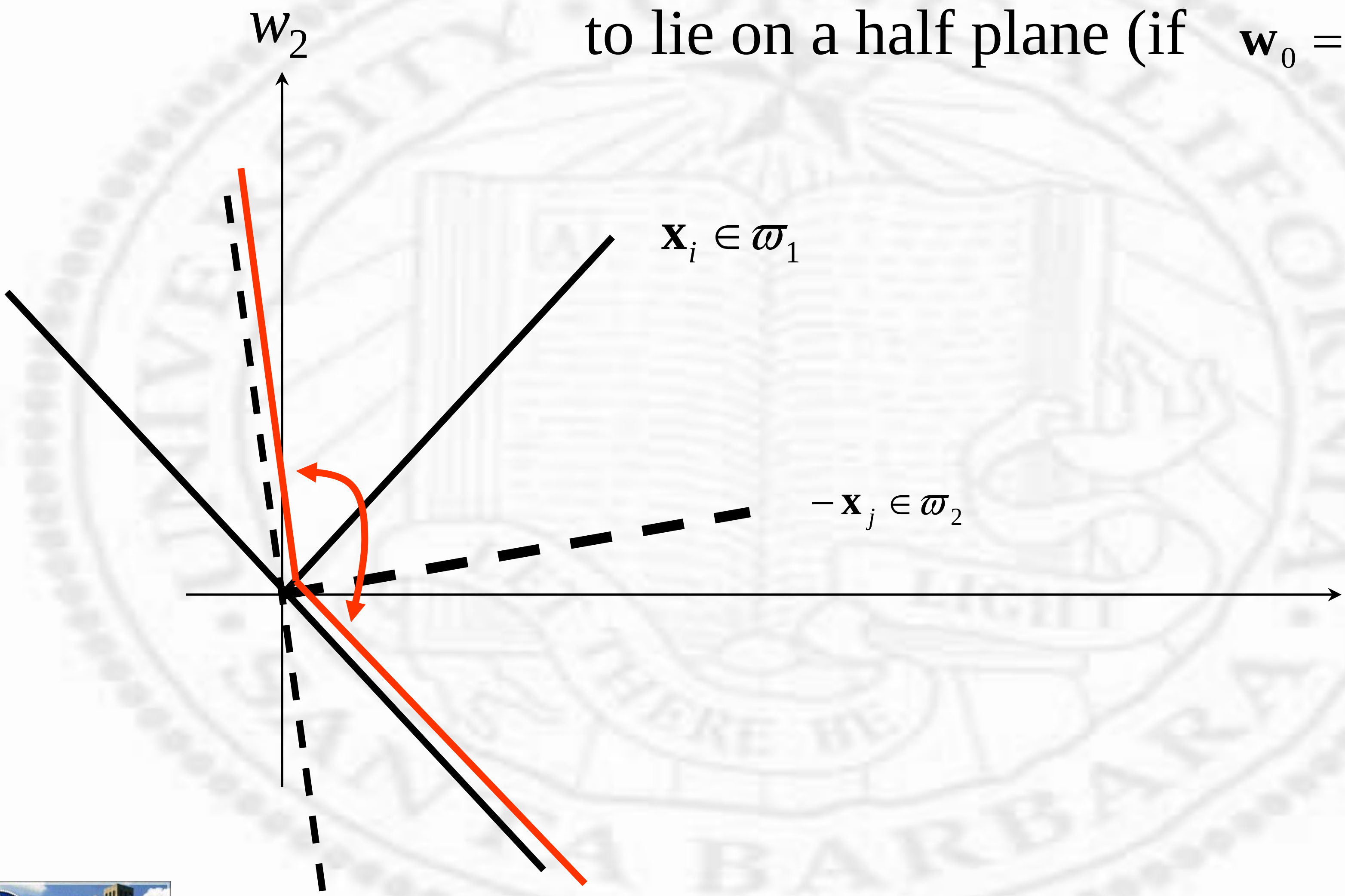
$$(\mathbf{w}^t, w_0)^t \boxed{(\mathbf{x}_i, 1)} > 0 \qquad \mathbf{x}_i \in \varpi_1$$

$$(\mathbf{w}^t, w_0)^t [\boxed{-(\mathbf{x}_j, 1)}] > 0 \quad \mathbf{x}_j \in \varpi_2$$

Augmented feature vectors

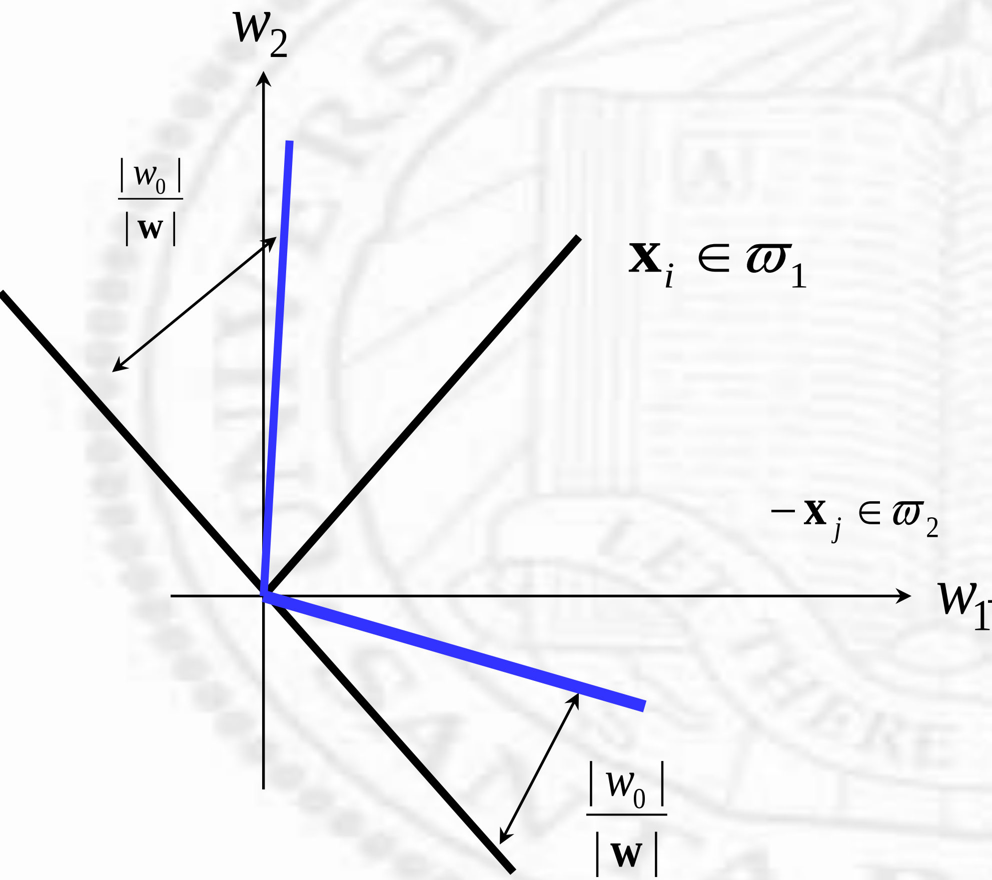
# Training Procedure (cont.)

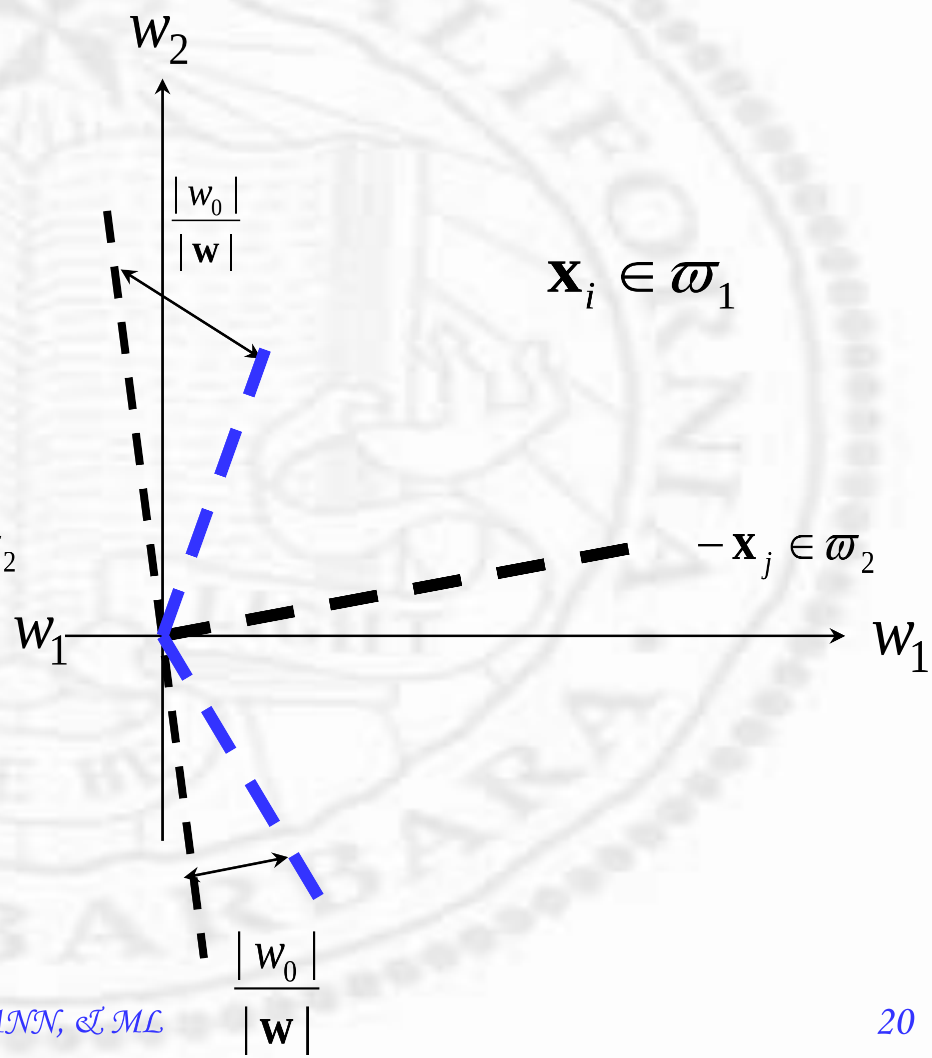
- Each training sample constrains ***w*** to lie on a half plane (if $\mathbf{w}_0 = 0$ )



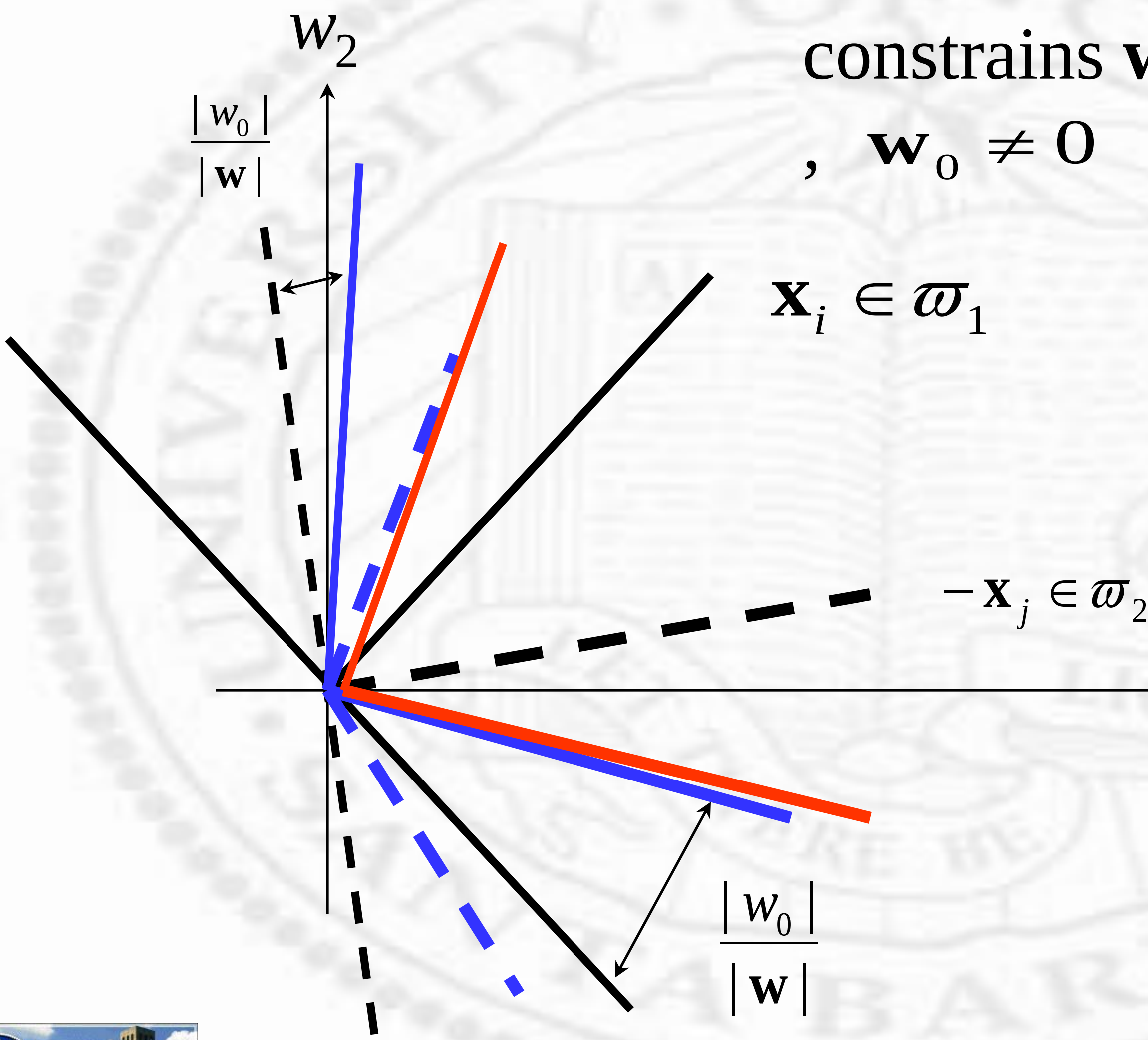
# Training Procedure (cont.)

$$\frac{\mathbf{w}^{\mathbf{t}}\mathbf{x}_i - \mathbf{w}_o}{|\mathbf{w}|} > 0 \Rightarrow \frac{\mathbf{w}^{\mathbf{t}}\mathbf{x}_i}{|\mathbf{w}|} > \frac{\mathbf{w}_o}{|\mathbf{w}|}$$



# Training Procedure (cont.)




- Each training sample constrains $\mathbf{w}$ to lie on a pie (if , $\mathbf{w}_0 \neq 0$ with margin)

# *Using Gradient Descent*

- A search mechanism
- Start at an arbitrarily chosen starting point
- Move in a direction (gradient) to minimize the cost function
- Basic calculus, to be expected of every engineer after 5 minute thought ☺

# *Using Gradient Descent*

- Cost function (in terms of augmented feature vector [x,1])
  - penalized for all samples misclassified

$$c(\mathbf{w}) = \sum_{x \in \Im} (-\mathbf{w}^t \mathbf{x}) \qquad \Im : \text{misclassif ied samples}$$

- Gradient direction

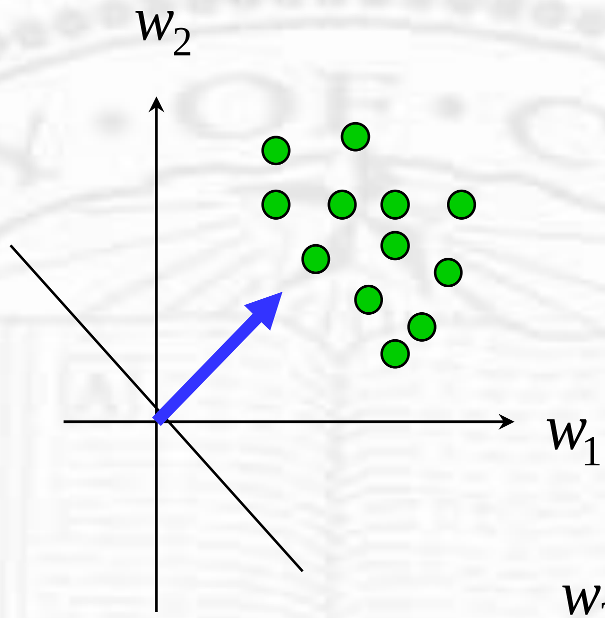
$$\nabla c(\mathbf{w}) = \begin{bmatrix} \frac{\partial c(\mathbf{w})}{\partial w_1} & \frac{\partial c(\mathbf{w})}{\partial w_2} & . & . & \frac{\partial c(\mathbf{w})}{\partial w_d} \end{bmatrix}$$

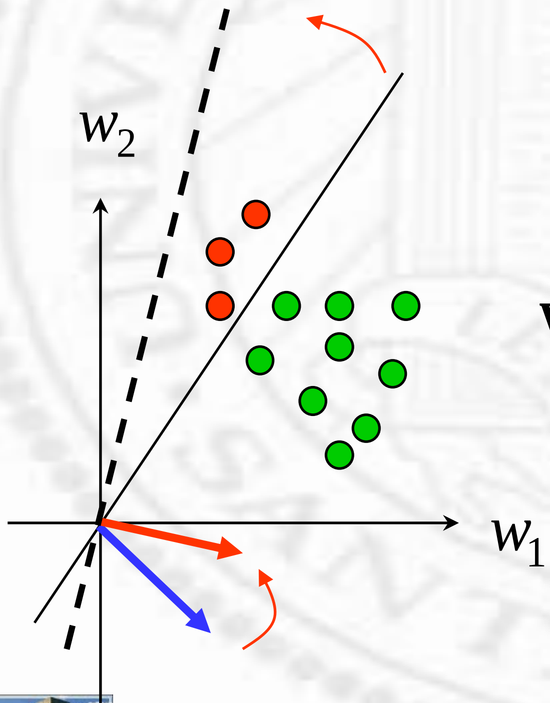
$$= \begin{bmatrix} -\sum_{i=1}^{n} x_{i1} & -\sum_{i=1}^{n} x_{i2} & . & . & -\sum_{i=1}^{n} x_{id} \end{bmatrix} = \sum_{i=1}^{n} (-\mathbf{x}_i)$$

- Update

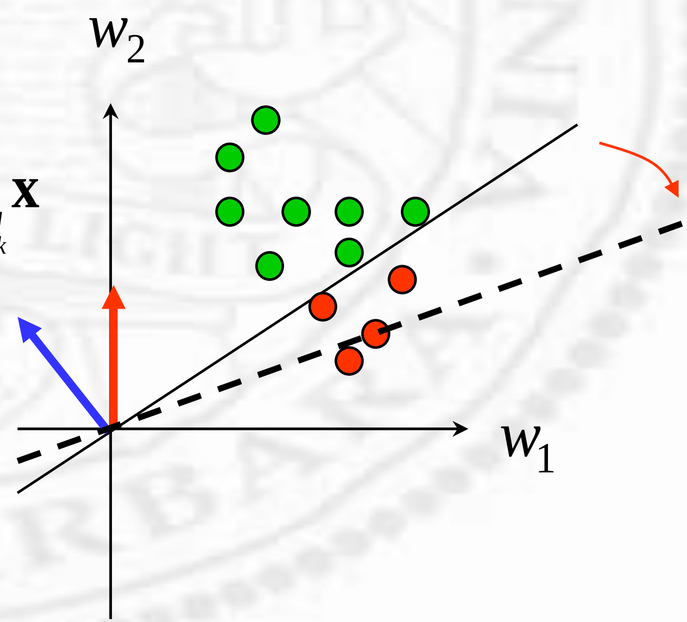
$$\mathbf{w}^{k+1} = \mathbf{w}^k - \rho^k \nabla c(\mathbf{w}^k) = \mathbf{w}^k + \rho^k \sum_{x \in \Im^k} \mathbf{x}$$

# Graphical Interpretation




$$\mathbf{w}^{k+1} = \mathbf{w}^{k} + \rho^{k} \sum_{x \in \mathfrak{I}^{k}} \mathbf{x}$$

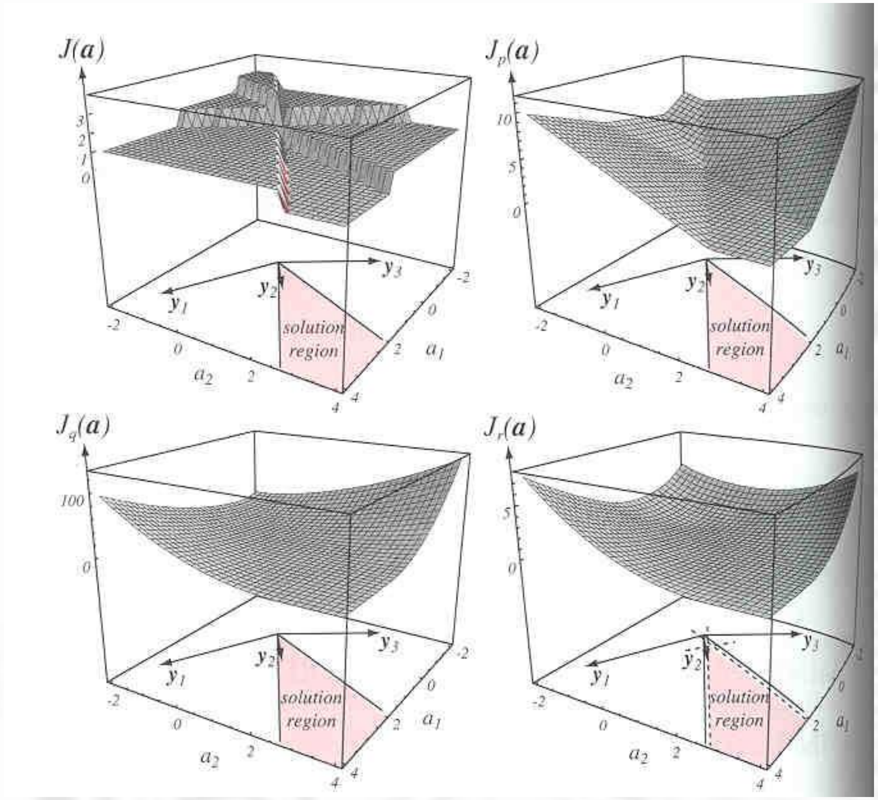


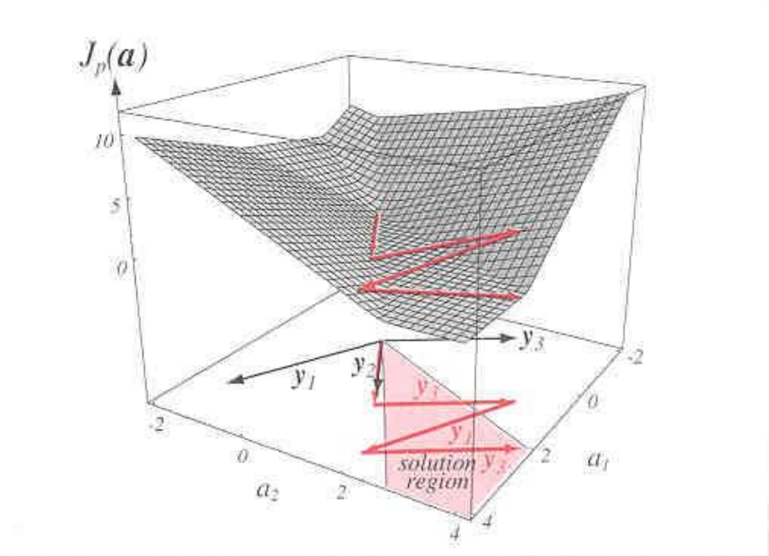
# Graphical Interpretation (cont)

- Weight is the *signed* sum of samples
  - The more difficult a sample is to be classified, the more its weight
- During classification, we have

$$y = \mathbf{w} \cdot \mathbf{x} = (\sum_i \alpha_i y_i \mathbf{x}_i) \cdot \mathbf{x} = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x}$$

  - Only inner product of "troublesome" training samples and test samples are needed ($\alpha$:weight, y: class)
- These two concepts are very important, they appear again and again later in
  - Perceptron
  - SVM
  - Kernel methods




Number of wrong samples

$$\sum_{x \in \Im} (-\mathbf{w}^{\mathbf{t}}\mathbf{x})$$

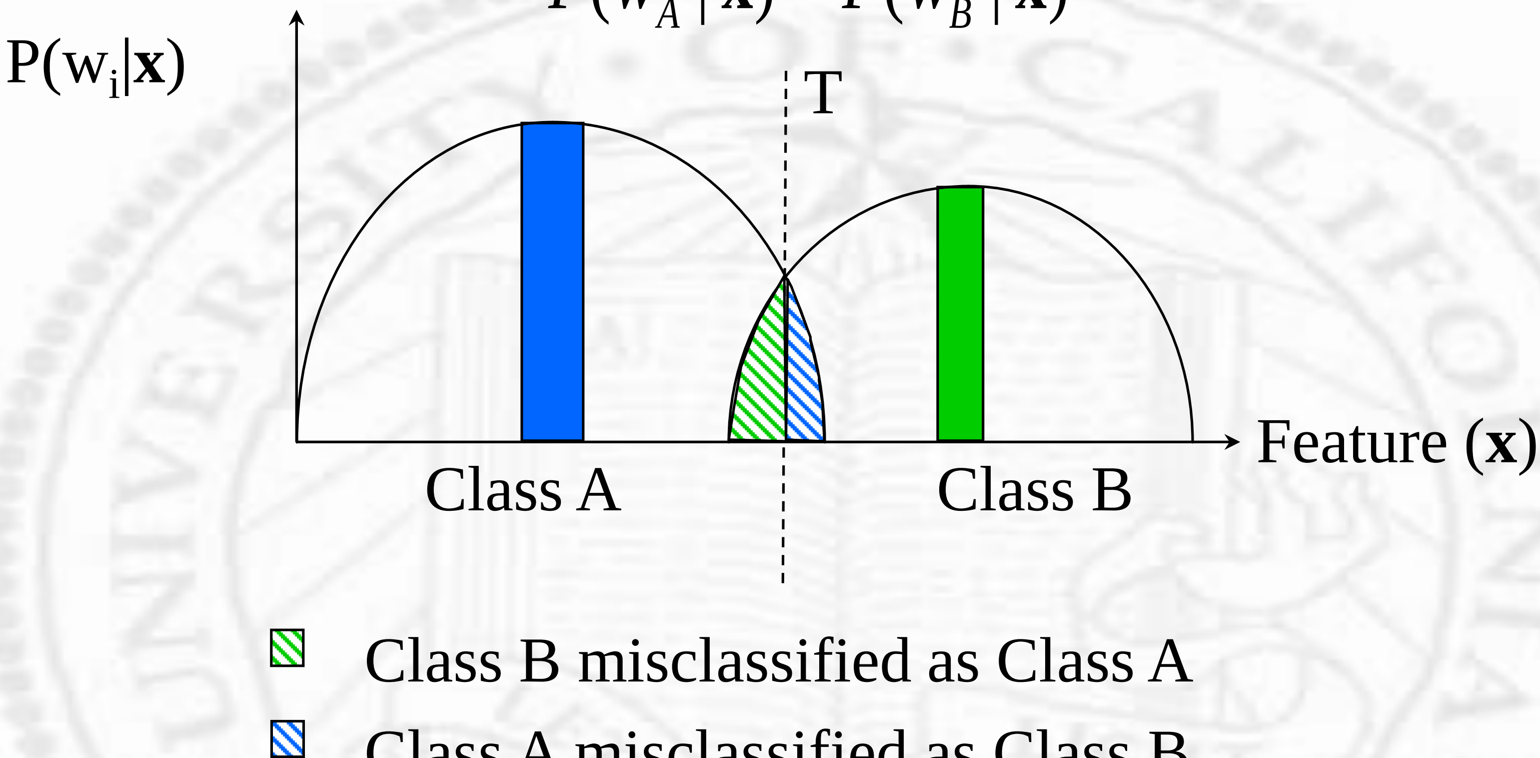
$$\sum_{x \in \Im} (\mathbf{w}^{\mathbf{t}}\mathbf{x})^2$$

$$\boxed{\sum_{x \in \Im} \frac{(\mathbf{w}^{\mathbf{t}}\mathbf{x} - \mathbf{b})^2}{|\mathbf{x}|^2}}$$

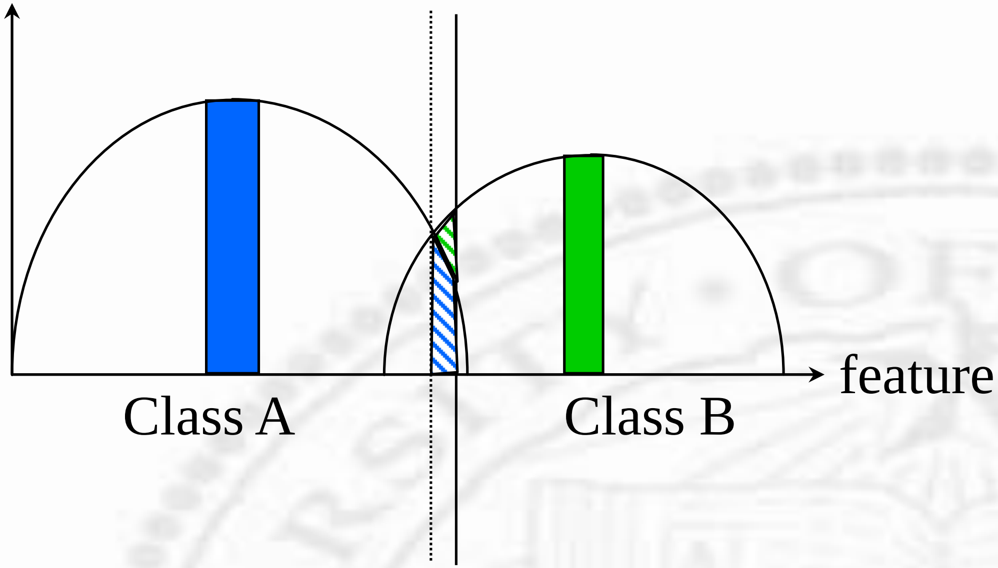
A bias term makes sure that solution is more in the center of the feasible region

# *How Good can a 2-Category Classifier be?*

$$P(w_A \mid \mathbf{x}) = P(w_B \mid \mathbf{x})$$

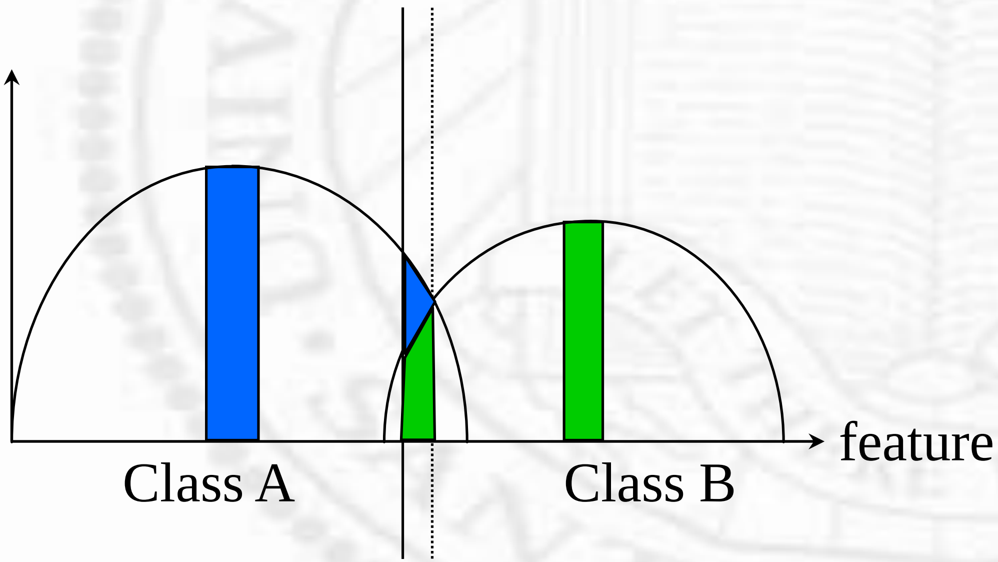



- As good as that by Bayesian rule





less Class A misclassified
< more Class B misclassified





less Class B misclassified
< more Class A misclassified

# *Implementation Details*

- Difficulty: features can be correlated
  - Un-correlate features using SVD
  - Add "regularization"

$$c(\mathbf{w}) = \sum_{x \in \Im} (-\mathbf{w}^t\mathbf{x}) + \lambda \sum \mathbf{w}^t\mathbf{w} \qquad \Im: \text{misclassified samples}$$

  - Numerically, the system is still quadratic so GD still works

# *Solving AX = B*

- Row interpretation
- Each row is a line
- Intersection of multiple lines
- Or
- Each row is a plane
- Multiple planes define a feasible region

- Column interpretation
- Each column is a vector
- Combination of these vectors to approximate B

# Non-iterative Method

$$\mathbf{w} = \arg\min_{\mathbf{w}} \left\{ \sum_i (y_i - w_o - \sum_j x_{ij} w_j)^2 \right\} \qquad y_i = \boxed{\begin{cases} 1 & \textit{positive} \\ 0 & \textit{negative} \end{cases}}$$

Valid only for regression problem
Fix it later in logistic regression

$$\mathbf{w} = \arg\min_{\mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$\frac{d(\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})}{d\mathbf{w}} = 0$$

$$\Rightarrow \mathbf{X}^T (\mathbf{y} - \mathbf{X}\mathbf{w}) = 0$$

$$\Rightarrow \mathbf{X}^T \mathbf{X}\mathbf{w} = \mathbf{X}^T \mathbf{y}$$

$$\Rightarrow \mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

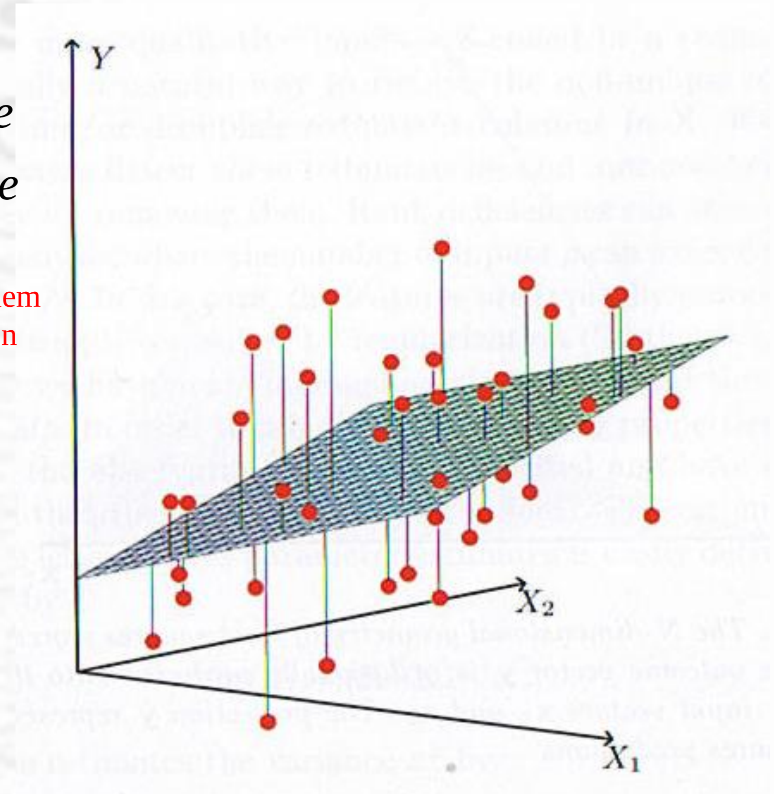
$$\hat{y} = < \mathbf{x}, (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} >$$

$$\mathbf{w} = [w_o, w_1, \cdots, w_d]^T,$$

$$\mathbf{x} = [1, x_1, \cdots, x_d]^T,$$

$$\mathbf{y} = [y_1, y_2, \cdots, y_n]^T$$

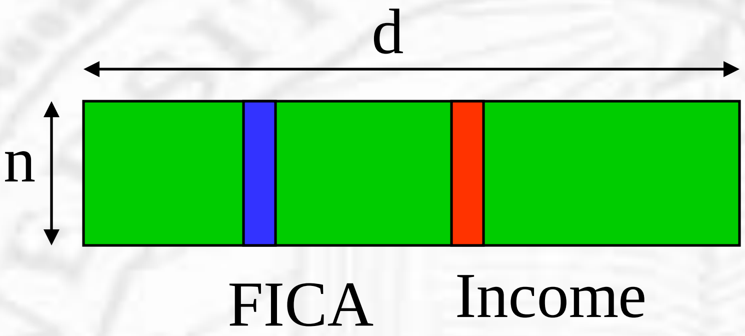
$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix}_{n \times d}$$



# *Graphical Interpretation*

$$\hat{\mathbf{y}} = \mathbf{X}\boxed{\mathbf{w}} = \boxed{\mathbf{H}}\mathbf{y} = \mathbf{X}\boxed{(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}} = \boxed{\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T}\mathbf{y}$$

**X=**

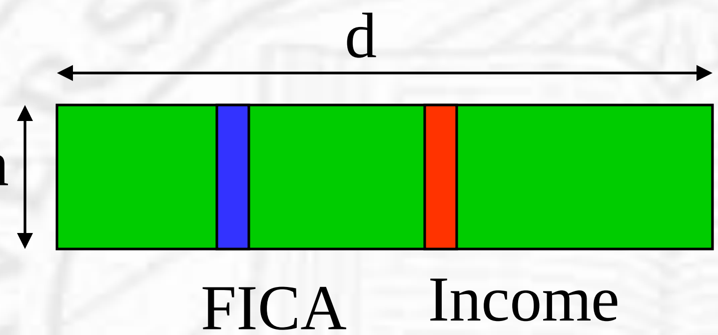


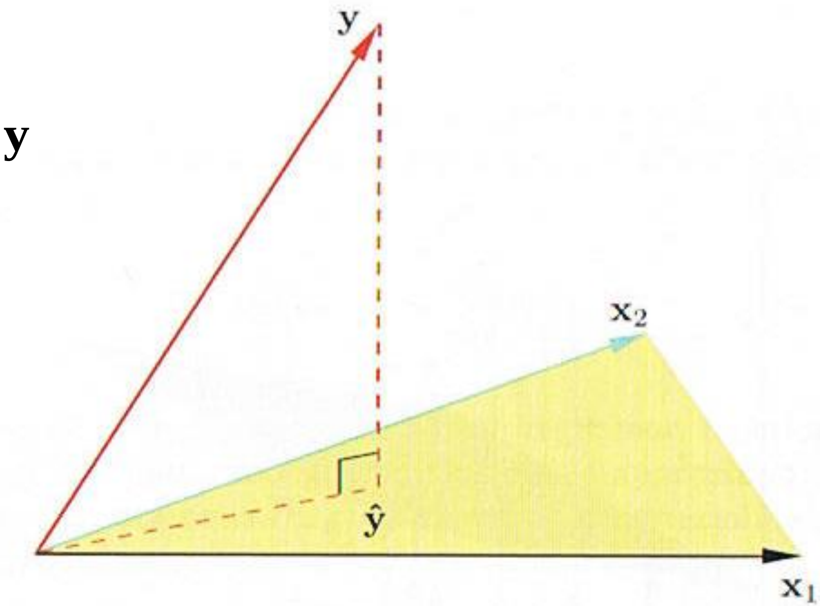

- **Xw:** classify training set X by learned parameter **w**
- **X** is a n (sample size) by d (dimension of data) matrix
- **w** combines the columns of $\mathbf{X}_{nxd}$ to best approximate $\mathbf{y}_{nx1}$
  - Combine features (FICA, income, etc.) to decisions (loan)
  - y^hat$_{nx1}$ is a combination of columns of $\mathbf{X}_{nxd}$
  - What is y^hat? How close is y^hat to y (GT)?

# *Graphical Interpretation*

$$\hat{\mathbf{y}} = \mathbf{X}\boxed{\mathbf{w}} = \boxed{\mathbf{H}}\mathbf{y} = \mathbf{X}\boxed{(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}} = \boxed{\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T}\mathbf{y}$$

**X=**




- **H** projects **y** onto the space spanned by columns of **X**
  - Simplify the decisions to fit the features

$$\mathbf{w} = \arg\min_{\mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) = \arg\min_{\mathbf{w}} (\mathbf{y} - \mathbf{H}\mathbf{y})^T(\mathbf{y} - \mathbf{H}\mathbf{y})$$

$$= \arg\min_{\mathbf{w}} (\mathbf{y} - \hat{\mathbf{y}})^T(\mathbf{y} - \hat{\mathbf{y}})$$

# Ugly Math

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \qquad \mathbf{X} = \mathbf{U\Sigma}V^T$$

$$\widehat{\mathbf{y}} = \mathbf{Xw} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

$$= \mathbf{U\Sigma}V^T(\mathbf{V\Sigma}^T\mathbf{U}^T\mathbf{U\Sigma}V^T)^{-1}\mathbf{V\Sigma}^T\mathbf{U}^T\mathbf{y}$$

$$= \mathbf{U\Sigma}V^T(\mathbf{V\Sigma\Sigma}V^T)^{-1}\mathbf{V\Sigma}^T\mathbf{U}^T\mathbf{y}$$

$$= \mathbf{U\Sigma}V^T(\mathbf{V}^{-T}\mathbf{\Sigma}^{-1}\mathbf{\Sigma}^{-1}\mathbf{V}^{-1})\mathbf{V\Sigma}^T\mathbf{U}^T\mathbf{y}$$

$$= \boxed{\mathbf{UU}^T}\mathbf{y}$$

$\mathbf{UU}^T$ is the standard form of a projection operator

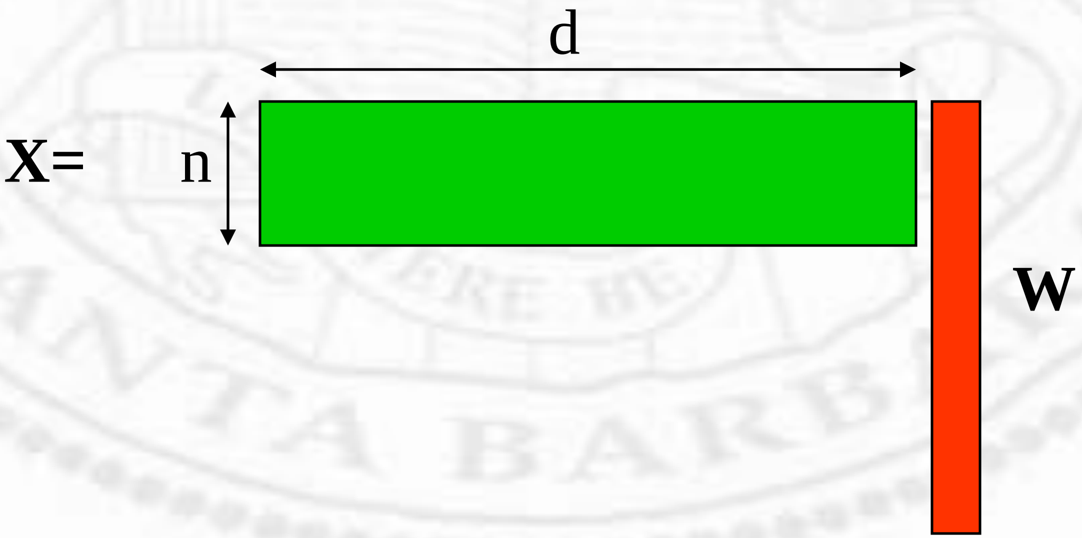
$\mathbf{U}^T$: inner product with the basis vector

$\mathbf{U}$: expand on the basis vector

($\mathbf{X}$ and $\mathbf{U}$ has the same column space)

# *Problem #1*

- n=d, exact solution
- n>d, least square, (most likely scenarios)
- When n < d, there are not enough constraints to determine coefficients **w** uniquely



# *Problem #2*

- If different attributes are highly correlated (income and FICA)
- The columns become dependent
- Coefficients are then poorly determined with high variance
  - E.g., large positive coefficient on one can be canceled by a similarly large negative coefficient on its correlated cousin
  - Size constraint is helpful
  - Caveat: constraint is problem dependent

# Ridge Regression (regularization)

$$\mathbf{w}^{ridge} = \arg\min_{\mathbf{w}} \left\{ \sum_i (y_i - w_o - \sum_j x_{ij} w_j)^2 + \lambda \sum_j w_j^2 \right\}$$

$$\mathbf{w}^{ridge} = \arg\min_{\mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w}$$

$$\frac{d(\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w}}{d\mathbf{w}} = 0$$

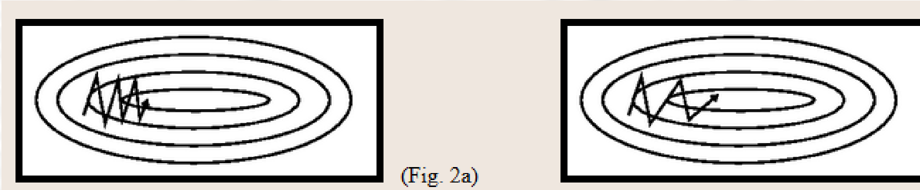
$$\Rightarrow -\mathbf{X}^T (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w} = 0$$

$$\Rightarrow \mathbf{X}^T \mathbf{y} = \mathbf{X}^T \mathbf{X}\mathbf{w} + \lambda \mathbf{w}$$

$$\Rightarrow \mathbf{X}^T \mathbf{y} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})\mathbf{w}$$

$$\Rightarrow \mathbf{w}^{ridge} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

$$\Rightarrow \hat{y} = < \mathbf{x}, (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y} >$$



(Fig. 2a)

$$\mathbf{w} = \arg\min_{\mathbf{w}} \left\{ \sum_i (y_i - w_o - \sum_j x_{ij} w_j)^2 \right\} \qquad y_i = \begin{cases} 1 & positive \\ 0 & negative \end{cases}$$

$$\hat{y} = < \mathbf{x}, (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} >$$

# Ugly Math

$$\mathbf{w}^{ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \qquad\qquad \mathbf{X} = \mathbf{U\Sigma}V^T$$

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}^{ridge} = \mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

$$= \mathbf{U\Sigma}V^T(\mathbf{V\Sigma}^T\mathbf{U}^T\mathbf{U\Sigma}V^T + \lambda\mathbf{I})^{-1}\mathbf{V\Sigma}^T\mathbf{U}^T\mathbf{y}$$

$$= \mathbf{U\Sigma}(V^{-T})^{-1}(\mathbf{V\Sigma}^T\mathbf{\Sigma}V^T + \lambda\mathbf{I})^{-1}(\mathbf{V}^{-1})^{-1}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{y}$$

$$= \mathbf{U\Sigma}(\mathbf{V}^{-1}\mathbf{V\Sigma}^T\mathbf{\Sigma}V^TV^{-T} + \mathbf{V}^{-1}\lambda\mathbf{I}V^{-T})^{-1}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{y}$$

$$= \mathbf{U\Sigma}(\mathbf{\Sigma}^T\mathbf{\Sigma} + \lambda\mathbf{I})^{-1}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{y}$$

$$= \sum_i \mathbf{u}_i \frac{\sigma_i^2}{\sigma_i^2 + \lambda}\mathbf{u}_i^T\mathbf{y}$$

# *How to Decipher This*

$$\hat{\mathbf{y}} = \sum_i \boxed{\mathbf{u}_i} \boxed{\frac{\sigma_i^2}{\sigma_i^2 + \lambda}} \boxed{\mathbf{u}_i^T \mathbf{y}}$$

- Red: best estimate (y hat) is composed of columns of **U** ("basis" features, recall **U** and **X** have the same column space)
- Green: how these basis columns are weighed
- Blue: projection of target (y) onto these columns
- Together: representing **y** in a body-fitted coordinate system ($\mathbf{u}_i$)

# *Sidebar*

- Recall that
  - Trace (sum of the diagonals) of a matrix is the same as the sum of the eigenvalues
  - Proof: every matrix has a standard Jordan form (an upper triangular matrix) where the eigenvalues appear on the diagonal (trace=sum of eigenvalues)
  - Jordan form results from a similarity transform (**PAP**$^{-1}$) which does not change eigenvalues

$$\begin{aligned}
&\mathbf{Ax} = \lambda \mathbf{x} \\
&\Rightarrow \mathbf{PAx} = \lambda \mathbf{Px} \\
&\Rightarrow \mathbf{PAP}^{-1}\mathbf{Px} = \lambda \mathbf{Px} \\
&\Rightarrow \mathbf{A}^{J}\mathbf{y} = \lambda \mathbf{y}
\end{aligned}$$

# *Physical Interpretation*

- Singular values of **X** represents the spread of data along different *body-fitting* dimensions (orthonormal columns)
- To estimate y$(=<\mathbf{x},\mathbf{w}^{ridge}>)$ regularization minimizes the contribution from less spread-out dimensions
  - Less spread-out dimensions usually have much larger variance (high dimension eigen modes) harder to estimate gradients reliably
  - Trace $\mathbf{X}(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T$ is called effective degrees of freedom

# More Details

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} = \mathbf{H}\mathbf{y} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

- Trace $\mathbf{X}(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T$ is called effective degrees of freedom
  - Controls how many eigen modes are actually used or active

$$df(\lambda) = d, \lambda = 0, df(\lambda) = 0, \lambda \to \infty$$

- Different methods are possible
  - Shrinking smoother: contributions are scaled
  - Projection smoother: contributions are used (1) or not used (0)

# *Dual Formulation (iterative)*

- Weight vector can be expressed as a sum of the *n* training feature vectors

Regular

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$
$$= \mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-2}\mathbf{X}^T\mathbf{y}$$
$$= \mathbf{X}^T{}_{d\times n}\boldsymbol{\alpha}_{n\times 1}$$
$$= \sum_i \alpha_i \mathbf{x}_i$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_1^T \end{bmatrix}_{n\times d} \qquad \mathbf{X}^T = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_1 \end{bmatrix}$$

Ridge regression

$$\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w}$$
$$\lambda\mathbf{w} = \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\mathbf{w}$$
$$\mathbf{w} = \frac{1}{\lambda}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$
$$= \mathbf{X}^T{}_{d\times n}\boldsymbol{\alpha}_{n\times 1}$$
$$= \sum_i \alpha_i \mathbf{x}_i$$

$$\mathbf{w}^{ridge} = \arg\min_{\mathbf{w}} \left\{ \sum_i (y_i - w_o - \sum_j x_{ij} w_j)^2 + \lambda \sum_j w_j^2 \right\}$$
$$\mathbf{w}^{ridge} = \arg\min_{\mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda\mathbf{w}^T\mathbf{w}$$
$$\Rightarrow -\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda\mathbf{w} = 0$$
$$\Rightarrow \mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w}$$

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} = \mathbf{H}\mathbf{y} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

# Dual Formulation (cont.)

$$\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w}$$

$$\lambda\mathbf{w} = \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\mathbf{w}$$

$$\mathbf{w} = \underline{\frac{1}{\lambda}}\mathbf{X}^T\underline{(\mathbf{y} - \mathbf{X}\mathbf{w})}$$

$$= \mathbf{X}^T{}_{d\times n}\boxed{\boldsymbol{\alpha}}_{n\times 1}$$

$$= \sum_i \alpha_i \mathbf{x}_i$$

$$\boldsymbol{\alpha}_{n\times 1} = \frac{1}{\lambda}(\mathbf{y} - \mathbf{X}_{n\times d}\mathbf{w}_{d\times 1})$$

$$\lambda\boldsymbol{\alpha} = \mathbf{y} - \mathbf{X}\mathbf{w}$$

$$\lambda\boldsymbol{\alpha} = \mathbf{y} - \mathbf{X}\mathbf{X}^T\boldsymbol{\alpha}$$

$$(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})\boldsymbol{\alpha} = \mathbf{y}$$

$$\boldsymbol{\alpha} = (\mathbf{X}_{n\times d}\mathbf{X}^T{}_{d\times n} + \lambda\mathbf{I})^{-1}\mathbf{y}_{n\times 1} = (\mathbf{G} + \lambda\mathbf{I})^{-1}\mathbf{y}$$

$$\mathbf{w} = \mathbf{X}^T\boldsymbol{\alpha} = \boldsymbol{\alpha}(\mathbf{G} + \lambda\mathbf{I})^{-1}\mathbf{y}$$

$$g(\mathbf{x}) =< \mathbf{w}, \mathbf{x} >= \mathbf{w}^T\mathbf{x} = \left\langle \sum \alpha_i \mathbf{x}_i, \mathbf{x} \right\rangle = \sum \alpha_i \langle \mathbf{x}_i, \mathbf{x} \rangle$$

$$=< \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}\mathbf{y}, \mathbf{x} >= \mathbf{y}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}\begin{bmatrix} \langle \mathbf{x}_1, \mathbf{x} \rangle \\ \langle \mathbf{x}_2, \mathbf{x} \rangle \\ \vdots \\ \langle \mathbf{x}_n, \mathbf{x} \rangle \end{bmatrix}$$

University of California Santa Barbara

# *In More Details*

Gram matrix

$$\begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix}_{1\times n} \left( \begin{bmatrix} - & \mathbf{x}_1^T & - \\ - & \cdots & - \\ - & \mathbf{x}_n^T & - \end{bmatrix}_{n\times d} \begin{bmatrix} | & | & | \\ \mathbf{x}_1 & \vdots & \mathbf{x}_n \\ | & | & | \end{bmatrix}_{d\times n} + \lambda \mathbf{I} \right)^{-1} \begin{bmatrix} - & \mathbf{x}_1^T & - \\ - & \cdots & - \\ - & \mathbf{x}_n^T & - \end{bmatrix}_{n\times d} \mathbf{x}$$

$$\begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix}_{1\times n} \begin{bmatrix} \mathbf{x}_1^T\mathbf{x}_1 + \lambda & \mathbf{x}_1^T\mathbf{x}_2 & . & \mathbf{x}_1^T\mathbf{x}_n \\ \mathbf{x}_2^T\mathbf{x}_1 & \mathbf{x}_2^T\mathbf{x}_2 + \lambda & . & \mathbf{x}_2^T\mathbf{x}_n \\ . & . & \ddots & . \\ \mathbf{x}_n^T\mathbf{x}_1 & \mathbf{x}_n^T\mathbf{x}_2 & & \mathbf{x}_n^T\mathbf{x}_n + \lambda \end{bmatrix} \begin{bmatrix} - & \mathbf{x}_1^T\mathbf{x} & - \\ - & \cdots & - \\ - & \mathbf{x}_n^T\mathbf{x} & - \end{bmatrix}_{n\times 1}$$

# Observations

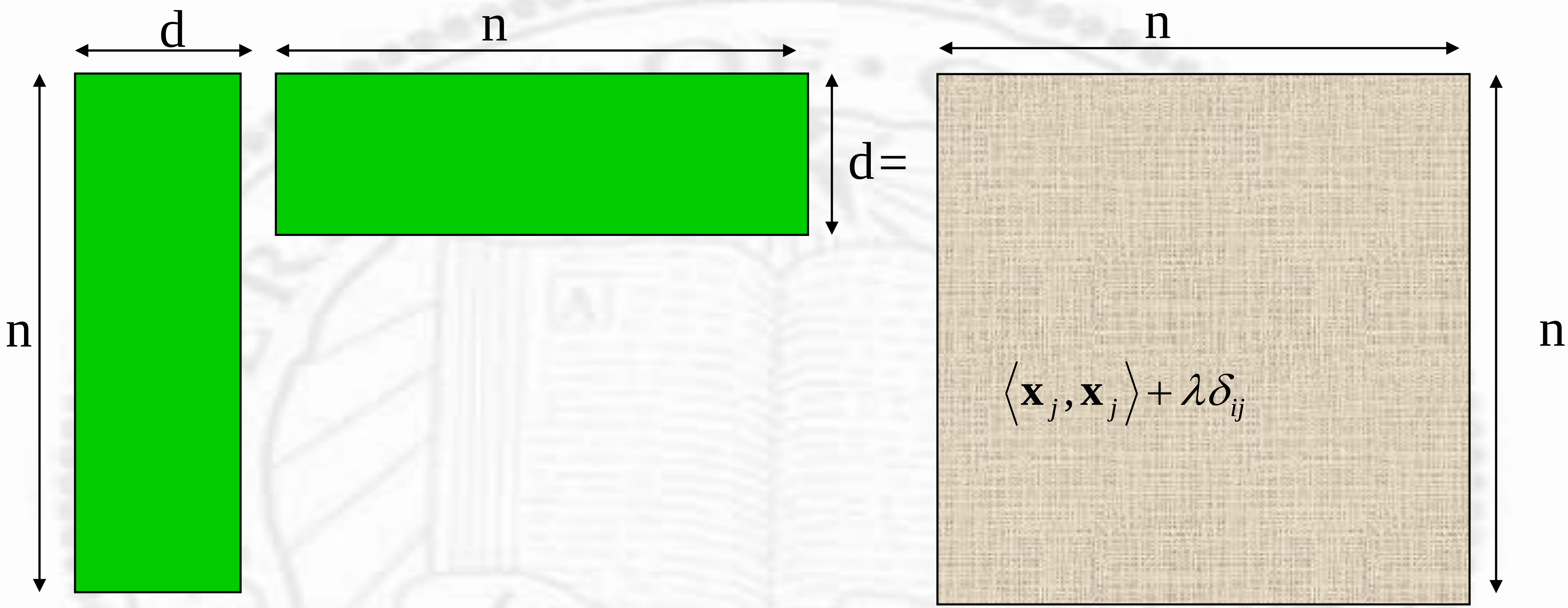
- Primary
- $\mathbf{X}^T\mathbf{X}$ is d by d
- Training: Slow for high feature dimension
- Use: fast O(d)

$$g(\mathbf{x}) = < \mathbf{x}_{d\times 1}, (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}_{d\times 1} >$$

- Dual
- Only inner products are involved
- $\mathbf{X}\mathbf{X}^T$ is n by n
- Training: Fast for high feature dimension
- Use: Slow O(nd)
  - N inner product to evaluate, each requires d multiplications

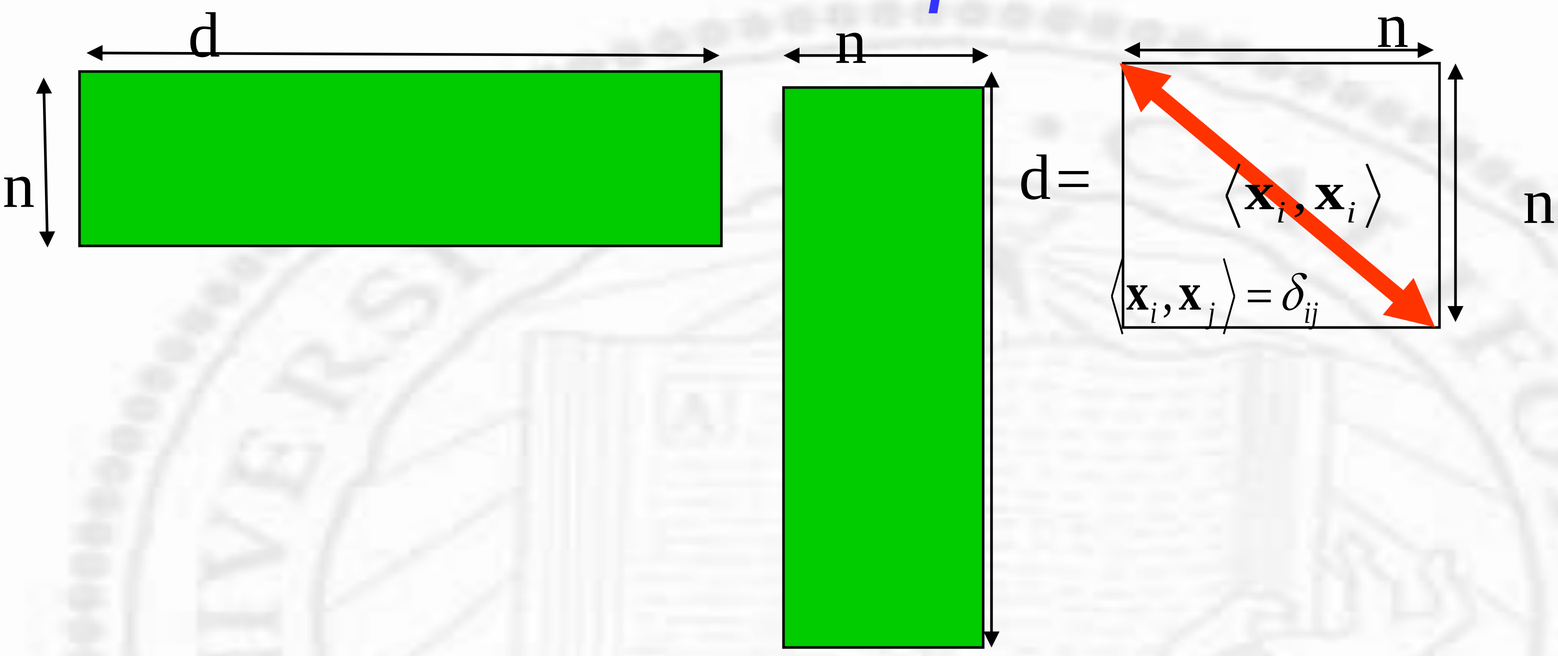
$$g(\mathbf{x}) = \mathbf{y}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}{}_{1\times n} \begin{bmatrix} \langle \mathbf{x}_1, \mathbf{x} \rangle \\ \langle \mathbf{x}_2, \mathbf{x} \rangle \\ \vdots \\ \langle \mathbf{x}_n, \mathbf{x} \rangle \end{bmatrix}_{n\times 1}$$

# *Graphical Interpretation*




$$g(\mathbf{x}) = <\mathbf{w}, \mathbf{x}> = \left\langle \sum \alpha_i \mathbf{x}_i, \mathbf{x} \right\rangle = \sum \alpha_i \langle \mathbf{x}_i, \mathbf{x} \rangle = \mathbf{y}^T (\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I})^{-1} \begin{bmatrix} \langle \mathbf{x}_1, \mathbf{x} \rangle \\ \langle \mathbf{x}_2, \mathbf{x} \rangle \\ \vdots \\ \langle \mathbf{x}_n, \mathbf{x} \rangle \end{bmatrix}$$

# *One Extreme – Perfect Uncorrelated*




$$g(\mathbf{x}) = \mathbf{y}^T (\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I})^{-1} \begin{bmatrix} \langle \mathbf{x}_1, \mathbf{x} \rangle \\ \langle \mathbf{x}_2, \mathbf{x} \rangle \\ \vdots \\ \langle \mathbf{x}_n, \mathbf{x} \rangle \end{bmatrix} = \sum_i y_i \frac{\langle \mathbf{x}_i, \mathbf{x} \rangle}{\langle \mathbf{x}_i, \mathbf{x}_i \rangle + \lambda}$$

- Orthogonal projection – no generalization

# General Case

$$\hat{\mathbf{y}}^T{}_{1\times n} = \mathbf{y}^T{}_{1\times n}(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}{}_{n\times n}\mathbf{X}_{n\times d}\mathbf{X}^T{}_{d\times n} \qquad \mathbf{X} = \mathbf{U}_{n\times d}\boldsymbol{\Sigma}_{d\times d}V^T{}_{d\times d}$$

$$= \mathbf{y}^T{}_{1\times n}(\mathbf{U}\boldsymbol{\Sigma}^2\mathbf{U}^T + \lambda\mathbf{I})^{-1}{}_{n\times n}\mathbf{U}\boldsymbol{\Sigma}V^T\mathbf{X}^T{}_{d\times n}$$

$$= \mathbf{y}^T{}_{1\times n}(\mathbf{U}(\boldsymbol{\Sigma}^2 + \lambda\mathbf{I})\mathbf{U}^T)^{-1}{}_{n\times n}\mathbf{U}\boldsymbol{\Sigma}V^T\mathbf{X}^T{}_{d\times n}$$

$$= \mathbf{y}^T{}_{1\times n}\mathbf{U}(\boldsymbol{\Sigma}^2 + \lambda\mathbf{I})^{-1}\mathbf{U}^{-1}\mathbf{U}\boldsymbol{\Sigma}V^T\mathbf{X}^T{}_{d\times n}$$

$$= \mathbf{y}^T{}_{1\times n}\mathbf{U}(\boldsymbol{\Sigma}^2 + \lambda\mathbf{I})^{-1}\boldsymbol{\Sigma}V^T\mathbf{X}^T{}_{d\times n}$$

$$= (\mathbf{U}^T{}_{d\times n}\mathbf{y}_{n\times 1})^T{}_{1\times d}(\boldsymbol{\Sigma}^2 + \lambda\mathbf{I})^{-1}\boldsymbol{\Sigma}V^T\boxed{\mathbf{X}^T{}_{d\times n}}$$

$$= (\mathbf{U}^T{}_{d\times n}\mathbf{y}_{n\times 1})^T{}_{1\times d}(\boldsymbol{\Sigma}^2 + \lambda\mathbf{I})^{-1}\boldsymbol{\Sigma}V^T\boxed{V\boldsymbol{\Sigma}\mathbf{U}^T}$$

$$= (\mathbf{U}^T{}_{d\times n}\mathbf{y}_{n\times 1})^T{}_{1\times d}(\boldsymbol{\Sigma}^2 + \lambda\mathbf{I})^{-1}\boldsymbol{\Sigma}^2\mathbf{U}^T$$

- How to interpret this? Does this still make sense?

# *Physical Meaning of SVD*

- Assume that $n > d$
- **X** is of rank $d$ at most
- **U** are the body (data)-fitted axes
- $\mathbf{U}^T$ is a projection from $n$ to $d$ space
- $\Sigma$ is the importance of the dimensions
- **V** is the representation of the **X** in the d space

$$\mathbf{X} = \mathbf{U}_{n\times d}\mathbf{\Sigma}_{d\times d}V^{T}{}_{d\times d}$$

# *Interpretation*

$$\hat{\mathbf{y}}^T{}_{1\times n} = (\mathbf{U}^T{}_{d\times n}\mathbf{y}_{n\times 1})^T{}_{1\times d}(\boldsymbol{\Sigma}^2 + \lambda\mathbf{I})^{-1}\boldsymbol{\Sigma}^2\mathbf{U}^T \longrightarrow \hat{\mathbf{y}} = \sum_i \mathbf{u}_i \frac{\sigma_i^2}{\sigma_i^2 + \lambda}\mathbf{u}_i{}^T\mathbf{y}$$

- In the new, uncorrelated space, there are only *d* training vectors and *d* decisions
- Red: *dx1* uncorrelated decision vector
- Green: weighting of the significance of the components in the uncorrelated decision vector
- Blue: transformed (uncorrelated) training samples
- Still the same interpretation: similarity measurement in a new space by
  - Gram matrix
  - Inner product of training samples and new sample

# *First Important Concept*

- The computation involves *only* inner product
  - For training samples in computing the Gram matrix
  - For new sample in computing regression or classification results
- Similarity is measured in terms of *angle*, instead of *distance*

# Second Important Concept

- Using angle or distance for similarity measurement doesn't make problems easier or harder
  - If you cannot separate data, it doesn't matter what similarity measures you use
- "Massage" data
  - Transform data (into higher – even infinite - dimensional space)
  - Data become "more likely" to be linearly separable (caveat: choice of the kernel function is important)
  - Cannot perform inner product efficiently
  - Kernel trick – do not have to

# *In reality*

- Calculating inverse of X^t X is very expensive
- The solution is by iteration
- Furthermore, features are often not used directly, but certain “nonlinear transformation” of features are used
- Furthermore, such “nonlinear transformation” is not calculated explicitly by Kernel trick

# *Math Detail*

$$X = [x_1^*, x_2^*, \ldots, x_N^*] \in R^{d \times N}$$

$$y \in R^d$$

Nonlinear transform

$$\varphi(X) = [\varphi(x_1^*), \varphi(x_2^*), \ldots, \varphi(x_N^*)] \in R^{D \times N}$$

$$\varphi(y) \in R^D$$

# Math Detail (cont)

$$\hat{\theta} = \min_{\theta} \left( \left\| \varphi(y) - \sum_{i=1}^{N} \theta_i \varphi(x_i^*) \right\|^2 + \mu \|\theta\|_1 \right)$$

$$\hat{\theta} = \min_{\theta} \left( k(y, y) - 2k(\cdot, y)^T \theta + \theta^T K \theta + \mu \|\theta\|_1 \right)$$

$$k(\cdot, y) = (k(x_1^*, y), k(x_2^*, y), \ldots, k(x^*_N, y))^T$$

$$K = \begin{pmatrix} k(x_1^*, x_1^*) & \cdots & k(x_1^*, x^*_N) \\ \vdots & \ddots & \vdots \\ k(x^*_{\_N}, x_1^*) & \cdots & k(x^*_{\_N}, x^*_{\_N}) \end{pmatrix}.$$

# Math Details (cont.)

$$J(\theta) = k(y,y) - 2k(\cdot,y)^T\theta + \theta^T K\theta + \mu\|\theta\|_1$$

$$\frac{\partial J(\theta)}{\partial \theta_i} = 2\sum_{j=1}^{N}\theta_j k(x_j^*, x_i^*) - 2k(x_i^*, y) + \mu\,\mathrm{sgn}(\theta_i) = 0$$

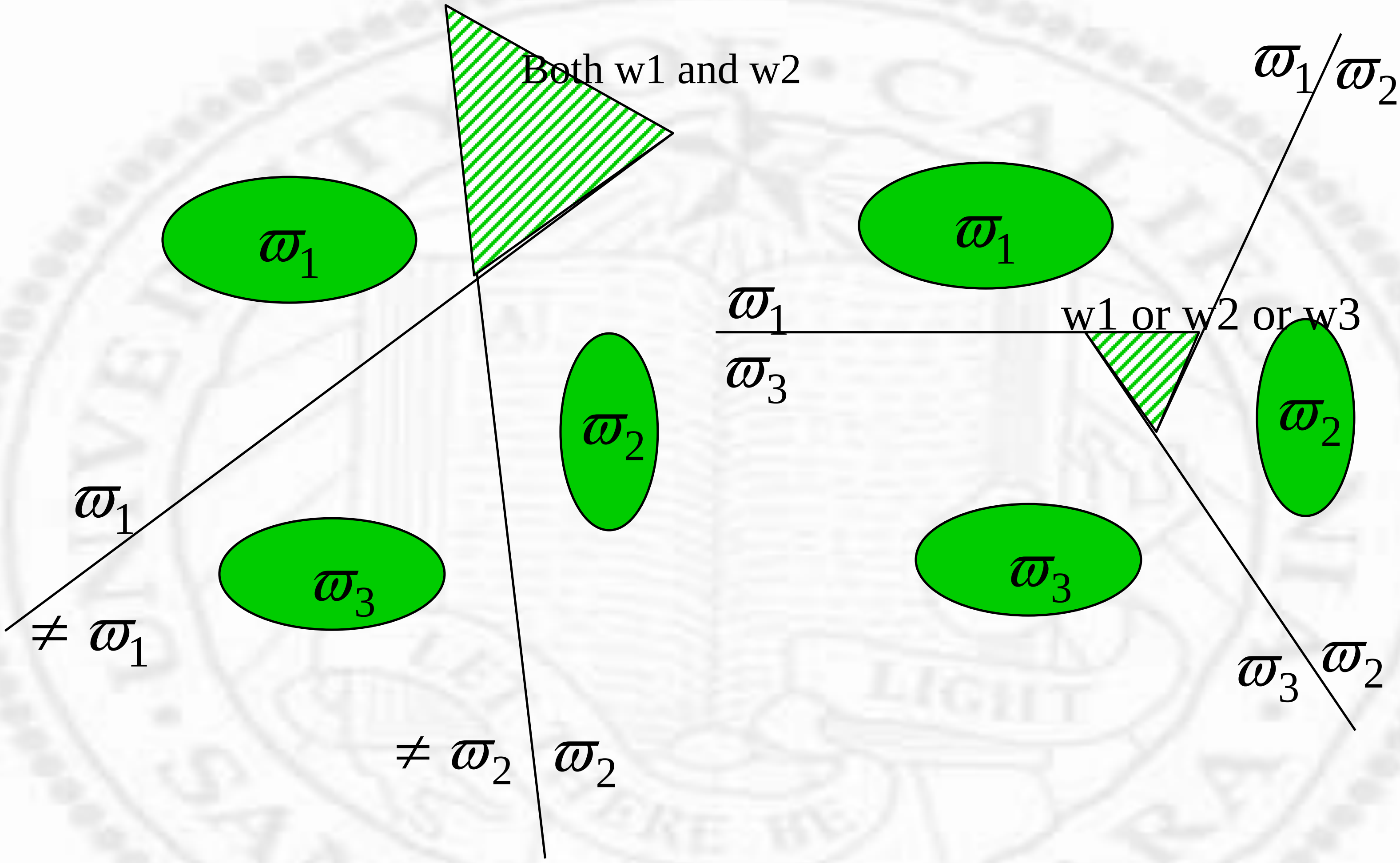
$$\theta_i = k(x_i^*, y) - \sum_{j=1, j\neq i}^{N}\theta_j k(x_j^*, x_i^*) - \frac{\mu}{2}\mathrm{sgn}(\theta_i)$$

$$\theta_i = w_\theta(x_i) - \frac{\mu}{2}\mathrm{sgn}(\theta_i) \qquad w_\theta(x_i) = k(x_i, y) - \sum_{j=1, j\neq i}^{N}\theta_j k(x_j, x_i)$$

# Math Details

- This represents a Gauss-Siedal iterative solution to the problem

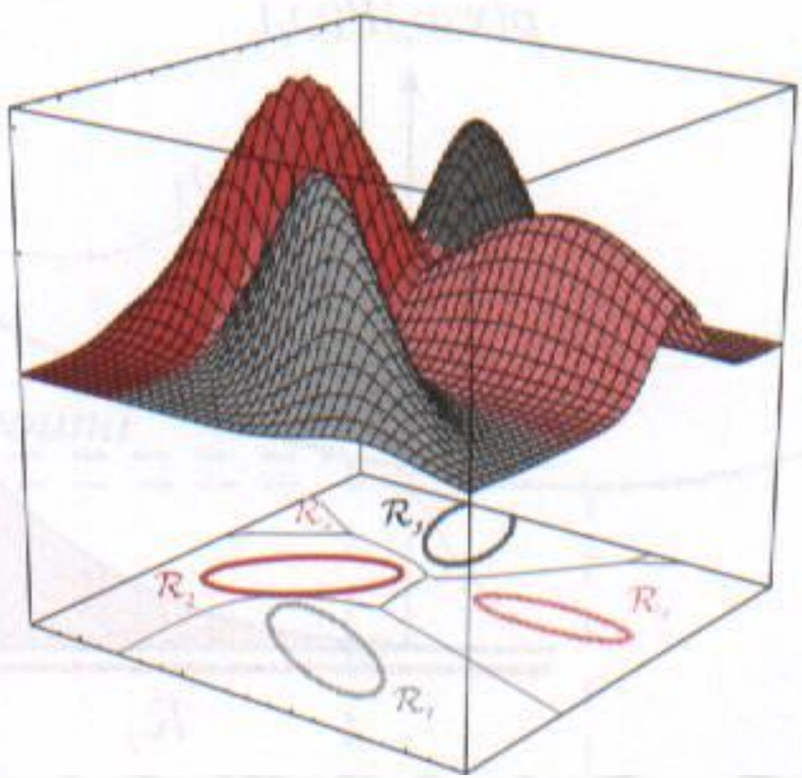
# Multi-category case




c-1 two-category
1 against all for all $w_i$

c(c-1)/2 two-category

# Multi-category Case (cont.)



# *Multi-category case*

- Theoretical (Kesler's) construction
- Assume linear separability

$$c \text{ classes} \quad \mathbf{w}_i \; i = 1, \ldots, c$$

$$\mathbf{w}_i \mathbf{x}_k - \mathbf{w}_j \mathbf{x}_k > 0 \; \text{for all} \; j \neq i \; \text{if} \; \mathbf{x}_k \in \varpi_i$$

$$\hat{\mathbf{w}} = [\mathbf{w}_1^T \quad . \quad . \quad \mathbf{w}_c^T]^T{}_{(c \times d)x1}$$

$$\hat{\mathbf{x}}_{12} = \left[\mathbf{x}^T \quad -\mathbf{x}^T \quad 0 \quad . \quad . \quad 0\right]^T{}_{(c \times d)x1}$$

$$\hat{\mathbf{x}}_{13} = \left[\mathbf{x}^T \quad 0 \quad -\mathbf{x}^T \quad . \quad . \quad 0\right]^T{}_{(c \times d)x1}$$

$$\hat{\mathbf{x}}_{1n} = \left[\mathbf{x}^T \quad 0 \quad 0 \quad . \quad . \quad -\mathbf{x}^T\right]^T{}_{(c \times d)x1}$$

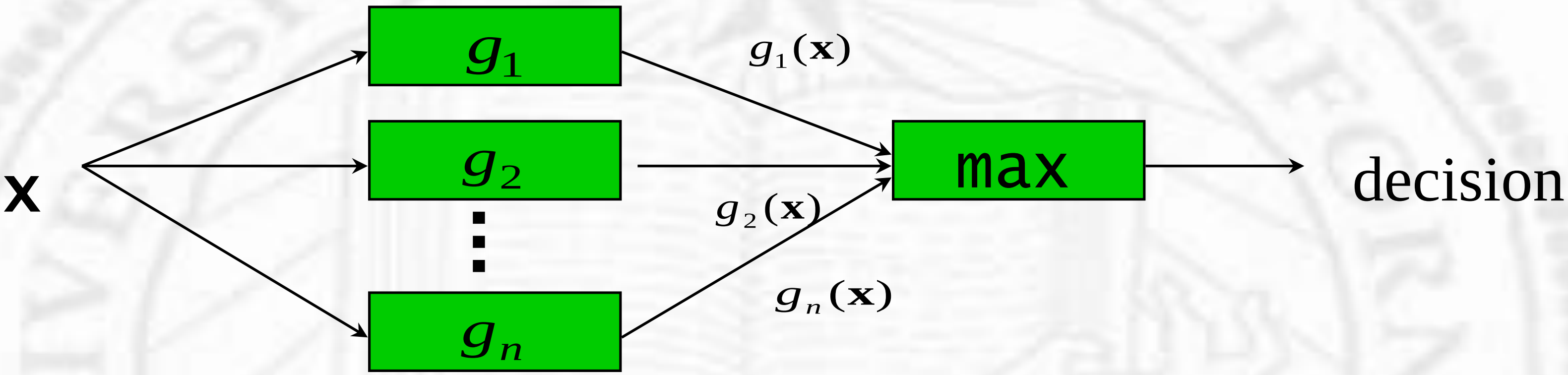
one weight $\hat{\mathbf{w}}$ $(c \cdot d$ dimension) must classify

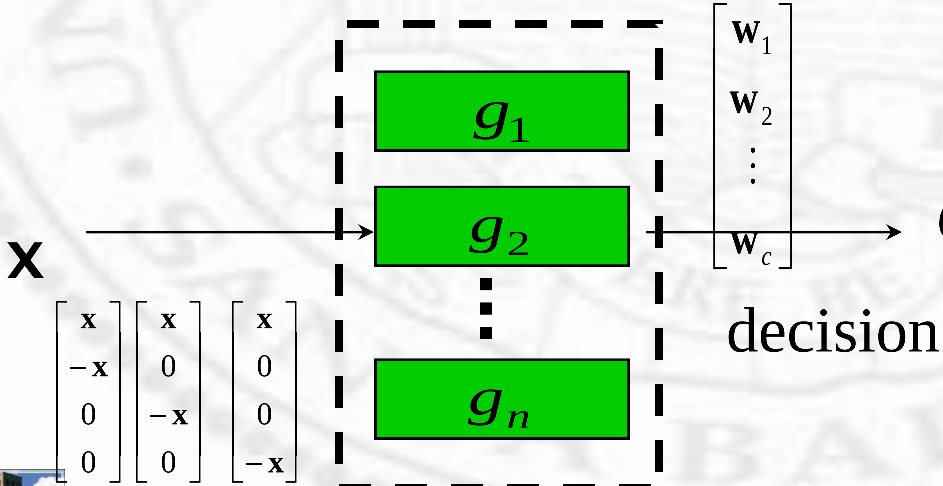
$c - 1$ samples $\hat{\mathbf{x}}_{12}, \hat{\mathbf{x}}_{13}, \ldots, \hat{\mathbf{x}}_{1n}$ $(c \cdot d$ dimension) correctly

# Graphical Interpretation

$$g_i(\mathbf{x}) = \mathbf{w}_i^t \mathbf{x} + w_{i0} \qquad i = 1, \dots, c$$

$$\varpi_i \qquad g_i(\mathbf{x}) - g_j(\mathbf{x}) > 0 \text{ for all } j \neq i$$




$$\begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_c \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{x} \\ -\mathbf{x} \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 0 \\ -\mathbf{x} \\ 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 0 \\ 0 \\ -\mathbf{x} \end{bmatrix}$$

$$G\left(\begin{bmatrix} \mathbf{x} \\ -\mathbf{x} \\ 0 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} \mathbf{w}_1 & \mathbf{w}_2 & & \mathbf{w}_c \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ -\mathbf{x} \\ 0 \\ 0 \end{bmatrix}$$

# *Kesler Construction*

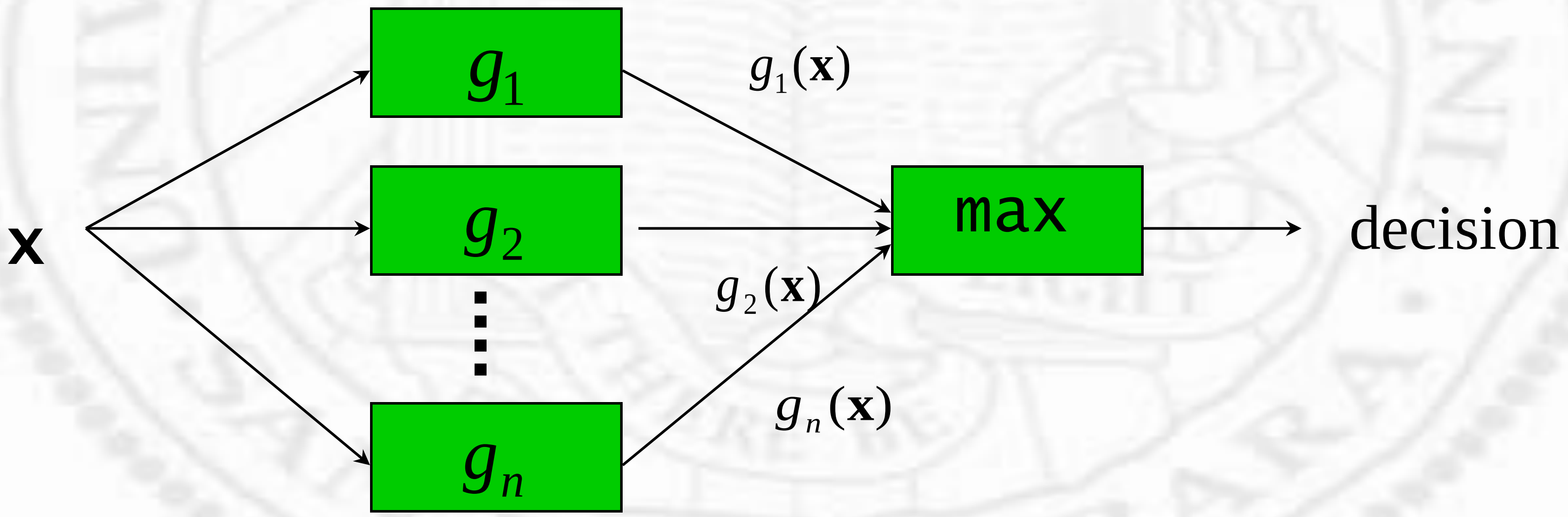
- Training
- "faked" 2-class
- One big $w=[w_1 \ldots w_c]$
- Every training sample is duplicated (1 against c-1) to generate c-1 positive samples
- Standard 2-class iterative gradient descent training

- Classification
- Break down w into c components $w_1 \ldots w_c$
- Evaluate a sample against all $w_i$ $(x.w_i)$
- Take the largest one as result

# Linear Machine

$$g_i(\mathbf{x}) = \mathbf{w}_i^t \mathbf{x} + \mathbf{w}_{i0} \qquad i = 1, \dots, c$$

$$\varpi_i \qquad g_i(\mathbf{x}) > g_j(\mathbf{x}) \text{ for all } j \neq i$$



# Multiple-categories

- Kesler construction does not detect boundies
- Find cluster center

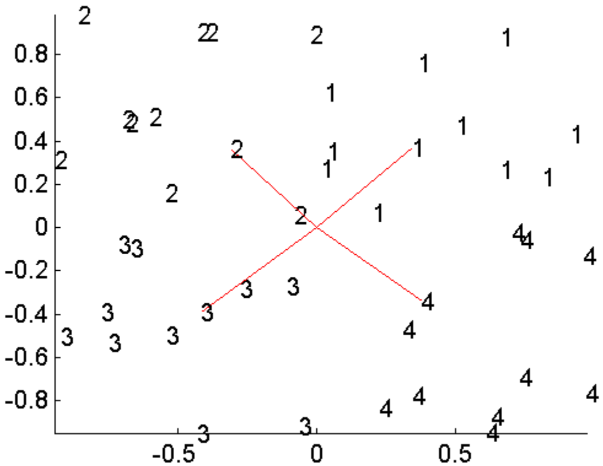
$$\begin{bmatrix} - & f_1 & - & - \\ - & f_2 & - & - \\ \cdots & \cdots & \cdots & \cdots \\ - & f_n & - & - \end{bmatrix}_{n\times d} \begin{bmatrix} | & | & | & | \\ | & | & | & | \\ w_1 & w_2 & \cdots & w_c \\ | & | & | & | \end{bmatrix}_{d\times c} = \begin{bmatrix} 1 & 0 & ... & 0 \\ 0 & ... & 1 & 0 \\ ... & ... & ... & ... \\ 0 & 1 & ... & 0 \end{bmatrix}_{n\times c}$$

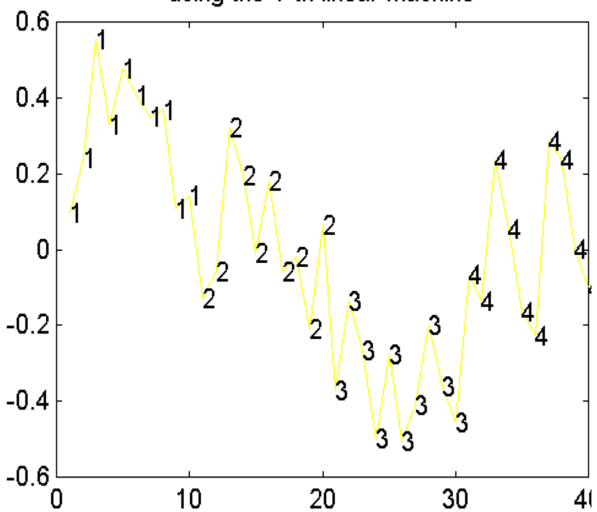
$n: samples$

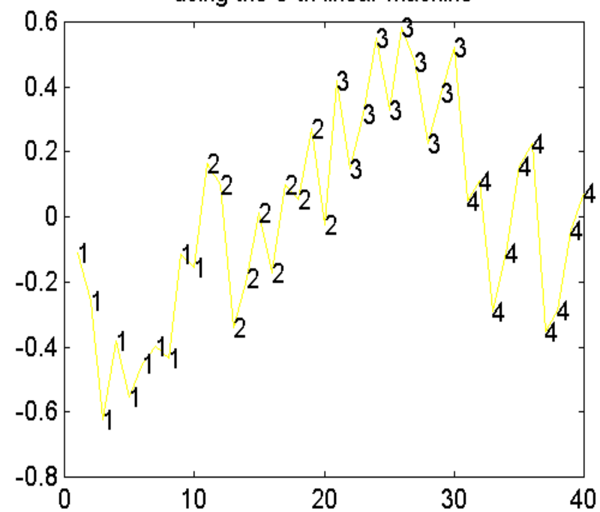
$d: \ features$

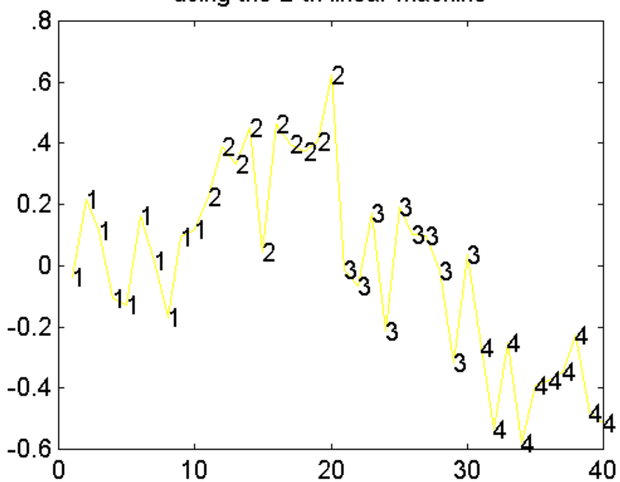
$c: classes$

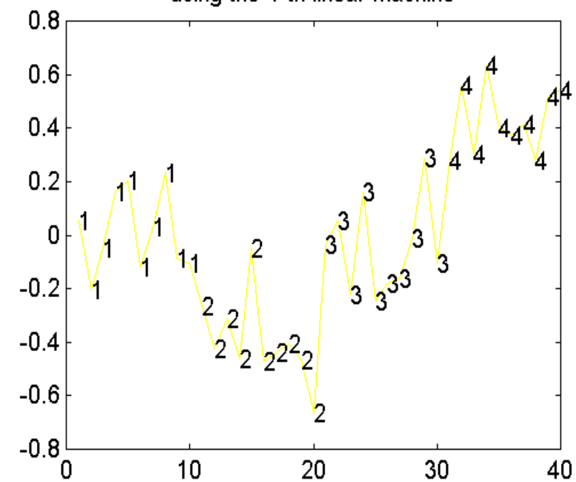
$$n = 40$$
$$c = 4$$
$$d = 2$$




using the 1-th linear machine


using the 3-th linear machine


using the 2-th linear machine


using the 4-th linear machine

# In Reality

- Linear Machine works if
  - samples in a class tight, compact clusters
  - class statistics are single mode (one single peak)
  - then, a class can be represented by a typical sample (class mean)
  - a case of nearest centroid classifier
  - otherwise ...

# *Linear Machine Example – Text Classification*

- Use standard TF/IDF weighted vectors to represent text documents (normalized by maximum term frequency).
- For each category, compute a *prototype* vector by summing the vectors of the training documents in the category.
- Assign test documents to the category with the closest prototype vector based on cosine similarity

# *Term Frequency*

- Term frequency(term, document): tf(t,d)
  - t: term, d: document
  - Raw frequency (f(t,d)): # of occurrences
  - Boolean frequency: 1 or 0
  - Log-scaled frequency: log (f(t,d)+1)
  - Augmented: adjusted for document length (/ by max raw freq of any term w in document d)

$$\mathrm{tf}(t,d) = 0.5 + \frac{0.5 \times \mathrm{f}(t,d)}{\max\{\mathrm{f}(w,d) : w \in d\}}$$

# *Inverse Document Frequency*

- N: total number of documents in corpus
- $|\{d \in D : t \in d\}|$ $\quad 1 + |\{d \in D : t \in d\}|$
  - number of documents where t appears

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

- Penalize common terms in corpus

# TF/IDF

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \times \mathrm{idf}(t, D)$$

- This is usually a very long vector, with n "keywords"
- Each document is described by such a long vector, recording occurrence of all keywords
- Again, the scheme is naïve Bayesian, correlation among terms (bi-grams, tri-grams, etc.) is ignored

# *Text Categorization, Rocchio (Training)*

- Assume the set of categories is $\{c_1, c_2, \ldots c_n\}$
- For $i$ from 1 to $n$ let $\mathbf{p}_i = \langle 0, 0, \ldots, 0 \rangle$ *(init. prototype vectors)*
- For each training example $\langle x, c(x) \rangle \in D$
- Let $\mathbf{d}$ be the frequency normalized TF/IDF term vector for doc $x$
- Let $i = j: (c_j = c(x))$
- *(sum all the document vectors in $c_i$ to get $\mathbf{p}_i$)*
- Let $\mathbf{p}_i = \mathbf{p}_i + \mathbf{d}$

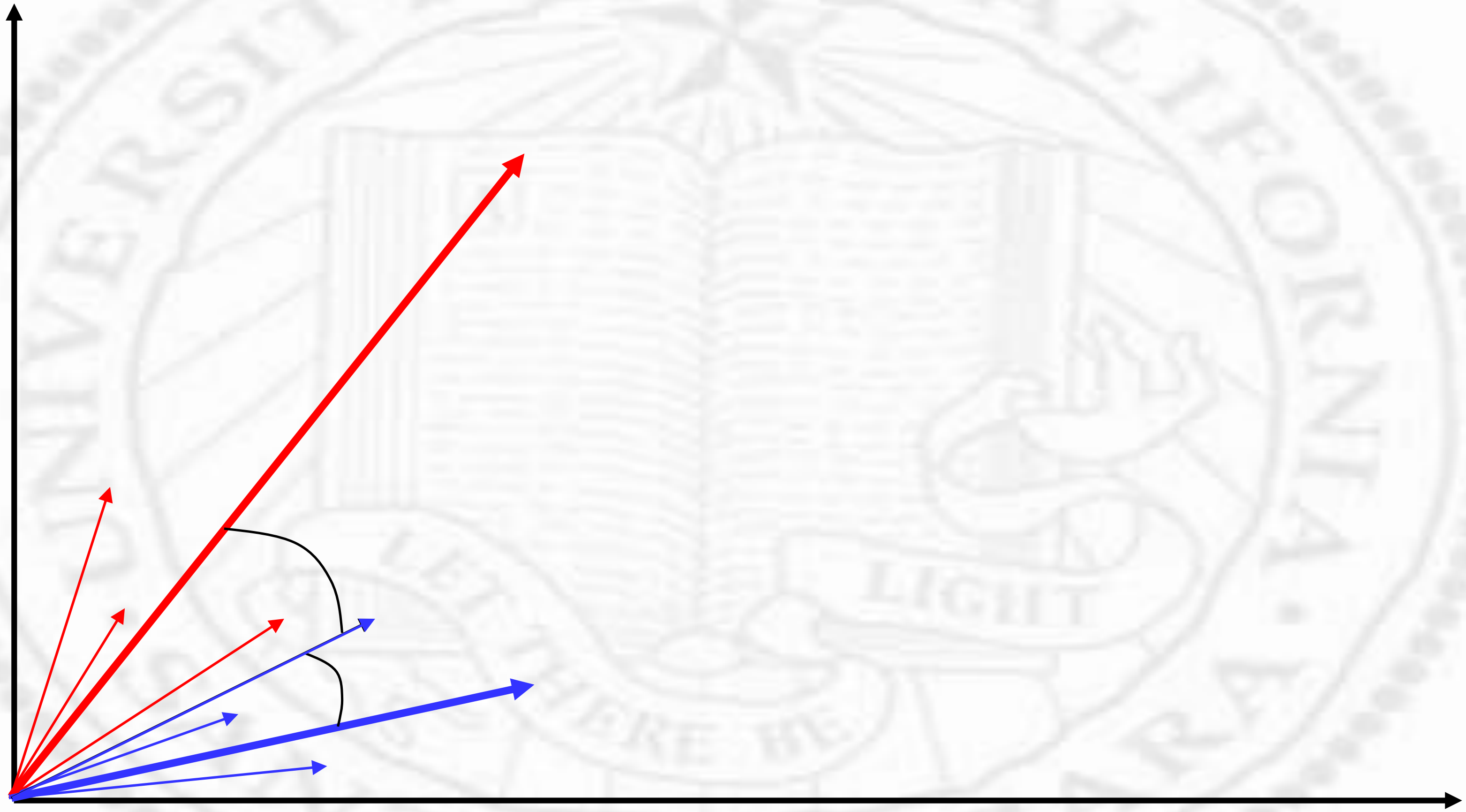


72

# *Rocchio Text Categorization (Test)*

- Given test document *x*
- Let **d** be the TF/IDF weighted term vector for *x*
- Let $m = -2$ (*init. maximum cosSim*)
- For *i* from 1 to *n*:
- (*compute similarity to prototype vector*)
- Let $s = \text{cosSim}(\mathbf{d}, \mathbf{p}_i)$
- if $s > m$
- let $m = s$
- let $r = c_i$ (*update most similar class prototype*)
- Return class *r*

# Illustration of Rocchio Text Categorization



# *Rocchio Properties*

- Does not guarantee a consistent hypothesis.
- Forms a simple generalization of the examples in each class (a *prototype*).
- Prototype vector does not need to be averaged or otherwise normalized for length since cosine similarity is insensitive to vector length.
- Classification is based on similarity to class prototypes.

# Other More Practical Classifiers

- Applicable for multiple classes
- Applicable for high feature dimensions
- Applicable for classes with multiple modes (peaks)

# *Two phases*

- Phase I (training): collect “tagged” (typical) samples from all classes, measure and record their features in the feature space (some statistics might be computed as well)
- Phase II (classification): given an unknown sample, classify that based on “similarity” or “ownership” in the feature space

# Nearest Centroid Classifier

$\mathbf{x}$ is in class $i$, if

$$|\mathbf{x}-\bar{\mathbf{x}}_i| \le |\mathbf{x}-\bar{\mathbf{x}}_j|, \; j=1,\ldots,n$$

- Need to record class centroids
- A single centroid -> linear machine model
- Multiple centroids possible (e.g. perform EM on mixture of Gaussian), but how do you find them if d>3?

# Nearest Neighbor Classifier

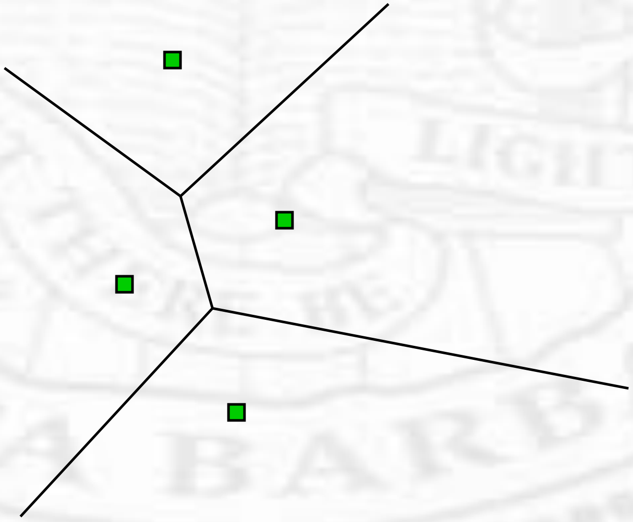
$\mathbf{x}$ is in class $i$, if $\exists k$

$$|\,\mathbf{x}-\bar{\mathbf{x}}_{i,k}\,| \le |\,\mathbf{x}-\bar{\mathbf{x}}_{j,l}\,|,\ j=1,\ldots,n, l=1,\ldots., m_j$$

- Do not need to record class centroids
- No analysis necessary
- Multiple modes/classes ok
- Need to remember all training data
- Computation efforts (distance checking)
- How about outliers?
- How about overfitting?

# Geometric Interpretation

- Nearest neighbor classifier performs Voronoi partition of the feature space
- In that sense, it is similar to assuming that different class distributions have the same prior and variance



# *K-Nearest Neighbor (k-NN)*

- Nearest neighbor can be susceptible to noise and outliers
- How about use more than 1? I.e. assign a sample to the class which has the most representatives among k nearest neighbors of the sample
- Intuitively appealing and followed from Parsen Windows & k-NN density estimation
- A compromise between nearest neighbor (too much data and erratic behaviors) and nearest centroid (global density fit)

# *k-NN classifier*

- Parsen window variant
- From density estimation to classifier (the same principle)
- n labeled training samples
- Given a query sample **x**, find k nearest samples from the training set
- Collect k *total* samples (for all classes), whichever class has the largest representation in the k samples wins

$$p_n(\mathbf{x},\varpi_i) = \frac{k_i / n}{V}$$

$$p_n(\varpi_i \mid \mathbf{x}) = \frac{p_n(\mathbf{x},\varpi_i)}{\sum_{j=1}^{c} p_n(\mathbf{x},\varpi_j)} = \frac{\frac{k_i / n}{V}}{\sum_{j=1}^{c} \frac{k_i / n}{V}} = \frac{\frac{k_i / n}{V}}{\frac{k / n}{V}} = \frac{k_i}{k}$$

$$\varpi_1 \Leftarrow k_1 > k_2$$

# k-NN Classifier (pool variant)

- We need at least k samples to maintain good resolution
- Assume the number of samples collected reflects the prior probability
- Collect the same # of samples (say, k), whichever class needs a smaller neighborhood to do that wins

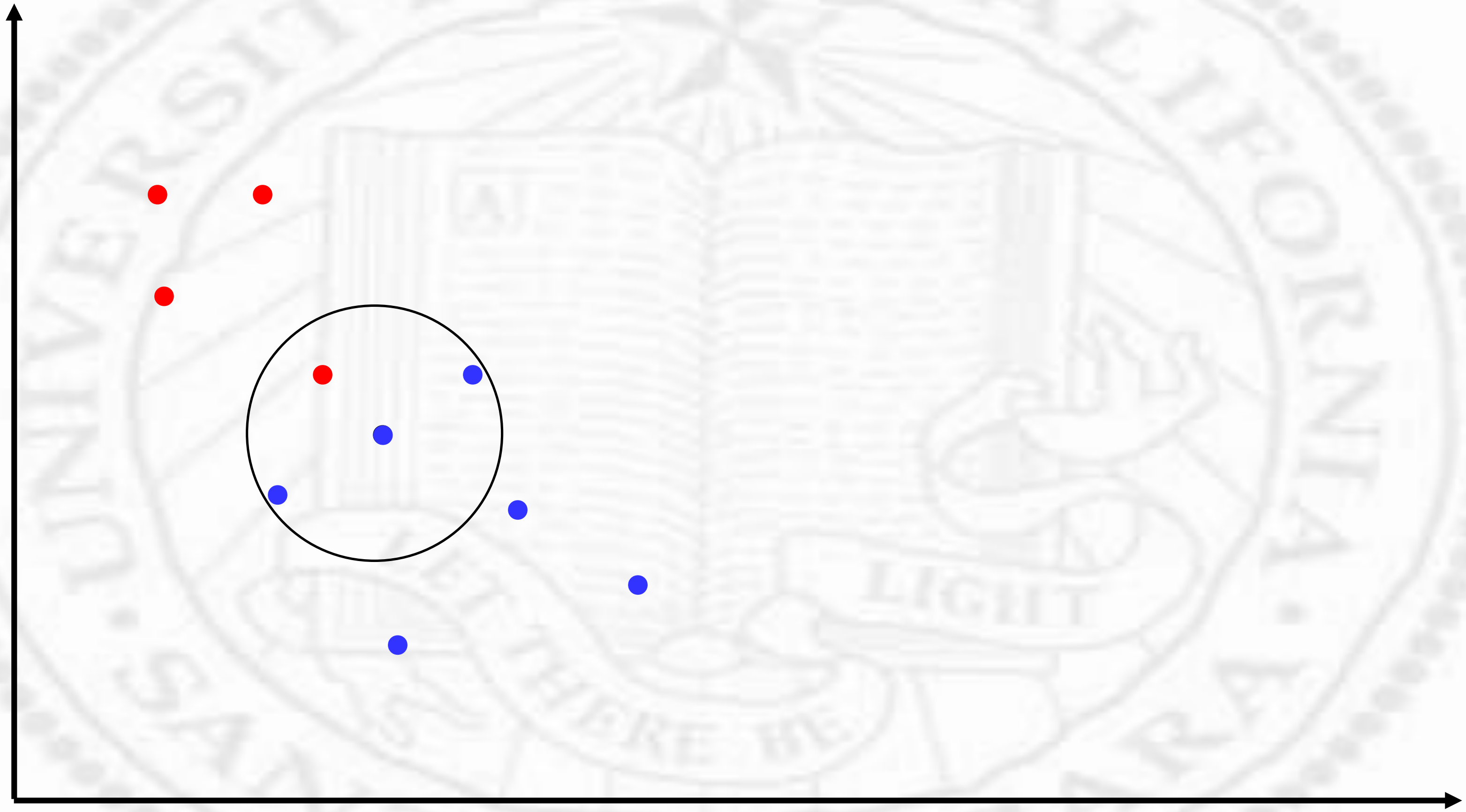
$$p_n(\mathbf{x}, \varpi_i) = \frac{k/n}{V_i}$$

$$p_n(\varpi_i \mid \mathbf{x}) = \frac{p_n(\mathbf{x}, \varpi_i)}{\sum_{j=1}^{c} p_n(\mathbf{x}, \varpi_j)} = \frac{\frac{k/n}{V_i}}{\sum_{j=1}^{c} \frac{k/n}{V_j}}$$

$$p_n(\varpi_1 \mid \mathbf{x}) = \frac{V_2}{V_1 + V_2} \qquad p_n(\varpi_2 \mid \mathbf{x}) = \frac{V_1}{V_1 + V_2}$$

$$\varpi_1 \Leftarrow V_2 > V_1$$

# 3 Nearest Neighbor Illustration
## (Euclidian Distance)



# *K Nearest Neighbor for Text*

**Training:**

For each each training example <$x$, $c(x)$> $\in D$

Compute the corresponding TF-IDF vector, $\mathbf{d}_x$, for document $x$

**Test instance *y*:**

Compute TF-IDF vector **d** for document $y$

For each <$x$, $c(x)$> $\in D$

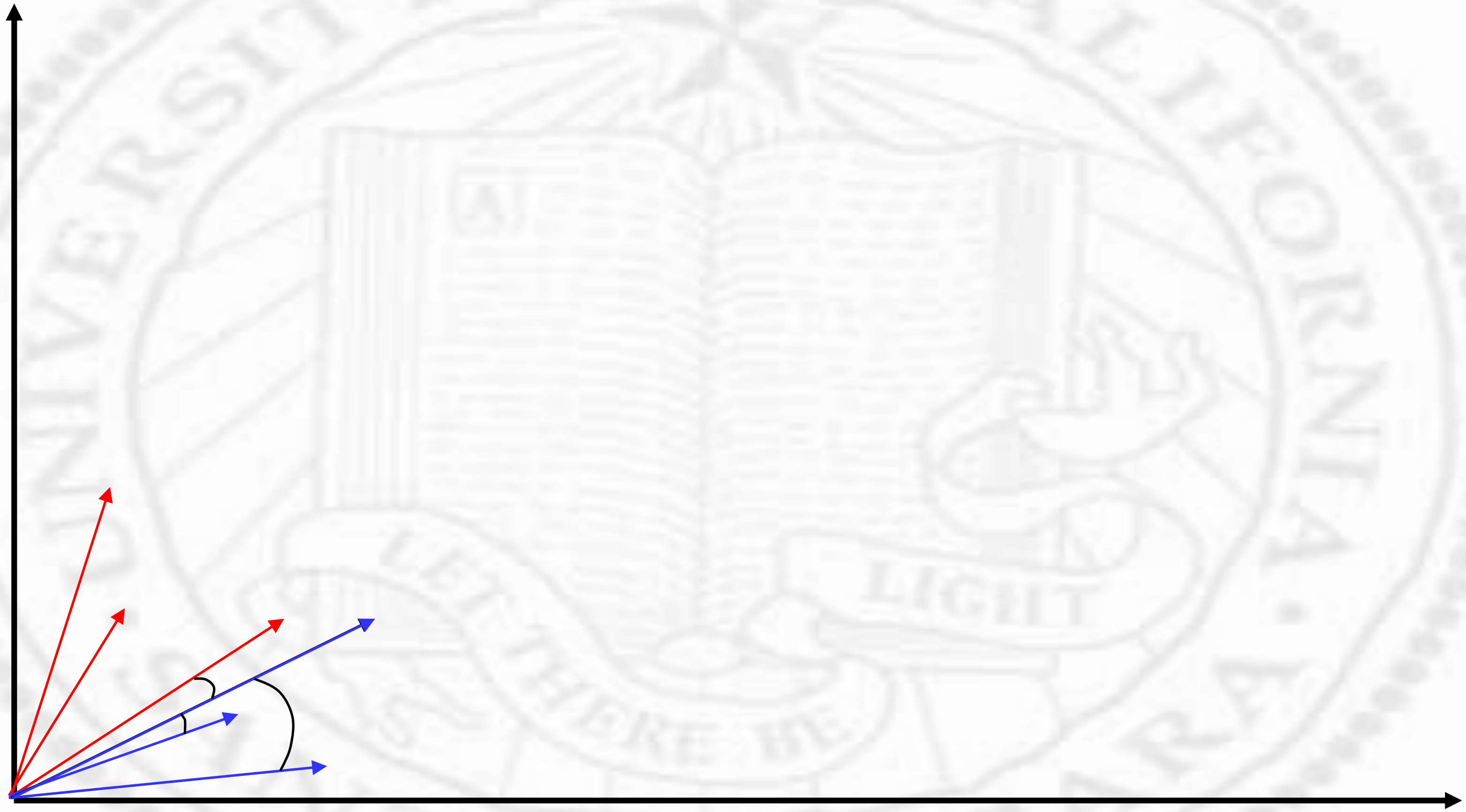
Let $s_x = \text{cosSim}(\mathbf{d}, \mathbf{d}_x)$

Sort examples, $x$, in $D$ by decreasing value of $s_x$

Let $N$ be the first $k$ examples in D. (*get most similar neighbors*)

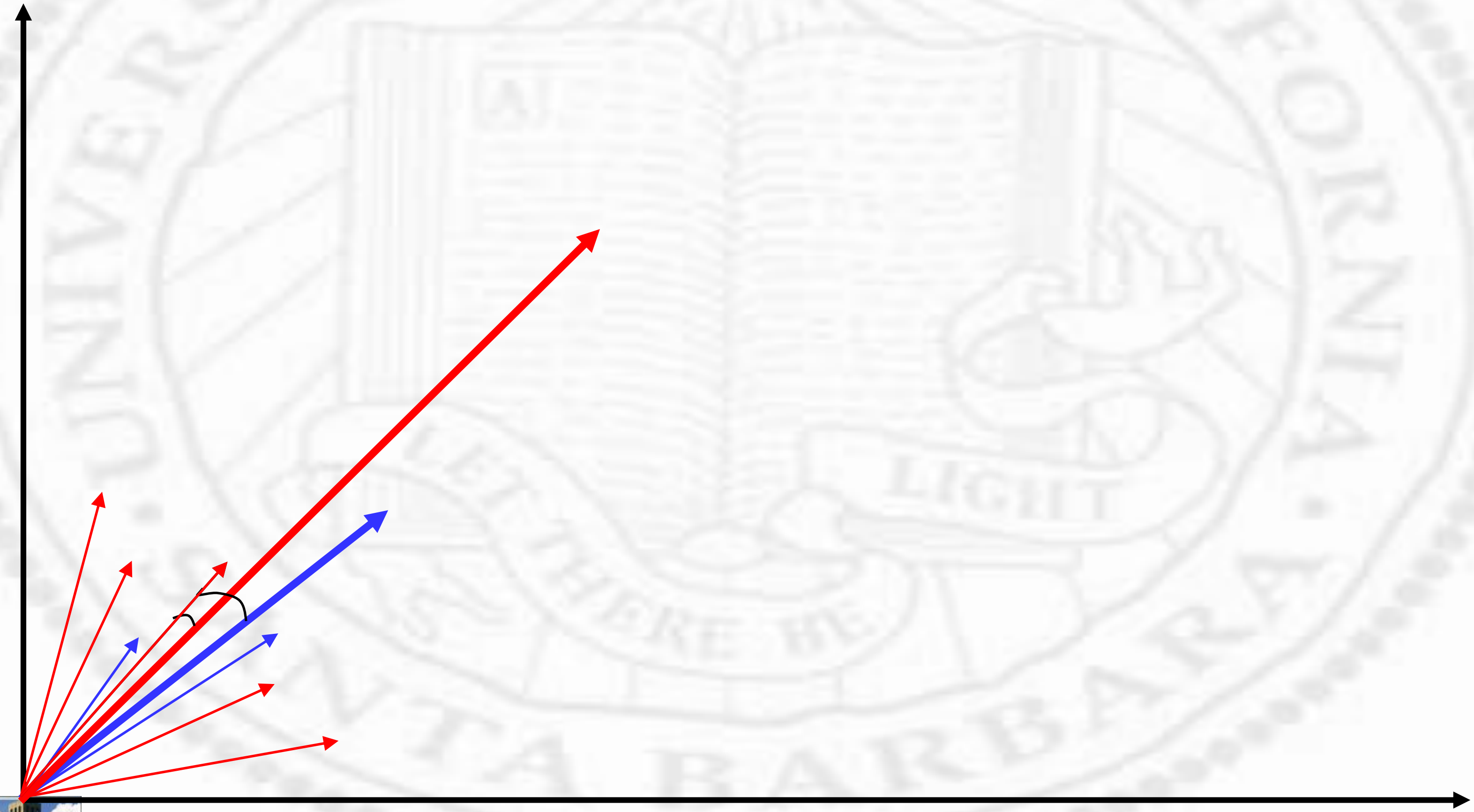
Return the majority class of examples in $N$

# *Illustration of 3 Nearest Neighbor for Text*



# *Rocchio Anomoly*

- Prototype models have problems with polymorphic (disjunctive) categories.




87

# 3 Nearest Neighbor Comparison

- Nearest Neighbor tends to handle polymorphic categories better.

# How Good Are the kNN?

- How good can it be?
  - Again, the best case scenario is the one dictated by Bayes rule: assign **x** to the class that most likely produces it based on *a posteriori* probability

$$P(w_m \mid \mathbf{x}) = \max_i P(w_i \mid \mathbf{x})$$

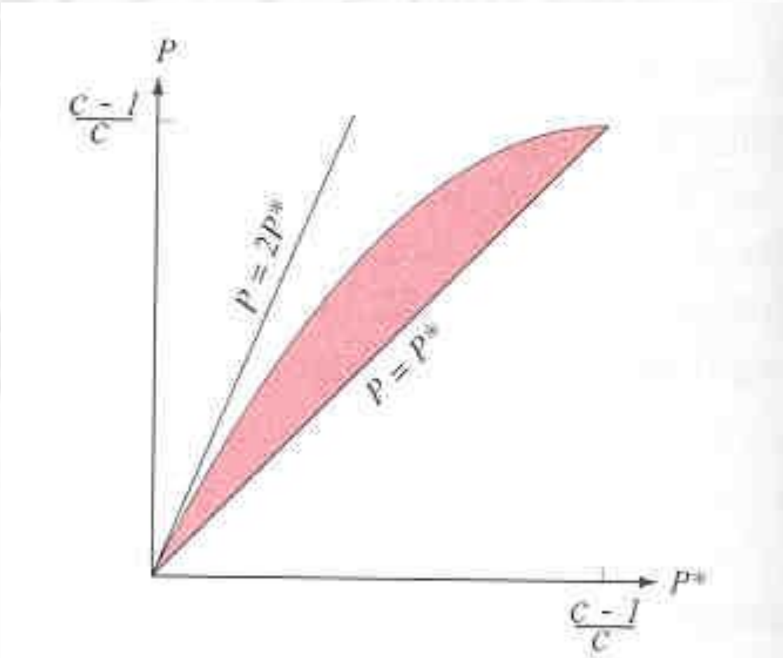
$$P^*(e \mid \mathbf{x}) = 1 - P(w_m \mid \mathbf{x})$$

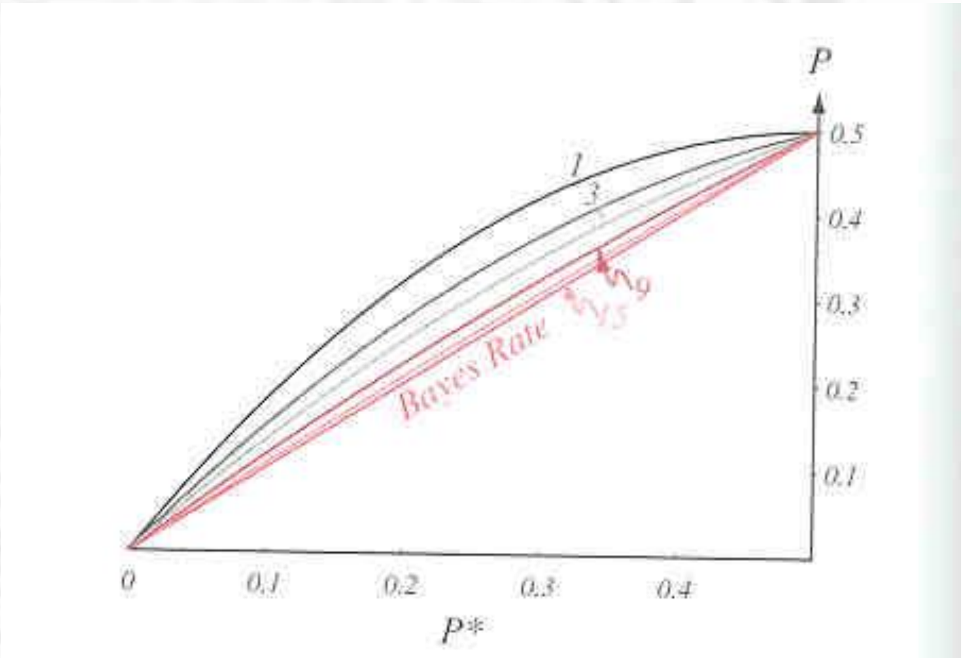
$$\boxed{P^* = \int P^*(e \mid \mathbf{x})\, p(\mathbf{x})\, d\mathbf{x}}$$

Holy Grail

# *However, knn's are not bad either*

- Surprisingly, 1nn (nearest) is not more than twice as bad as Bayesian and knn approaches Bayesian for large k



# *Interested in the Proof?*

- As promised, we don't do proof
- Instead, we rely on intuition
  - **x**: sample, **x**': nearest neighbor to **x**
  - θ: sample's class, θ': **x**' class
- Q: what is θ' ?
- A: $\arg\max_{i} P(w_i \mid \mathbf{x}') = P(w_m \mid \mathbf{x}')$
- With a large number of samples, it is reasonable to assume that **x**' is close to **x** $P(w_i \mid \mathbf{x}) = P(w_i \mid \mathbf{x}')$

- If $P(w_m|\mathbf{x}) \sim 1$ Bayes and 1nn likely produce the same results
- If $P(w_m|\mathbf{x}) \sim 1/c$ Bayes and 1nn likely produce different results, but both error rates are 1-1/c

# *Proof Sketch*

- We are looking for scenarios where **x** and **x**' (its nearest neighbor) belong to different classes θ and θ'
- In fact, we have to look at cases where the number of training samples are very very large
  - Because **x**' depends on the samples used in training and proof can not be based on the particular training set used
  - **x**' depends on n (samples used), we will write as $\mathbf{x}_n$' instead

# Proof Sketch (cont.)

- Error is when $\mathbf{x}$ and $\mathbf{x_n}$' are in different classes

$$P_n(e \mid \mathbf{x}, \mathbf{x}_n') = 1 - \sum_{i=1}^{c} P(\theta = \varpi_i, \theta_n' = \varpi_i \mid \mathbf{x}, \mathbf{x}_n')$$

$$= 1 - \sum_{i=1}^{c} P(\theta = \varpi_i \mid \mathbf{x}) P(\theta_n' = \varpi_i \mid \mathbf{x}_n')$$

  - Because all the training samples and test samples are drawn independently

- Average error cannot depend on $\mathbf{x_n}$' (which depends on the particular training sample set)

$$P_n(e \mid \mathbf{x}) = \int P(e \mid \mathbf{x}, \mathbf{x}_n') p(\mathbf{x_n}' \mid \mathbf{x}) d\mathbf{x}_n'$$

# Proof Sketch (cont.)

- Combine them together, we have

$$P_n(e|\mathbf{x}) = \int [1 - \sum_{i=1}^{c} P(\varpi_i|\mathbf{x})P(\varpi_i|\mathbf{x}_n')]p(\mathbf{x}_n'|\mathbf{x})d\mathbf{x}_n'$$

- When n is large, it is reasonable to expect **x** and **x**' are close

$$\lim_{n\to\infty} p(x_n'|\mathbf{x}) = \delta(\mathbf{x}_n' - \mathbf{x})$$

$$\lim_{n\to\infty} P_n(e|\mathbf{x}) = \lim_{n\to\infty} \int [1 - \sum_{i=1}^{c} P(\varpi_i|\mathbf{x})P(\varpi_i|\mathbf{x}_n')]p(\mathbf{x}_n'|\mathbf{x})d\mathbf{x}_n'$$

$$= \int [1 - \sum_{i=1}^{c} P(\varpi_i|\mathbf{x})P(\varpi_i|\mathbf{x}_n')]\delta(\mathbf{x}_n' - \mathbf{x})d\mathbf{x}_n'$$

$$= 1 - \sum_{i=1}^{c} P^2(\varpi_i|\mathbf{x})$$

Correct if
- x is in $w_i$
- Nearest sample is also in $w_i$
- i can be any class

# Proof Sketch (cont.)

- Then over all possible **x**'s

$$P = \lim_{n\to\infty} P_n(e) = \int \lim_{n\to\infty} P_n(e\,|\,\mathbf{x})\,p(\mathbf{x})d\mathbf{x}$$

$$= \int [1 - \sum_{i=1}^{c} P^2(\varpi_i\,|\,\mathbf{x})]\,p(\mathbf{x})d\mathbf{x}$$

- A quick check, if $P(w_m|\mathbf{x}) \sim 1$

$$1 - \sum_{i=1}^{c} P^2(\varpi_i\,|\,\mathbf{x}) \cong 1 - P^2(\varpi_m\,|\,\mathbf{x}) \cong 2\boxed{(1 - P(\varpi_m\,|\,\mathbf{x}))}$$

Holy Grail

$$\because 1 - x^2 = 1 - [1 - (1 - x)]^2$$
$$= 1 - (1 - 2(1 - x) + (1 - x)^2)$$
$$\cong 1 - (1 - 2(1 - x))$$
$$= 2(1 - x)$$

# Proof Sketch (cont.)

- Otherwise

$$\sum_{i=1}^{c} P^2(\varpi_i \mid \mathbf{x}) = P^2(\varpi_m \mid \mathbf{x}) + \sum_{i=1, i \neq m}^{c} P^2(\varpi_i \mid \mathbf{x})$$

- The second term is minimized if all of the other classes are equally likely

$$P(\varpi_i \mid \mathbf{x}) = \left\langle \begin{array}{ll} \dfrac{P^*(e \mid \mathbf{x})}{c-1} & i \neq m \\ 1 - P^*(e \mid \mathbf{x}) & i = m \end{array} \right.$$

# Proof Sketch (cont.)

$$\sum_{i=1}^{c} P^2(\varpi_i \mid \mathbf{x}) = (1 - P^*(e \mid \mathbf{x}))^2 + \frac{P^{*2}(e \mid \mathbf{x})}{(c-1)^2}$$

$$\geq (1 - P^*(e \mid \mathbf{x}))^2 + \frac{P^{*2}(e \mid \mathbf{x})}{c-1}$$

$$1 - \sum_{i=1}^{c} P^2(\varpi_i \mid \mathbf{x}) \leq 1 - (1 - P^*(e \mid \mathbf{x}))^2 - \frac{P^{*2}(e \mid \mathbf{x})}{c-1}$$

$$\Rightarrow \leq 2P^*(e \mid \mathbf{x}) - P^{*2}(e \mid \mathbf{x}) - \frac{P^{*2}(e \mid \mathbf{x})}{c-1}$$

$$\Rightarrow \leq 2P^*(e \mid \mathbf{x}) - \frac{c}{c-1} P^{*2}(e \mid \mathbf{x})$$

$$\Rightarrow \leq 2P^*(e \mid \mathbf{x})$$

An even tighter bound :

$$P^* \leq P \leq P^*\left(2 - \frac{c}{c-1} P^*\right)$$

# Other Variations

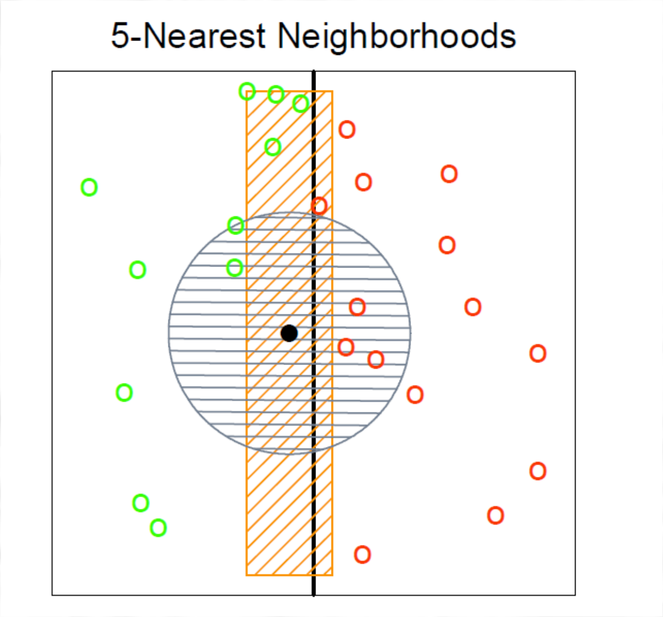
- Distance weighted: vote is weighed by how close a training sample is to the test sample
- Dimension weighted: distance is calculated by weighing features unequally
  - Weights can be learned by cross-validation

# *Adaptive Nearest Neighbors*

- Important for high-dimensional feature space where neighbors are far apart
- Idea: find local regions and compute feature dimensions
  - Where class labels change a lot – narrower focus
  - Where class labels doesn't change a lot – wider focus

# Adaptive Nearest Neighbors (cont.)

- Two classes and two features
- Uniform distribution but label changes only in x
- Extent  y to capture more features

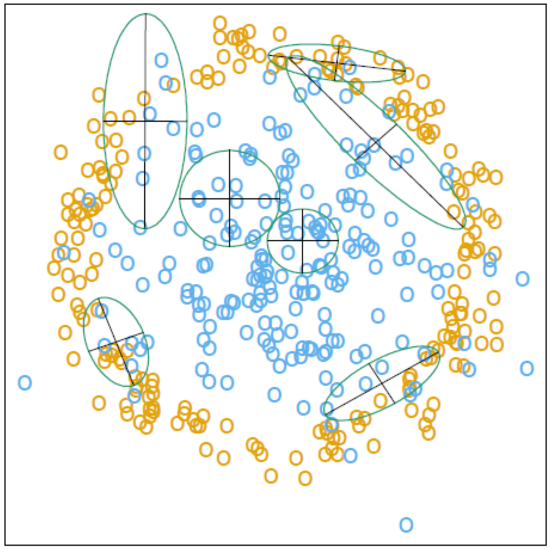


# *Adaptive Nearest Neighbors (cont.)*

- The same idea as in dimension reduction
- Use knn to find some neighboring points first
- Then recompute the distance measurements
- $W^{-1/2}W^{-1/2}$ "spheres" the data (within class var)
- Lengthen the dimension with small eigen values in B* (between class var)

$$D(x, x_0) = (x - x_0)^T \mathbf{\Sigma} (x - x_0),$$

$$\begin{aligned}\mathbf{\Sigma} &= \mathbf{W}^{-1/2}[\mathbf{W}^{-1/2}\mathbf{B}\mathbf{W}^{-1/2} + \epsilon\mathbf{I}]\mathbf{W}^{-1/2} \\ &= \mathbf{W}^{-1/2}[\mathbf{B}^* + \epsilon\mathbf{I}]\mathbf{W}^{-1/2}.\end{aligned}$$



# Local Weighted Regression

- knn is a local approximation method without *explicitly* building the local decision surface
- Approximation by explicitly building such a surface is possible
- Difference from parametric techniques
  - Local samples are used
  - Weighted by distance
  - Multiple local approximations (instead of one global one)

# *Example*

- Assume that locally the decision surface is a linear function of the *n* attributes $a_n$

$$\hat{f}(\mathbf{x}) = w_o + w_1 a_1 + w_2 a_2 + \cdots + w_n a_n$$

$$E = \frac{1}{2} \sum_{\mathbf{x} \in knn\,\mathbf{x}_q} (f(\mathbf{x}) - \hat{f}(\mathbf{x})) K(d(\mathbf{x}_q, \mathbf{x}))^2$$

- K is a nonincreasing function, e.g., $k(d(\mathbf{x}_q, \mathbf{x})) = e^{-\frac{d^2(\mathbf{x}_q, \mathbf{x})}{2\sigma_q^2}}$

# *Learning Rule*

- Starting from an arbitrary set of weights
- If *f* (true) and *f*^hat (estimated) are the same, no change
- Otherwise, change $w_i$

$$\hat{f}(\mathbf{x}) = w_o + w_1 a_1 + w_2 a_2 + \cdots + w_n a_n$$

$$\Delta w_i = \eta \sum_{\mathbf{x} \in knn\, \mathbf{x}_q} (f(\mathbf{x}) - \hat{f}(\mathbf{x})) K(d(\mathbf{x}_q, \mathbf{x})) a_i \quad \eta : \text{learning rate}$$

# *Learning Rule (cont.)*

- We will see later that this rule is the perceptron learning rule used in perceptron learning in ANN
- The locally weighted approximation is very similar to the radial basis function learning in ANN