CS 559: Machine Learning Fundamentals and Applications 9th Set of Notes

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

Overview

- Logistic Regression
  - Notes by T. Mitchell
  - Barber Ch. 17
  - HTF Ch. 4

- Linear Discriminant Functions (Slides based on Olga Veksler's)
  - Optimization with gradient descent
  - Perceptron Criterion Function
    - Batch perceptron rule
    - Single sample perceptron rule
  - Minimum Squared Error (MSE) rule

Overview (cont.)

- Support Vector Machines (SVM)
  - Introduction
  - Linear Discriminant
    - Linearly Separable Case
    - Linearly Non Separable Case
  - Kernel Trick
    - Non Linear Discriminant
  - Multi-class SVMs

- See HTF Ch. 12

Logistic Regression

- Idea: generative models compute $P(Y|X)$ by learning $P(Y)$ and $P(X|Y)$

- Why not learn $P(Y|X)$ directly?

Logistic Regression

- Consider learning f: X → Y, where
  - X is a vector of real-valued features, < X_1 … X_n >
  - Y is boolean
  - assume all X_i are conditionally independent given Y
  - model $P(X_i \mid Y = y_k)$ as Gaussian $N(\mu_{ik}, \sigma_i^2)$
  - model P(Y) as Bernoulli (π)
    - Y is 1 with probability π

Derivation of P(Y|X)

$$P(Y=1|X) = \frac{P(Y=1)P(X|Y=1)}{P(Y=1)P(X|Y=1) + P(Y=0)P(X|Y=0)}$$

$$= \frac{1}{1 + \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)}}$$

$$= \frac{1}{1 + \exp(\ln \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)})}$$

$$= \frac{1}{1 + \exp(\ (\ln \frac{1-\pi}{\pi}) + \boxed{\sum_i \ln \frac{P(X_i|Y=0)}{P(X_i|Y=1)}})}$$

$$P(x \mid y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} \quad e^{\frac{-(x-\mu_{ik})^2}{2\sigma_{ik}^2}}$$

$$\boxed{\sum_i \left(\frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2} X_i + \frac{\mu_{i1}^2 - \mu_{i0}^2)}{2\sigma_i^2} \right)}$$

$$P(Y=1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^{n} w_i X_i)}$$

Very Convenient

$$P(Y=1|X=<X_1,...X_n>) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

implies

$$P(Y=0|X=<X_1,...X_n>) = \frac{exp(w_0+\sum_i w_i X_i)}{1+exp(w_0+\sum_i w_i X_i)}$$

implies

$$\frac{P(Y=0|X)}{P(Y=1|X)} = exp(w_0+\sum_i w_i X_i)$$

linear classification rule!

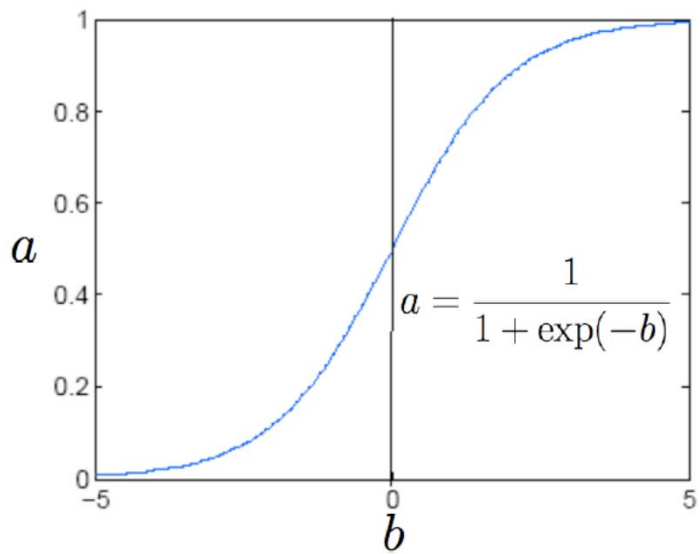
implies

$$\ln \frac{P(Y=0|X)}{P(Y=1|X)} = w_0+\sum_i w_i X_i$$

Very Convenient

- Posteriors sum to 1 and remain in [0, 1]
- Logit: $l = logit(p) = \log\left(\frac{p}{1-p}\right)$=α+βx
  - l is linear in x
- Probability: $p = \frac{e^l}{1+e^l}$

Logistic Function



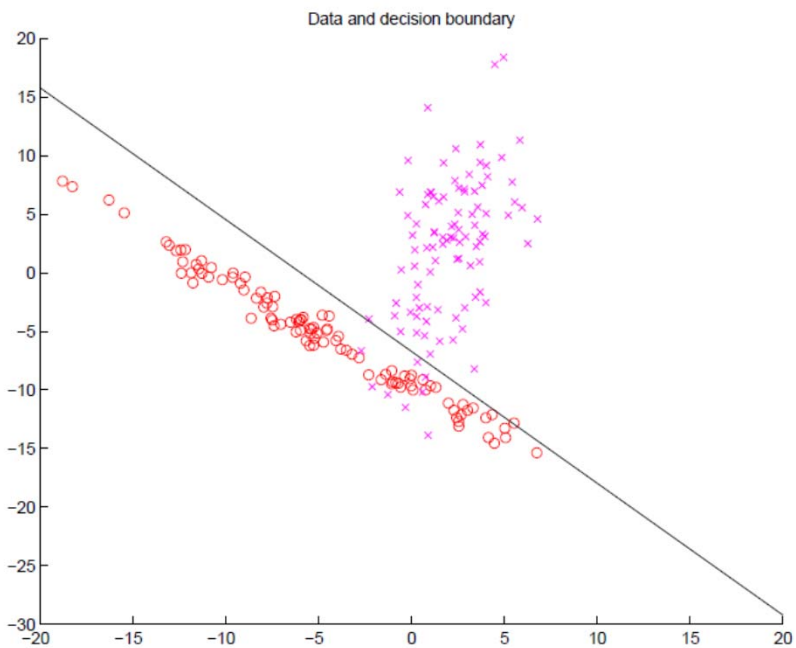
$$P(Y=1|X) = \frac{1}{1+\exp(w_0 + \sum_{i=1}^{n} w_i X_i)}$$

- $p = 0, \quad l = \log\left(\frac{p}{1-p}\right) = -\infty$
- $p = \frac{1}{2}, \quad l = \log\left(\frac{p}{1-p}\right) = 0$
- $p = 1, \quad l = \log\left(\frac{p}{1-p}\right) = +\infty$

Decision Boundary

- How to make decisions given

$$P(Y=1|X=<X_1,...X_n>) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$



Logistic Regression More Generally

- Logistic regression when Y is not boolean (but still discrete)
  - $y \in \{y_1 \ldots y_R\}$: learn R-1 sets of weights
  - for k<R $$P(Y = y_k|X) = \frac{\exp(w_{k0} + \sum_{i=1}^{n} w_{ki} X_i)}{1 + \sum_{j=1}^{R-1} \exp(w_{j0} + \sum_{i=1}^{n} w_{ji} X_i)}$$
  - for k=R $$P(Y = y_R|X) = \frac{1}{1 + \sum_{j=1}^{R-1} \exp(w_{j0} + \sum_{i=1}^{n} w_{ji} X_i)}$$

Training Logistic Regression: MCLE

- We have L training examples:

$$\{\langle X^1, Y^1\rangle, \ldots \langle X^L, Y^L\rangle\}$$

- Maximum likelihood estimate for parameters W

$$W_{MLE} = \arg\max_W P(< X^1, Y^1 > \ldots < X^L, Y^L > | W)$$
$$= \arg\max_W \prod_l P(< X^l, Y^l > | W)$$

- Maximum <u>conditional</u> likelihood estimate

Training Logistic Regression: MCLE

- Choose parameters <w_0 … w_n> to maximize conditional likelihood of training data

$$P(Y=0|X,W) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

$$P(Y=1|X,W) = \frac{exp(w_0+\sum_i w_i X_i)}{1+exp(w_0+\sum_i w_i X_i)}$$

- Training data D=$\{\langle X^1, Y^1\rangle, \ldots \langle X^L, Y^L\rangle\}$
- Data likelihood = $\prod_l P(X^l, Y^l|W)$
- Data conditional likelihood = $\prod_l P(Y^l|X^l, W)$

$$W_{MCLE} = \arg\max_W \prod_l P(Y^l|W, X^l)$$

Conditional Log Likelihood

$$l(W) \equiv \ln \prod_l P(Y^l|X^l, W) = \sum_l \ln P(Y^l|X^l, W)$$

$$P(Y=0|X,W) = \frac{1}{1+exp(w_0 + \sum_i w_i X_i)}$$

$$P(Y=1|X,W) = \frac{exp(w_0 + \sum_i w_i X_i)}{1+exp(w_0 + \sum_i w_i X_i)}$$

$$\begin{aligned} l(W) &= \sum_l Y^l \ln P(Y^l = 1|X^l, W) + (1 - Y^l) \ln P(Y^l = 0|X^l, W) \\ &= \sum_l Y^l \ln \frac{P(Y^l = 1|X^l, W)}{P(Y^l = 0|X^l, W)} + \ln P(Y^l = 0|X^l, W) \\ &= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + exp(w_0 + \sum_i^n w_i X_i^l)) \end{aligned}$$

Maximizing Conditional Log Likelihood

$$P(Y=0|X,W) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

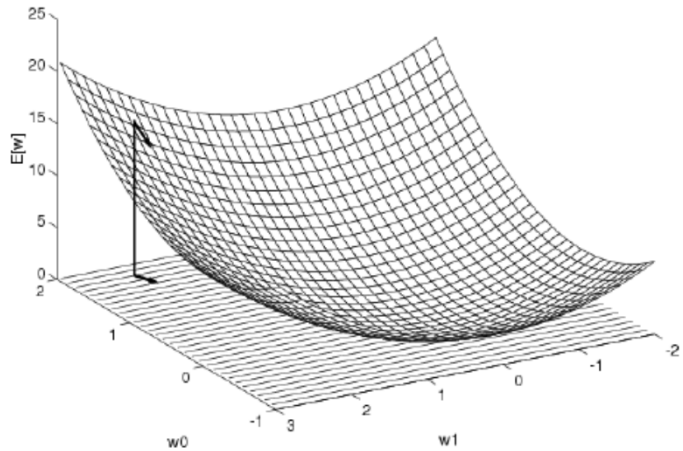
$$P(Y=1|X,W) = \frac{exp(w_0+\sum_i w_i X_i)}{1+exp(w_0+\sum_i w_i X_i)}$$

$$\begin{aligned} l(W) &\equiv \ln \prod_l P(Y^l|X^l,W) \\ &= \sum_l Y^l(w_0+\sum_i^n w_i X_i^l) - \ln(1+exp(w_0+\sum_i^n w_i X_i^l)) \end{aligned}$$

Good news: l(W) is concave function of W

Bad news: no closed-form solution to maximize l(W)

Gradient Descent



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

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

Maximize Conditional Log Likelihood: Gradient Ascent

$$
\begin{aligned}
l(W) &\equiv \ln \prod_l P(Y^l | X^l, W) \\
&= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + exp(w_0 + \sum_i^n w_i X_i^l))
\end{aligned}
$$

$$
\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))
$$

Maximize Conditional Log Likelihood: Gradient Ascent

$$
\begin{aligned}
l(W) &\equiv \ln \prod_l P(Y^l | X^l, W) \\
&= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + exp(w_0 + \sum_i^n w_i X_i^l))
\end{aligned}
$$

$$
\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))
$$

Gradient ascent algorithm: iterate until change < ε

For all i, repeat

$$
w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))
$$

Logistic Regression: Summary

- Consider learning f: X → Y, where
  - X is a vector of real-valued features, < X_1 … X_n >
  - Y is boolean
  - assume all X_i are conditionally independent given Y
  - model $P(X_i \mid Y = y_k)$ as Gaussian $N(\mu_{ik}, \sigma_i^2)$
  - model P(Y) as Bernoulli (π)
- Then P(Y|X) is of this form and we can directly estimate W

$$P(Y=1|X=<X_1,...X_n>) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

Linear Discriminant Functions

Augmented Feature Vector

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

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

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

old problem

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

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

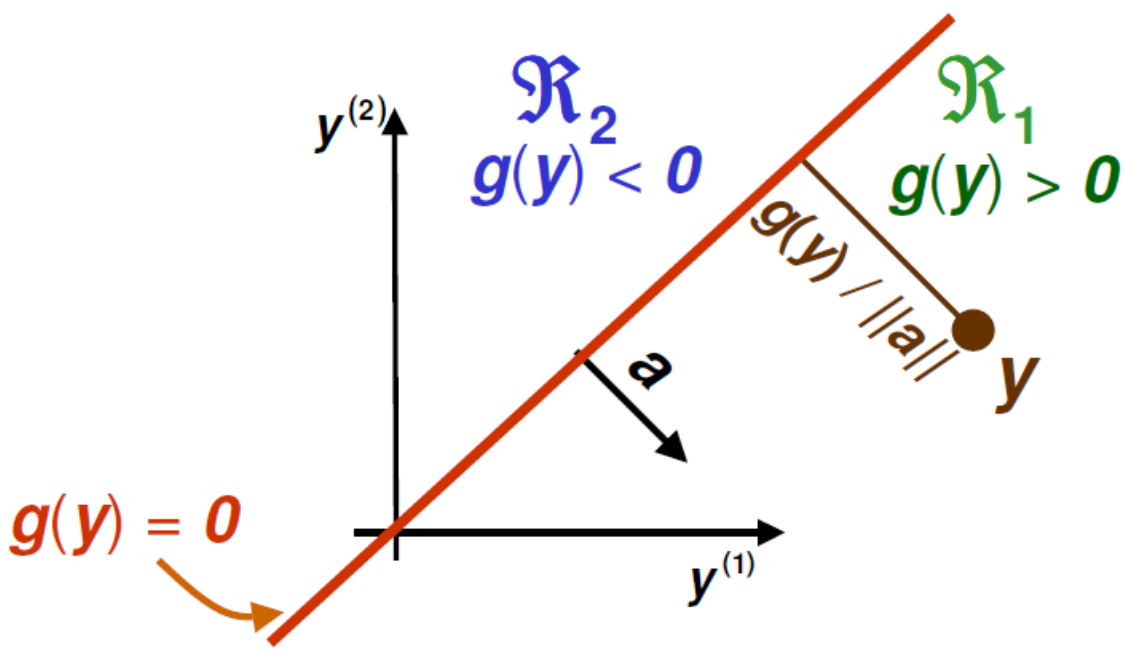
new problem

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

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

- Feature augmentation is done for simpler notation
- From now on, always assume that we have augmented feature vectors
  - Given samples $\mathbf{x}_1,\ldots, \mathbf{x}_n$ convert them to augmented samples $\mathbf{y}_1,\ldots, \mathbf{y}_n$ by adding a new dimension of value 1

$$\mathbf{y}_i = \begin{bmatrix} 1 \\ \mathbf{x}_i \end{bmatrix}$$



Training Error

- For the rest of this part, assume we have 2 classes
  - Samples: $y_1,\ldots, y_n$, some in class 1, some in class 2
- Use samples to determine weights **a** in the discriminant function $g(y) = a^t y$
- What should the criterion for determining **a** be?
- For now, suppose we want to minimize the training error (the number of misclassified samples $y_1,\ldots, y_n$)

- Recall that: $g(y_i)>0 \Rightarrow y_i$ ***classified as*** c_1
  $g(y_i)<0 \Rightarrow y_i$ ***classified as*** c_2

- Thus training error is 0 if $\begin{cases} g(y_i) > 0 & \forall y_i \in c_1 \\ g(y_i) < 0 & \forall y_i \in c_2 \end{cases}$

"Normalization"

- Thus training error is 0 if: $\begin{cases} \boldsymbol{a}^t \boldsymbol{y}_i > 0 & \forall \boldsymbol{y}_i \in \boldsymbol{c}_1 \\ \boldsymbol{a}^t \boldsymbol{y}_i < 0 & \forall \boldsymbol{y}_i \in \boldsymbol{c}_2 \end{cases}$
- Equivalently, training error is 0 if: $\begin{cases} \boldsymbol{a}^t \boldsymbol{y}_i > 0 & \forall \boldsymbol{y}_i \in \boldsymbol{c}_1 \\ \boldsymbol{a}^t (-\boldsymbol{y}_i) > 0 & \forall \boldsymbol{y}_i \in \boldsymbol{c}_2 \end{cases}$
- This suggests "normalization" (a.k.a. reflection):
  1. Replace all examples from class 2 by:

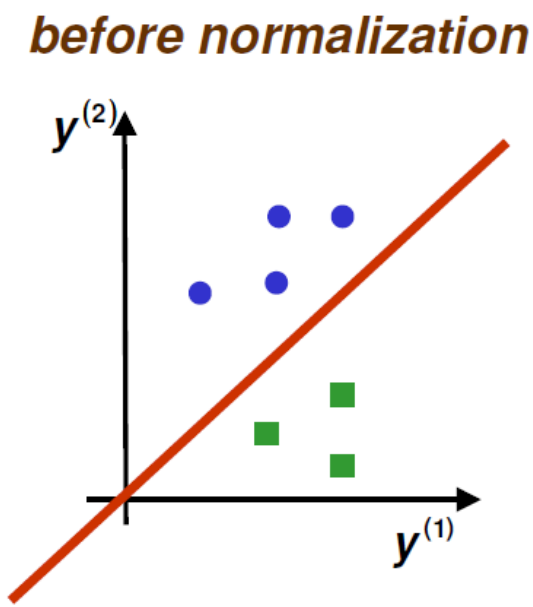
     $$\boldsymbol{y}_i \rightarrow -\boldsymbol{y}_i \qquad \forall \boldsymbol{y}_i \in \boldsymbol{c}_2$$

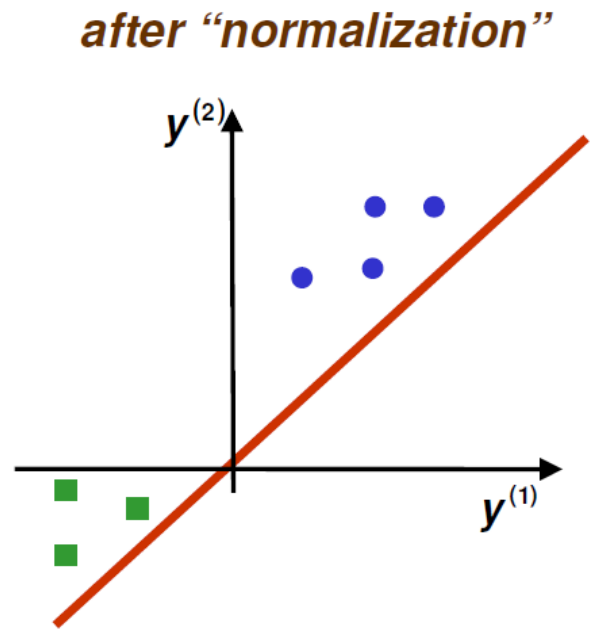
  2. Seek weight vector ***a*** such that

     $$\boldsymbol{a}^t \boldsymbol{y}_i > 0 \qquad \forall \boldsymbol{y}_i$$

  – If such **a** exists, it is called a separating or solution vector
  – Original samples $\mathbf{x}_1, \ldots, \mathbf{x}_n$ can indeed be separated by a line

Normalization



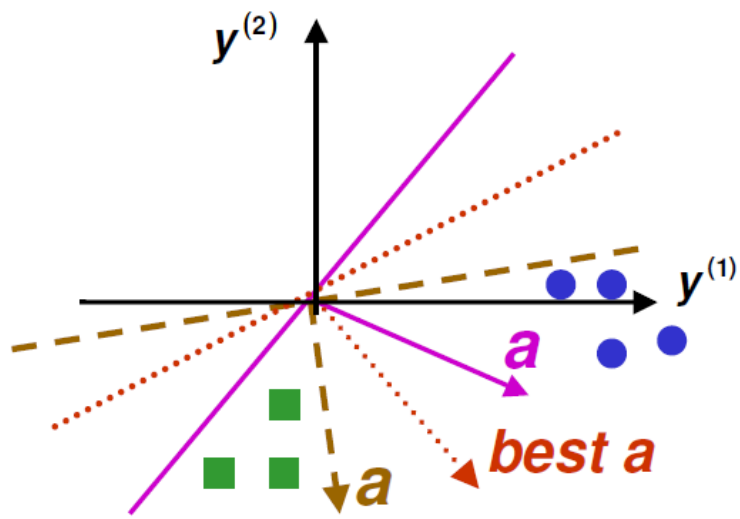


- Seek a hyperplane that separates patterns from different categories
- Seek hyperplane that puts *normalized* patterns on the same(positive) side

Solution Region

- Find weight vector ***a*** such that for all samples:

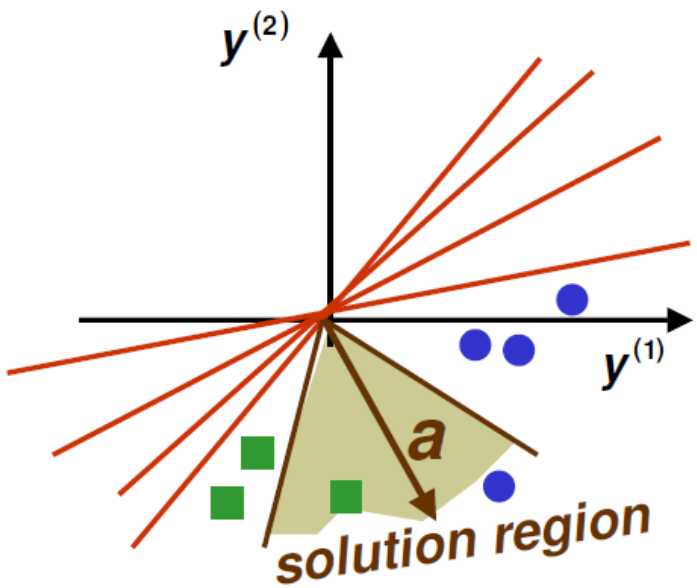
$$\boldsymbol{a}^t \boldsymbol{y}_i = \sum_{k=0}^{d} \boldsymbol{a}_k \boldsymbol{y}_i^{(k)} > 0$$



- In general, there can be many solutions

Solution Region

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



Optimization

- Need to minimize a function of many variables

$$J(x) = J(x_1, \ldots, x_d)$$

- We know how to minimize **J(x)**
  - Take partial derivatives and set them to zero

$$\begin{bmatrix} \frac{\partial}{\partial x_1} J(x) \\ \vdots \\ \frac{\partial}{\partial x_d} J(x) \end{bmatrix} = \nabla J(x) = 0$$

gradient

Optimization

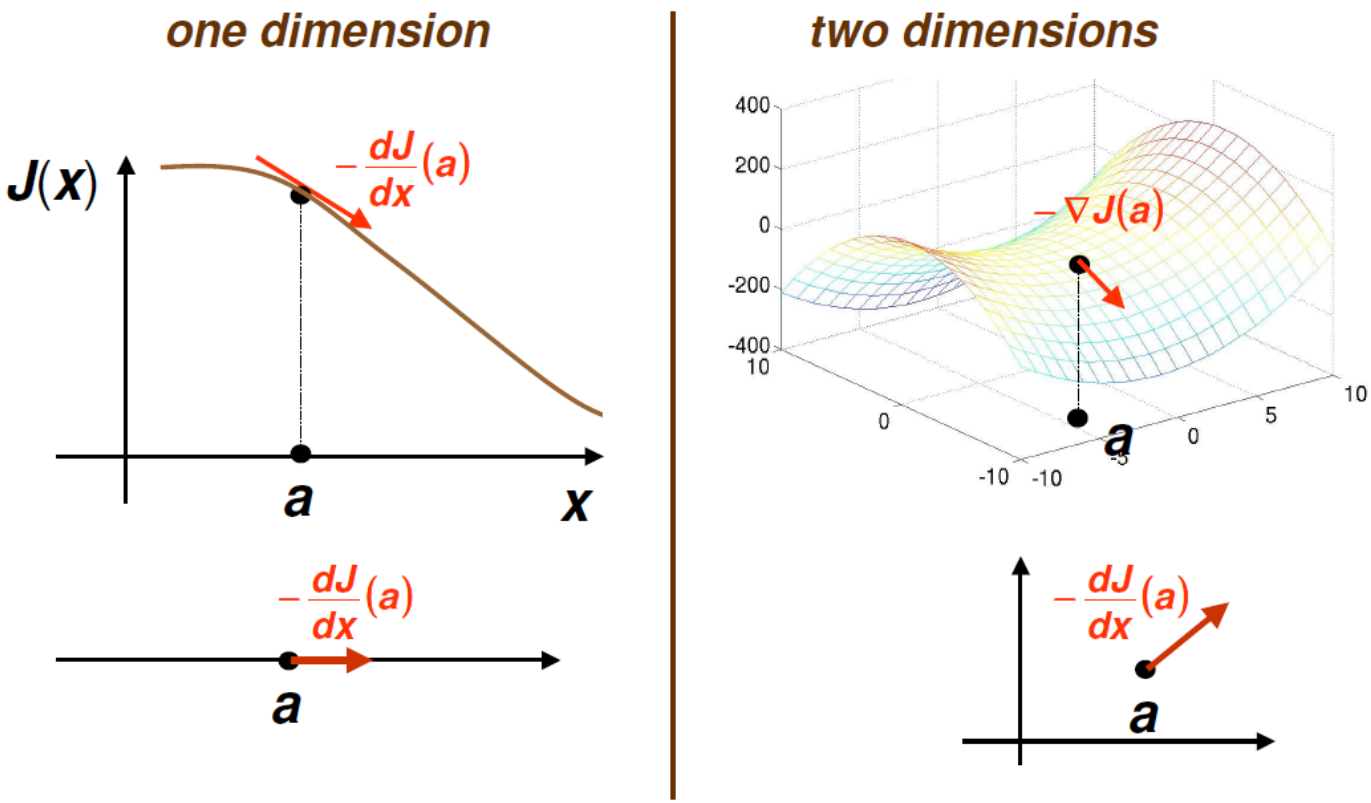
- However solving analytically is not always easy
  - For example:

$$\begin{cases} \sin\left(x_1^2 + x_2^3\right) + e^{x_4^2} = 0 \\ \cos\left(x_1^2 + x_2^3\right) + \log\left(x_5^3\right)^{x_4^2} = 0 \end{cases}$$

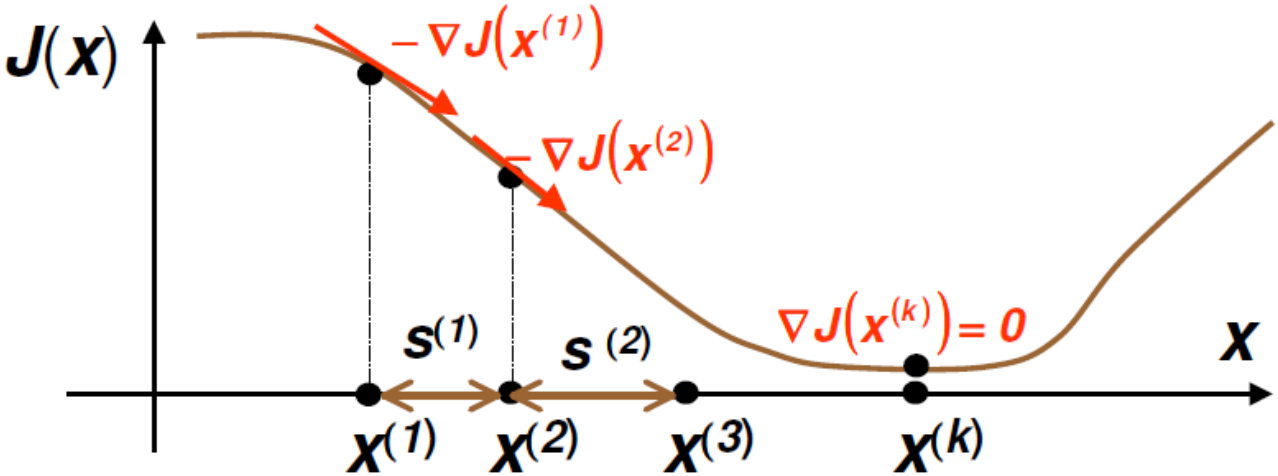
- Sometimes it is not even possible to write down an analytical expression for the derivative (example later today)

Gradient Descent

- Gradient $\nabla J(x)$ points in direction of steepest increase of J(x), and $-\nabla J(x)$ in direction of steepest decrease



Gradient Descent



Gradient Descent for minimizing any function ***J(x)***

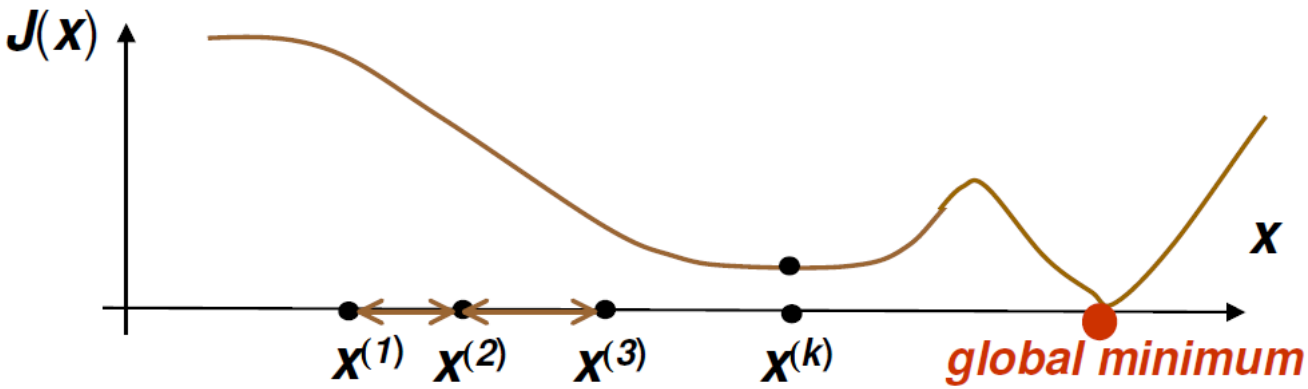
- Set k = 1 and $x^{(1)}$ to some initial guess for the weight vector
- While $\eta^{(k)} \left| \nabla J(x^{(k)}) \right| > \varepsilon$
  - Choose learning rate $\eta^{(k)}$

$$x^{(k+1)} = x^{(k)} - \eta^{(k)} \nabla J(x) \qquad \text{(update rule)}$$

$$k = k + 1$$

Gradient Descent

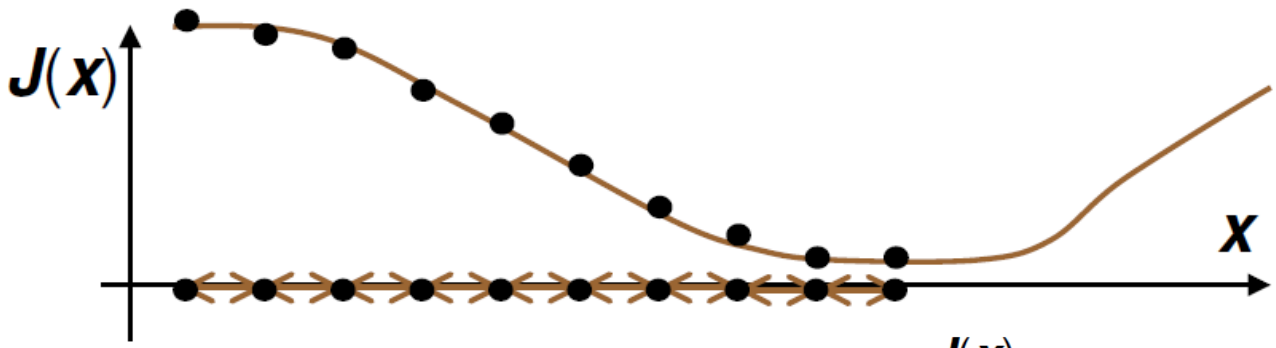
- Gradient decent is guaranteed to only find local minima



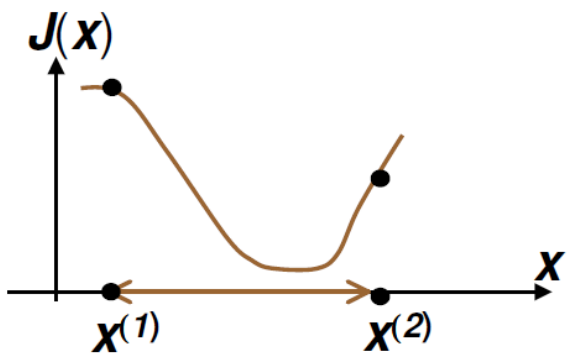
- Nevertheless gradient descent is very popular because it is simple and applicable to any function

Gradient Descent

- Main issue: how to set parameter η (learning rate)
  - If η is too small, too many iterations



  - If η is too large may overshoot the minimum and possibly never find it



LDF Criterion Function

- Find weight vector **a** such that for all samples $y_1,\ldots, y_n$

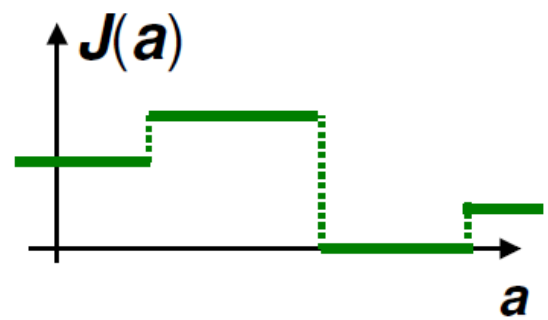
$$a^t y_i = \sum_{k=0}^{d} a_k y_i^{(k)} > 0$$

- Need criterion function **J(a)** which is minimized when **a** is a solution vector
- Let Y_M be the set of examples misclassified by **a**

$$Y_M(a) = \{ y_i \text{ s.t. } a^t y_i < 0 \}$$

- First natural choice: number of misclassified examples

$$J(a) = |Y_M(a)|$$

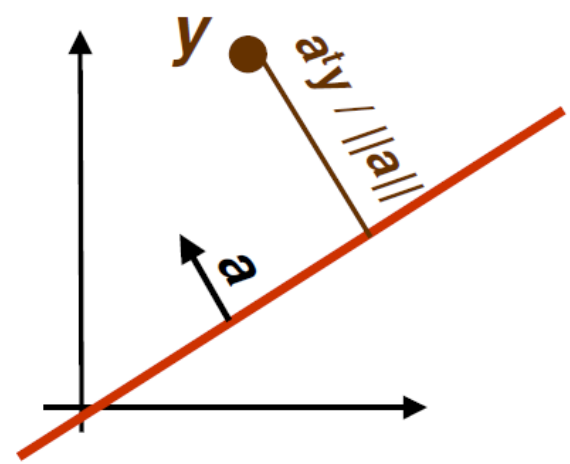


- Piecewise constant, gradient descent is useless

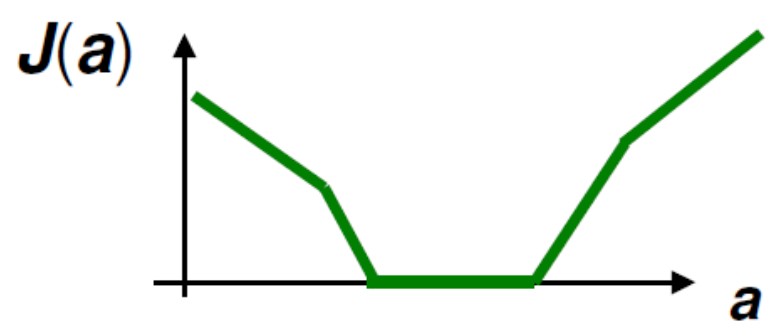
Perceptron

Perceptron Criterion Function

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



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



Perceptron Batch Rule

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

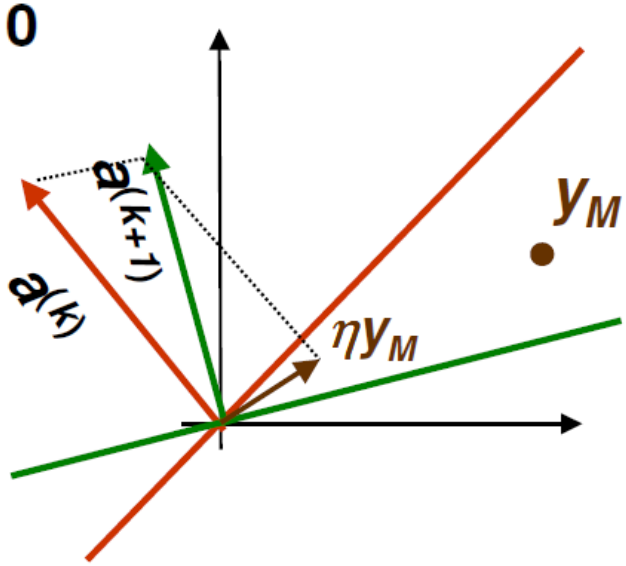
- Gradient of $J_p(a)$ is: $\nabla J_p(a) = \sum_{y \in Y_M} (-y)$
  - Y_M are samples misclassified by $a^{(k)}$
  - It is not possible to solve $\nabla J_p(a) = 0$ analytically because of Y_M
- Update rule for gradient descent: $x^{(k+1)} = x^{(k)} - \eta^{(k)} \nabla J(x)$
- Thus the *gradient decent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

$$a^{(k+1)} = a^{(k)} + \eta^{(k)} \sum_{y \in Y_M} y$$

- It is called batch rule because it is based on all misclassified examples

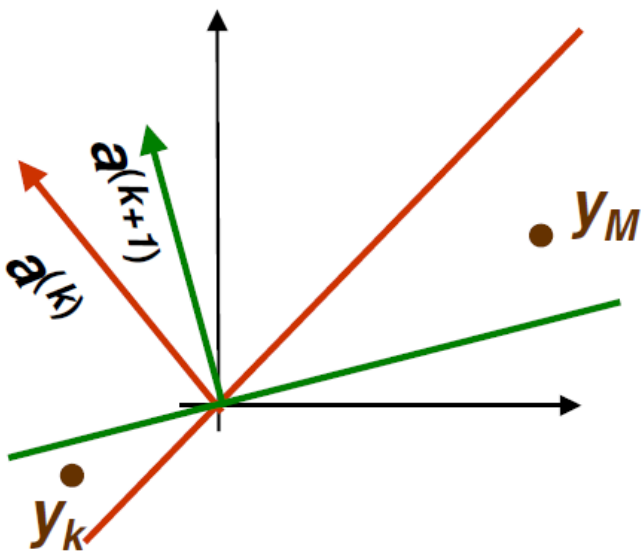
Perceptron Single Sample Rule

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

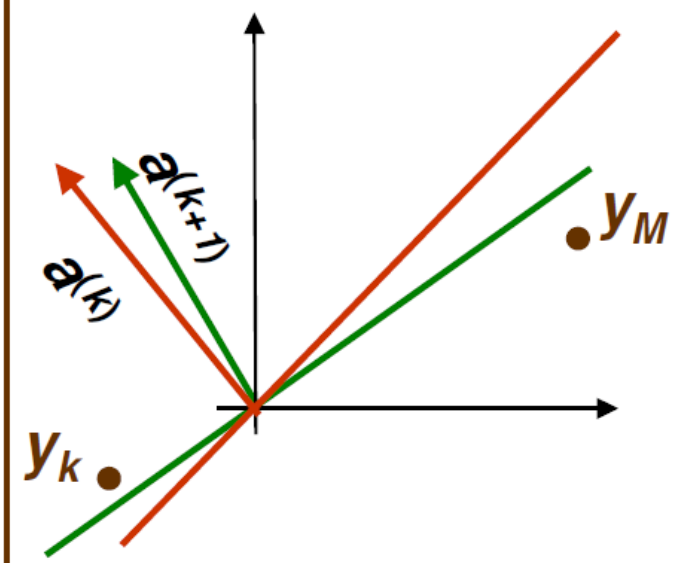


Perceptron Single Sample Rule

$$\boldsymbol{a}^{(k+1)} = \boldsymbol{a}^{(k)} + \eta^{(k)} \boldsymbol{y}_M$$



η is too large, previously correctly classified sample $\boldsymbol{y}_k$ is now misclassified



η is too small, $\boldsymbol{y}_M$ is still misclassified

Perceptron Example

	features				grade
name	***good attendance?***	***tall?***	***sleeps in class?***	***chews gum?***	
Jane	*yes (1)*	*yes (1)*	*no (-1)*	*no (-1)*	*A*
Steve	*yes (1)*	*yes (1)*	*yes (1)*	*yes (1)*	*F*
Mary	*no (-1)*	*no (-1)*	*no (-1)*	*yes (1)*	*F*
Peter	*yes (1)*	*no (-1)*	*no (-1)*	*yes (1)*	*A*

- Class 1: students who get A
- Class 2: students who get F

	features					grade
name	***extra***	***good attendance?***	***tall?***	***sleeps in class?***	***chews gum?***	
Jane	*1*	*yes (1)*	*yes (1)*	*no (-1)*	*no (-1)*	*A*
Steve	*1*	*yes (1)*	*yes (1)*	*yes (1)*	*yes (1)*	*F*
Mary	*1*	*no (-1)*	*no (-1)*	*no (-1)*	*yes (1)*	*F*
Peter	*1*	*yes (1)*	*no (-1)*	*no (-1)*	*yes (1)*	*A*

- Augment samples by adding an extra feature (dimension) equal to 1

	features					grade
name	***extra***	***good attendance?***	***tall?***	***sleeps in class?***	***chews gum?***	
Jane	*1*	*yes (1)*	*yes (1)*	*no (-1)*	*no (-1)*	*A*
Steve	*-1*	*yes (-1)*	*yes (-1)*	*yes (-1)*	*yes (-1)*	*F*
Mary	*-1*	*no (1)*	*no (1)*	*no (1)*	*yes (-1)*	*F*
Peter	*1*	*yes (1)*	*no (-1)*	*no (-1)*	*yes (1)*	*A*

- Normalize:
  - Replace all examples from class 2 by their negative values $\mathbf{y}_i \rightarrow -\mathbf{y}_i \quad \forall \mathbf{y}_i \in \mathbf{c}_2$
  - Seek $\boldsymbol{a}$ such that: $\mathbf{a}^t \mathbf{y}_i > 0 \quad \forall \mathbf{y}_i$

	features					grade
name	***extra***	***good attendance?***	***tall?***	***sleeps in class?***	***chews gum?***	
Jane	*1*	*yes (1)*	*yes (1)*	*no (-1)*	*no (-1)*	*A*
Steve	*-1*	*yes (-1)*	*yes (-1)*	*yes (-1)*	*yes (-1)*	*F*
Mary	*-1*	*no (1)*	*no (1)*	*no (1)*	*yes (-1)*	*F*
Peter	*1*	*yes (1)*	*no (-1)*	*no (-1)*	*yes (1)*	*A*

- Single Sample Rule
  - Sample is misclassified if $\boldsymbol{a}^t \boldsymbol{y}_i = \sum_{k=0}^{4} a_k y_i^{(k)} < 0$
  - Gradient descent single sample rule: $\boldsymbol{a}^{(k+1)} = \boldsymbol{a}^{(k)} + \eta^{(k)} \sum_{\boldsymbol{y} \in Y_M} \boldsymbol{y}$
  - Set η fixed learning rate to $\eta^{(k)} = 1$: $\boxed{\boldsymbol{a}^{(k+1)} = \boldsymbol{a}^{(k)} + \boldsymbol{y}_M}$

- Set equal initial weights
  $a^{(1)} = [0.25, 0.25, 0.25, 0.25, 0.25]$
- Visit all samples sequentially, modifying the weights after each misclassified example

name	$a^t y$	*misclassified?*
Jane	*0.25*1+0.25*1+0.25*1+0.25*(-1)+0.25*(-1) >0*	*no*
Steve	*0.25*(-1)+0.25*(-1)+0.25*(-1)+0.25*(-1)+0.25*(-1)<0*	*yes*

- New weights

$$\begin{aligned} a^{(2)} = a^{(1)} + y_M &= [0.25 \;\; 0.25 \;\; 0.25 \;\; 0.25 \;\; 0.25] + \\ &+ [-1 \;\; -1 \;\; -1 \;\; -1 \;\; -1] = \\ &= [-0.75 \;\; -0.75 \;\; -0.75 \;\; -0.75 \;\; -0.75] \end{aligned}$$

$$a^{(2)} = [-0.75 \;\; -0.75 \;\; -0.75 \;\; -0.75 \;\; -0.75]$$

name	$a^t y$	misclassified?
Mary	-0.75*(-1)-0.75*1 -0.75 *1 -0.75 *1 -0.75*(-1) <0	yes

- New weights

$$a^{(3)} = a^{(2)} + y_M = [-0.75 \;\; -0.75 \;\; -0.75 \;\; -0.75 \;\; -0.75] +$$
$$+ [-1 \quad 1 \quad 1 \quad 1 \quad -1] =$$
$$= [-1.75 \quad 0.25 \quad 0.25 \quad 0.25 \quad -1.75]$$

$$a^{(3)} = [-1.75 \quad 0.25 \quad 0.25 \quad 0.25 \quad -1.75]$$

name	$a^t y$	*misclassified?*
Peter	*-1.75 *1 +0.25* 1+0.25* (-1) +0.25 *(-1)-1.75*1 <0*	*yes*

- New weights

$$a^{(4)} = a^{(3)} + y_M = [-1.75 \quad 0.25 \quad 0.25 \quad 0.25 \quad -1.75] +$$
$$+ [1 \quad 1 \quad -1 \quad -1 \quad 1] =$$
$$= [-0.75 \quad 1.25 \quad -0.75 \quad -0.75 \quad -0.75]$$

$$a^{(4)} = [-0.75 \quad 1.25 \quad -0.75 \quad -0.75 \quad -0.75]$$

name	$a^t y$	*misclassified?*
Jane	*-0.75 *1 +1.25*1 -0.75*1 -0.75 *(-1) -0.75 *(-1)+0*	*no*
Steve	*-0.75*(-1)+1.25*(-1) -0.75*(-1) -0.75*(-1)-0.75*(-1)>0*	*no*
Mary	*-0.75 *(-1)+1.25*1-0.75*1 -0.75 *1 –0.75*(-1) >0*	*no*
Peter	*-0.75 *1+ 1.25*1-0.75* (-1)-0.75* (-1) -0.75 *1 >0*	*no*

- Thus the discriminant function is:

$$g(y) = -0.75 * y^{(0)} + 1.25 * y^{(1)} - 0.75 * y^{(2)} - 0.75 * y^{(3)} - 0.75 * y^{(4)}$$

- Converting back to the original features ***x:***

$$g(x) = 1.25 * x^{(1)} - 0.75 * x^{(2)} - 0.75 * x^{(3)} - 0.75 * x^{(4)} - 0.75$$

- Converting back to the original features ***x:***

$$1.25 * x^{(1)} - 0.75 * x^{(2)} - 0.75 * x^{(3)} - 0.75 * x^{(4)} > 0.75 \Rightarrow grade\ A$$
$$1.25 * x^{(1)} - 0.75 * x^{(2)} - 0.75 * x^{(3)} - 0.75 * x^{(4)} < 0.75 \Rightarrow grade\ F$$

good attendance — tall — sleeps in class — chews gum

- This is just one possible solution vector
- If we started with weights

$$a^{(1)}=[0,0.5, 0.5, 0, 0\ ],$$

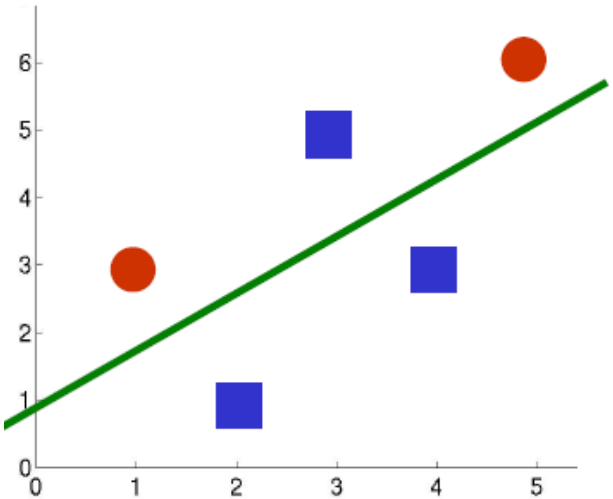
- The solution would be [-1,1.5, -0.5, -1, -1]

$$1.5 * x^{(1)} - 0.5 * x^{(2)} - x^{(3)} - x^{(4)} > 1 \Rightarrow grade\ A$$
$$1.5 * x^{(1)} - 0.5 * x^{(2)} - x^{(3)} - x^{(4)} < 1 \Rightarrow grade\ F$$

- In this solution, being tall is the least important feature

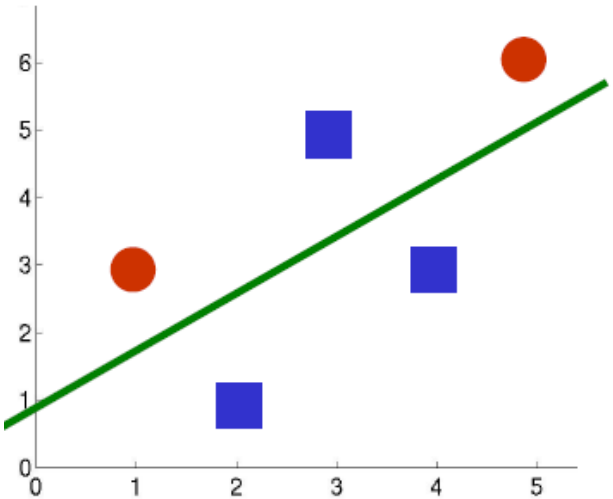
LDF: Non-separable Example

- Suppose we have 2 features and the samples are:
  - Class 1: [2,1], [4,3], [3,5]
  - Class 2: [1,3] and [5,6]
- These samples are not separable by a line
- Still would like to get approximate separation by a line
  - A good choice is shown in green
  - Some samples may be “noisy”, and we could accept them being misclassified



LDF: Non-separable Example

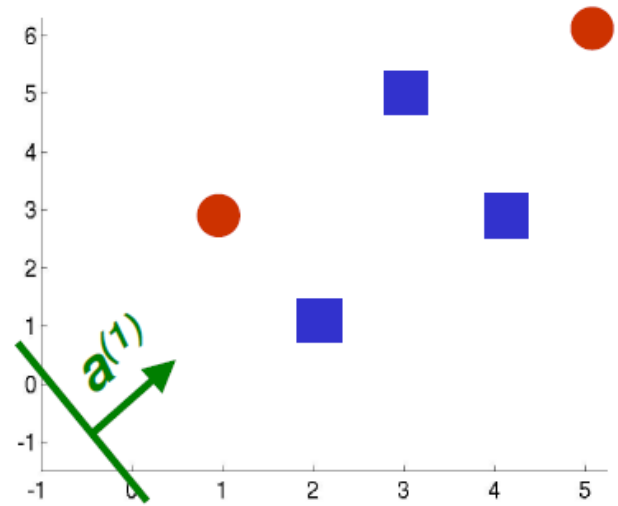
- Obtain y_1, y_2, y_3, y_4 by adding extra feature and "normalizing"



$$y_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad y_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad y_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$

LDF: Non-separable Example

- Apply Perceptron single sample algorithm
- Initial equal weights $a^{(1)} = [1\ 1\ 1]$
  - Line equation $x^{(1)}+x^{(2)}+1=0$
- Fixed learning rate $\eta = 1$



$$a^{(k+1)} = a^{(k)} + y_M$$

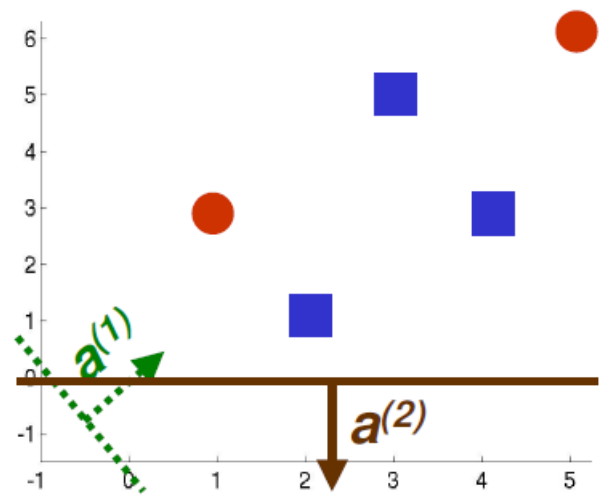
$$y_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad y_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad y_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$

- $y^t_1 a^{(1)} = [1\ 1\ 1]*[1\ 2\ 1]^t > 0$ ✓
- $y^t_2 a^{(1)} = [1\ 1\ 1]*[1\ 4\ 3]^t > 0$ ✓
- $y^t_3 a^{(1)} = [1\ 1\ 1]*[1\ 3\ 5]^t > 0$ ✓

LDF: Non-separable Example

$$a^{(1)} = [1\ 1\ 1] \qquad a^{(k+1)} = a^{(k)} + y_M$$

$$y_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad y_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad y_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$



- $y^t_4 a^{(1)} = [1\ 1\ 1]*[-1\ -1\ -3]^t = -5 < 0$

$$a^{(2)} = a^{(1)} + y_M = [1\ 1\ 1] + [-1\ -1\ -3] = [0\ 0\ -2]$$

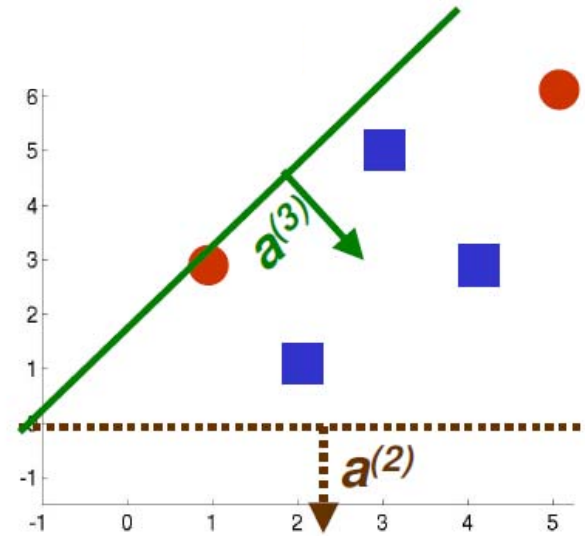
- $y^t_5 a^{(2)} = [0\ 0\ -2]*[-1\ -5\ -6]^t = 12 > 0$ ✔
- $y^t_1 a^{(2)} = [0\ 0\ -2]*[1\ 2\ 1]^t < 0$

$$a^{(3)} = a^{(2)} + y_M = [0\ 0\ -2] + [1\ 2\ 1] = [1\ 2\ -1]$$

LDF: Non-separable Example

$$a^{(3)} = [1\ 2\ -1] \qquad a^{(k+1)} = a^{(k)} + y_M$$

$$y_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad y_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad y_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$



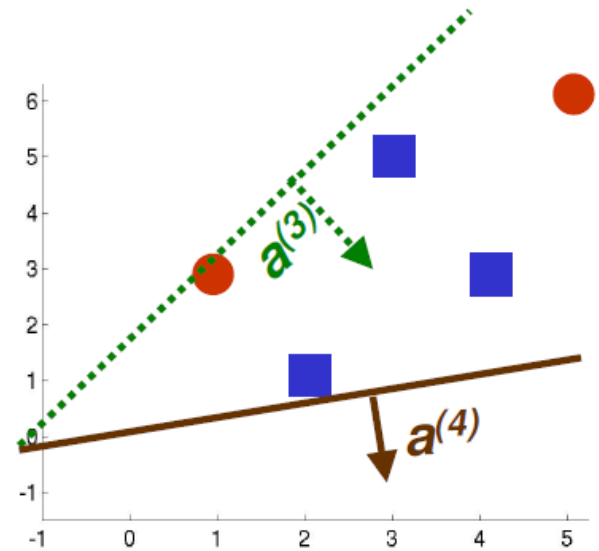
- $y^t_2\, a^{(3)} = [1\ 4\ 3]*[1\ 2\ -1]^t = 6 > 0$ ✓
- $y^t_3\, a^{(3)} = [1\ 3\ 5]*[1\ 2\ -1]^t > 0$ ✓
- $y^t_4\, a^{(3)} = [-1\ -1\ -3]*[1\ 2\ -1]^t = 0$

$$a^{(4)} = a^{(3)} + y_M = [1\ 2\ -1] + [-1\ -1\ -3] = [0\ 1\ -4]$$

LDF: Non-separable Example

$$a^{(4)} = [0\ 1\ -4] \qquad a^{(k+1)} = a^{(k)} + y_M$$

$$y_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad y_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad y_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$



- $y_5^t a^{(4)}$=[-1 -5 -6]*[0 1 -4]=19>0
- $y_1^t a^{(4)}$=[1 2 1]*[0 1 -4]=-2<0
- ....

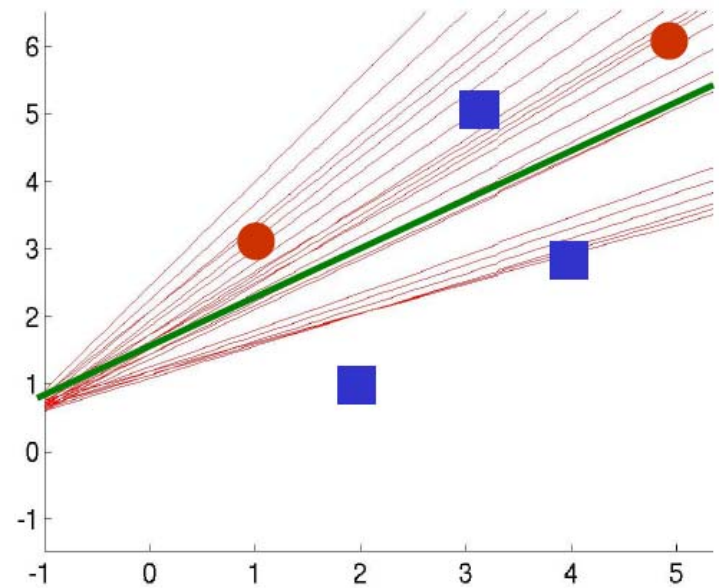
LDF: Non-separable Example

- We can continue this forever
- There is no solution vector $\boldsymbol{a}$ satisfying for all $\boldsymbol{i}$

$$\boldsymbol{a}^t \boldsymbol{y}_i = \sum_{k=0}^{5} \boldsymbol{a}_k \boldsymbol{y}_i^{(k)} > 0$$

- Need to stop but at a good point

- Solutions at iterations 900 through 915
  - Some are good and some are not
- How do we stop at a good solution?



Convergence of Perceptron Rules

- If classes are linearly separable and we use fixed learning rate, that is for $\eta^{(k)}$ **=const**
- Then, *both the single sample and batch perceptron rules converge to a correct solution (could be any* ***a*** in the solution space)
- If classes are not linearly separable:
  - The algorithm does not stop, it keeps looking for a solution which does not exist

Convergence of Perceptron Rules

- If classes are not linearly separable:
  - By choosing appropriate learning rate, we can always ensure convergence: $\eta^{(k)} \to 0 \;\; as \;\; k \to \infty$
  - For example inverse linear learning rate: $\eta^{(k)} = \frac{\eta^{(1)}}{k}$
  - For inverse linear learning rate, convergence in the linearly separable case can also be proven
  - No guarantee that we stopped at a good point, but there are good reasons to choose inverse linear learning rate

Minimum Squared-Error Procedures

Minimum Squared-Error Procedures

- Idea: convert to easier and better understood problem

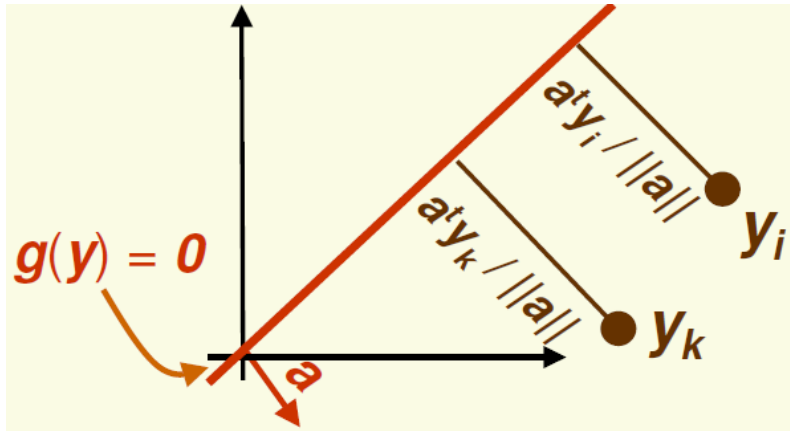
$\boldsymbol{a}^t\boldsymbol{y}_i > 0$ for all samples $\boldsymbol{y}_i$
solve system of linear inequalities

$\Downarrow$

$\boldsymbol{a}^t\boldsymbol{y}_i = \boldsymbol{b}_i$ for all samples $\boldsymbol{y}_i$
solve system of linear equations

- MSE procedure
  - Choose positive constants $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$
  - 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



- If $a^t y_i = b_i$, y_i must be at distance b_i from the separating hyperplane (normalized by $||a||$)
- Thus $b_1, b_2, \ldots, b_n$ 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 $b_1 = b_2 = \ldots = b_n = 1$

MSE Matrix Notation

- Need to solve **n** equations
- In matrix form **Ya=b**

$$\begin{cases} a^t y_1 = b_1 \\ \vdots \\ a^t y_n = b_n \end{cases}$$

$$\underbrace{\begin{bmatrix} y_1^{(0)} & y_1^{(1)} & \cdots & y_1^{(d)} \\ y_2^{(0)} & y_2^{(1)} & \cdots & y_2^{(d)} \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ y_n^{(0)} & y_n^{(1)} & \cdots & y_n^{(d)} \end{bmatrix}}_{Y} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix}}_{a} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}}_{b}$$

Exact Solution is Rare

- Need to solve a linear system $\mathbf{Ya} = \mathbf{b}$
  - $\mathbf{Y}$ is an $\mathbf{n} \times (\mathbf{d} + \mathbf{1})$ matrix
- Exact solution only if $\mathbf{Y}$ is non-singular and square (the inverse $\mathbf{Y}^{-1}$ exists)
  - $a = \mathbf{Y}^{-1}\mathbf{b}$
  - (number of samples) = (number of features + 1)
  - Almost never happens in practice
  - Guaranteed to find the separating hyperplane

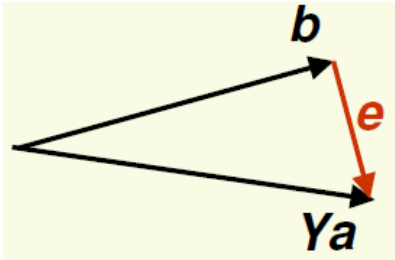
Approximate Solution

- Typically **Y** is overdetermined, that is it has more rows (examples) than columns (features)
  - If it has more features than examples, should reduce dimensionality
- Need **Ya = b**, but no exact solution exists for an over-determined system of equations
  - More equations than unknowns
- Find an approximate solution
  - Note that approximate solution **a** does **not** necessarily give the separating hyperplane in the separable case
  - But the hyperplane corresponding to **a** may still be a good solution, especially if there is no separating hyperplane

MSE Criterion Function

- Minimum squared error approach: find **a** which minimizes the length of the error vector **e**

$$\mathbf{e} = \mathbf{Ya} - \mathbf{b}$$



- Thus minimize the minimum squared error criterion function:

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

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

Computing the Gradient

$$J_s(a) = \|Ya - b\|^2 = \sum_{i=1}^{n} \left(a^t y_i - b_i\right)^2$$

$$\nabla J_s(a) = \begin{bmatrix} \frac{\partial J_s}{\partial a_0} \\ \vdots \\ \frac{\partial J_s}{\partial a_d} \end{bmatrix} = \frac{dJ_s}{da} = \sum_{i=1}^{n} \frac{d}{da}\left(a^t y_i - b_i\right)^2$$

$$= \sum_{i=1}^{n} 2\left(a^t y_i - b_i\right)\frac{d}{da}\left(a^t y_i - b_i\right)$$

$$= \sum_{i=1}^{n} 2\left(a^t y_i - b_i\right) y_i$$

$$= 2Y^t(Ya - b)$$

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 **a** uniquely:

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

pseudo inverse of $\mathbf{Y}$

$$\left((\mathbf{Y}^t\mathbf{Y})^{-1}\mathbf{Y}^t\right)\mathbf{Y} = (\mathbf{Y}^t\mathbf{Y})^{-1}(\mathbf{Y}^t\mathbf{Y}) = \mathbf{I}$$

MSE Procedures

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

$$\boldsymbol{Ya} = \begin{bmatrix} b_1 + \varepsilon_1 \\ \vdots \\ b_n + \varepsilon_n \end{bmatrix}$$

  - where ε may be negative
- If $\varepsilon_1, \ldots, \varepsilon_n$ are small relative to $b_1, \ldots, b_n$, then each element of $\mathbf{Ya}$ 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

MSE Procedures

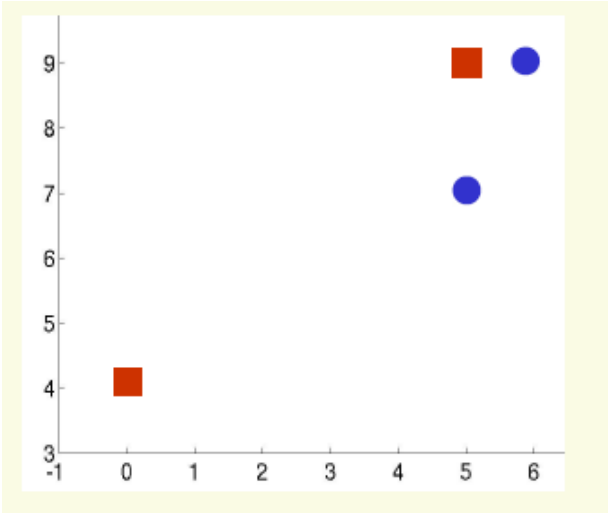
- We are free to choose **b**. We may be tempted to make **b** large as a way to ensure **Ya** =**b** > 0
  - Does not work
  - Let β be a scalar, let's try β**b** instead of **b**
- If **a*** is a least squares solution to **Ya = b**, then for any scalar β, the least squares solution to **Ya** = β**b** is β**a***

$$\arg\min_a \|Ya - \beta b\|^2 = \arg\min_a \beta^2 \|Y(a/\beta) - b\|^2 = \beta a^*$$

- Thus if the **i** th element of **Ya** is less than **0**, that is $y_i^t a < 0$, then $y_i^t(\beta a) < 0$,
  - The relative difference between components of **b** matters, but not the size of each individual component

LDF using MSE: Example 1

- Class 1: (6 9), (5 7)
- Class 2: (5 9), (0 4)
- Add extra feature and "normalize"



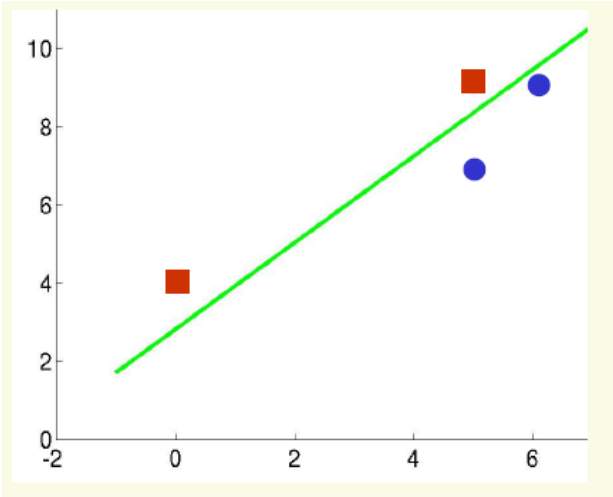
$$y_1 = \begin{bmatrix} 1 \\ 6 \\ 9 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 5 \\ 7 \end{bmatrix} \quad y_3 = \begin{bmatrix} -1 \\ -5 \\ -9 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ 0 \\ -4 \end{bmatrix}$$

$$Y = \begin{bmatrix} 1 & 6 & 9 \\ 1 & 5 & 7 \\ -1 & -5 & -9 \\ -1 & 0 & -4 \end{bmatrix}$$

LDF using MSE: Example 1

- Choose **b=[1 1 1 1]**^{T}
- In Matlab, **a=Y\b** solves the least squares problem

$$a = \begin{bmatrix} 2.66 \\ 1.045 \\ -0.944 \end{bmatrix}$$

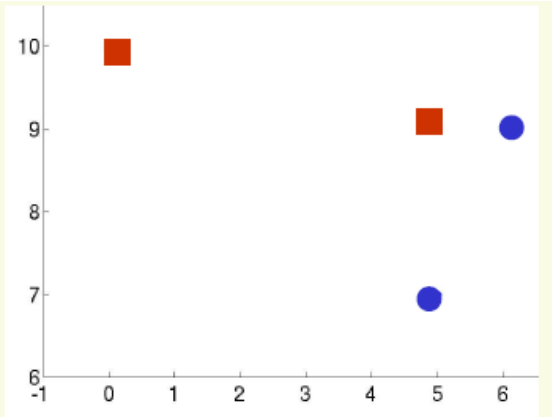


- Note **a** is an approximation to **Ya = b**, since no exact solution exists
- This solution gives a separating hyperplane since **Ya >0**

$$Ya = \begin{bmatrix} 0.44 \\ 1.28 \\ 0.61 \\ 1.11 \end{bmatrix}$$

LDF using MSE: Example 2

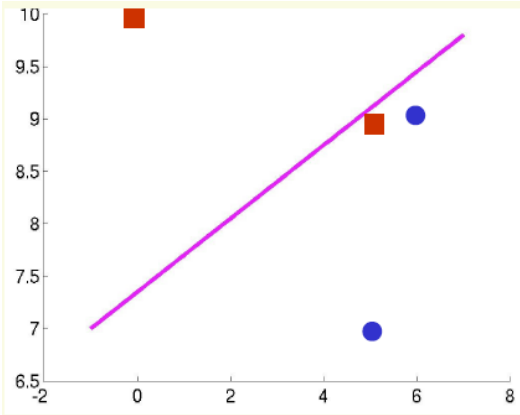
- Class 1: (6 9), (5 7)
- Class 2: (5 9), (0 10)
- The last sample is very far compared to others from the separating hyperplane



$$y_1 = \begin{bmatrix} 1 \\ 6 \\ 9 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 5 \\ 7 \end{bmatrix} \quad y_3 = \begin{bmatrix} -1 \\ -5 \\ -9 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ 0 \\ -10 \end{bmatrix}$$

$$Y = \begin{bmatrix} 1 & 6 & 9 \\ 1 & 5 & 7 \\ -1 & -5 & -9 \\ -1 & 0 & -10 \end{bmatrix}$$

LDF using MSE: Example 2



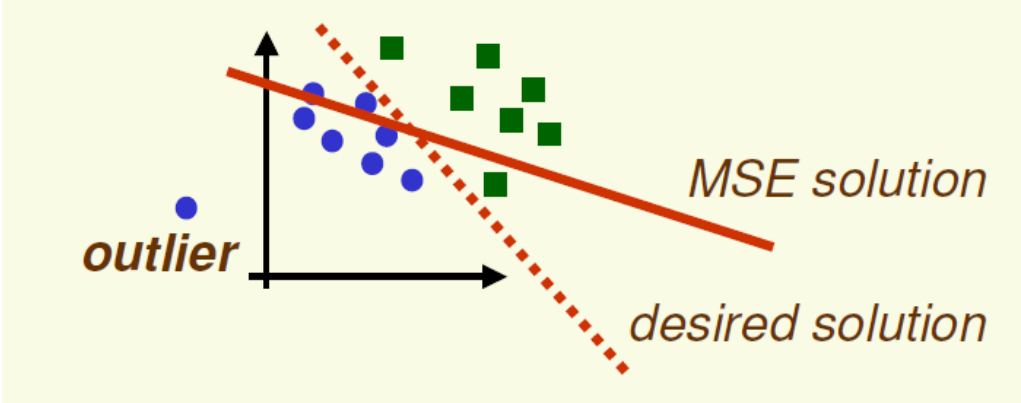
- Choose **b=[1 1 1 1]**^{T}
- In Matlab, **a=Y\b** solves the least squares problem

$$a = \begin{bmatrix} 3.2 \\ 0.2 \\ -0.4 \end{bmatrix} \qquad Ya = \begin{bmatrix} 0.2 \\ 0.9 \\ -0.04 \\ 1.16 \end{bmatrix} \neq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

- This solution does not provide a separating hyperplane since $\mathbf{a}^t\mathbf{y}_3 < 0$

LDF using MSE: Example 2

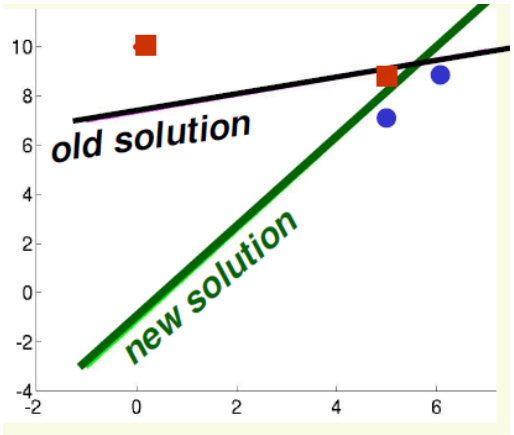
- MSE pays too much attention to isolated "noisy" examples
  - such examples are called outliers



- No problems with convergence
- Solution ranges from reasonable to good

LDF using MSE: Example 2

- We can see that the 4th point is vary far from separating hyperplane
  - In practice we don't know this
- A more appropriate **b** could be $\boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 10 \end{bmatrix}$
- In Matlab, solve **a=Y\b**



$$\boldsymbol{a} = \begin{bmatrix} -1.1 \\ 1.7 \\ -0.9 \end{bmatrix}$$

$$\boldsymbol{Ya} = \begin{bmatrix} 0.9 \\ 1.0 \\ 0.8 \\ 10.0 \end{bmatrix} \neq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 10 \end{bmatrix}$$

- This solution gives the separating hyperplane since **Ya > 0**

Gradient Descent for MSE

$$J_s(\boldsymbol{a}) = \|\boldsymbol{Ya} - \boldsymbol{b}\|^2$$

- May wish to find MSE solution by gradient descent:
  1. Computing the inverse of Y^tY may be too costly
  2. Y^tY may be close to singular if samples are highly correlated (rows of Y are almost linear combinations of each other) computing the inverse of Y^tY is not numerically stable
- As shown before, the gradient is:

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

Widrow-Hoff Procedure

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

- Thus the update rule for gradient descent is:

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

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

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

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 even if classes are linearly separable
  - Use pseudoinverse if $\mathbf{Y}^t\mathbf{Y}$ is not singular and not too large
  - Use gradient descent (Widrow-Hoff procedure) otherwise

Support Vector Machines

SVM Resources

- Burges tutorial
  - http://research.microsoft.com/en-us/um/people/cburges/papers/SVMTutorial.pdf
- Shawe-Taylor and Christianini tutorial
  - http://www.support-vector.net/icml-tutorial.pdf
- Lib SVM
  - http://www.csie.ntu.edu.tw/~cjlin/libsvm/
- LibLinear
  - http://www.csie.ntu.edu.tw/~cjlin/liblinear/
- SVM Light
  - http://svmlight.joachims.org/
- Power Mean SVM (very fast for histogram features)
  - https://sites.google.com/site/wujx2001/home/power-mean-svm

SVMs

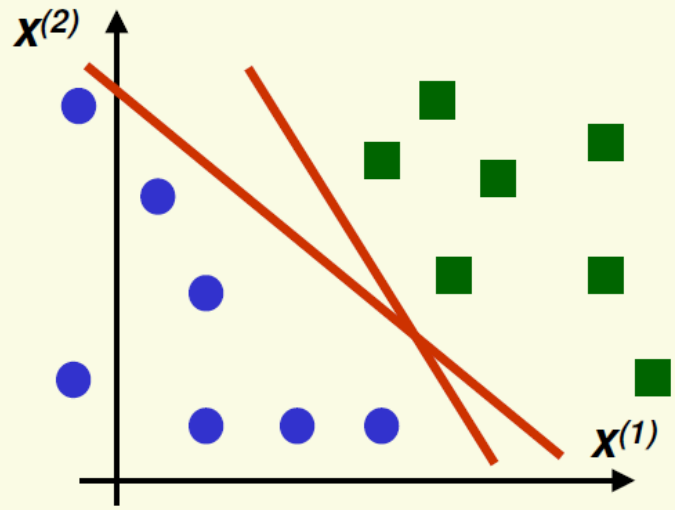
- One of the most important developments in pattern recognition in the last years
- Elegant theory
  - Has good generalization properties
- Have been applied to diverse problems very successfully

Linear Discriminant Functions

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

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

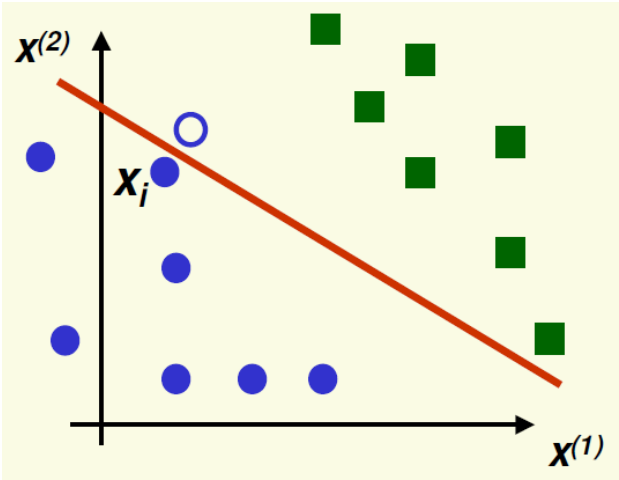
$$\boxed{\begin{array}{ll} g(\mathbf{x}) > 0 & \Rightarrow \mathbf{x} \in \textit{class } 1 \\ g(\mathbf{x}) < 0 & \Rightarrow \mathbf{x} \in \textit{class } 2 \end{array}}$$



- which separating hyperplane should we choose?

Linear Discriminant Functions

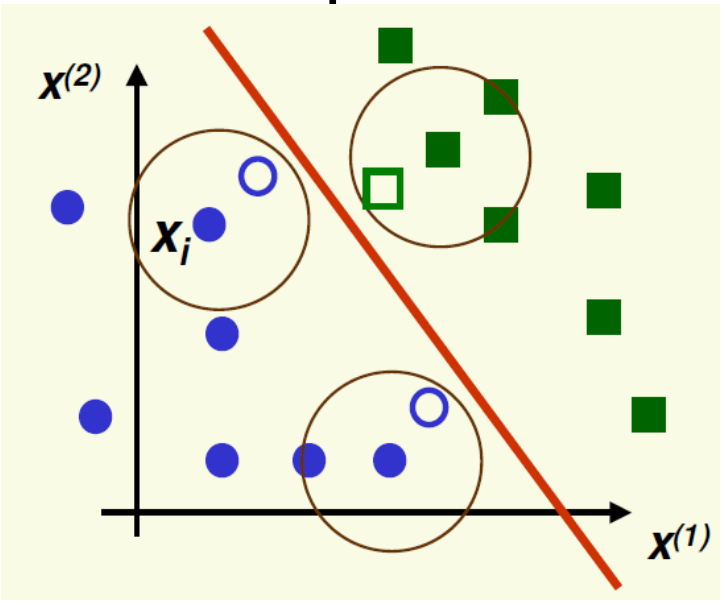
- Training data is just a subset of all possible data
  - Suppose hyperplane is close to sample $\mathbf{x}_i$
  - If we see new sample close to $\mathbf{x}_i$, it may be on the wrong side of the hyperplane



- Poor generalization (performance on unseen data)

Linear Discriminant Functions

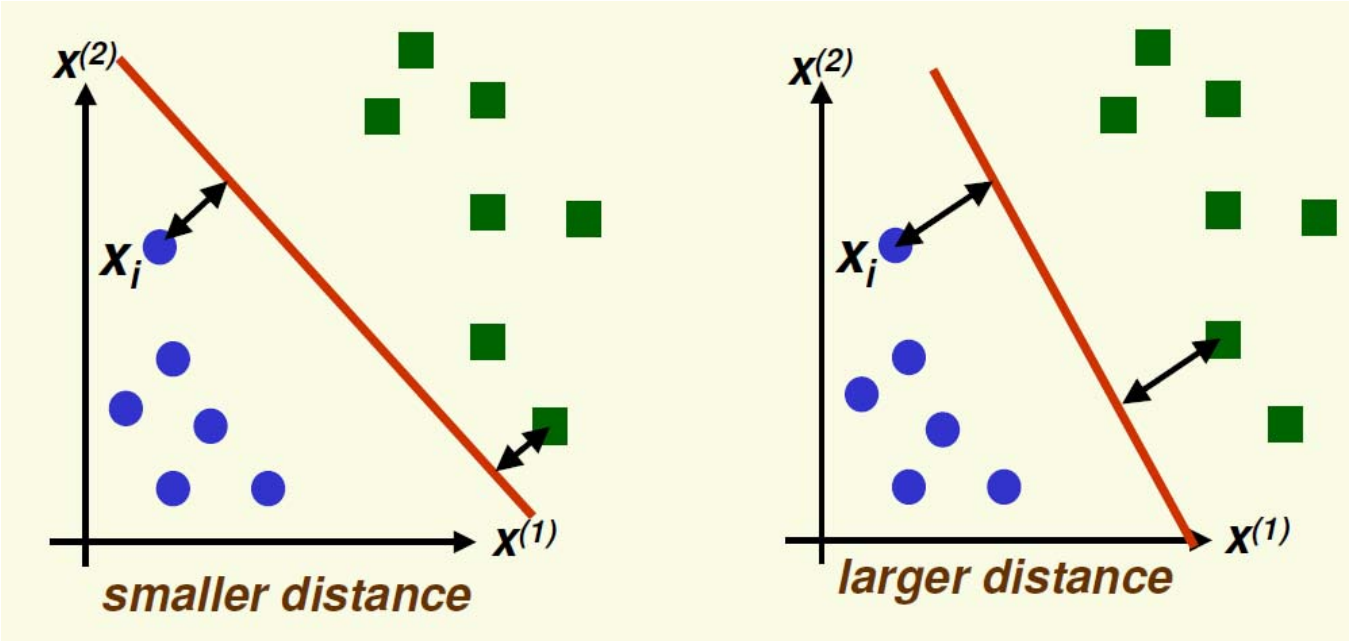
- Hyperplane as far as possible from any sample



- New samples close to the old samples will be classified correctly
- Good generalization

SVM

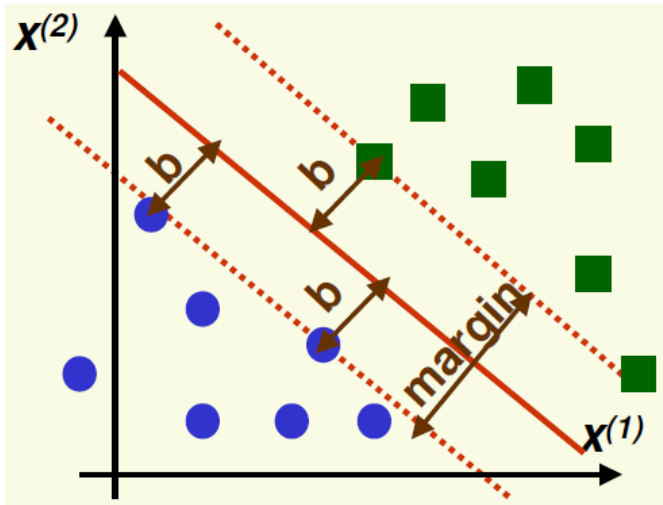
- Idea: maximize distance to the closest example



- For the optimal hyperplane
  - distance to the closest negative example = distance to the closest positive example

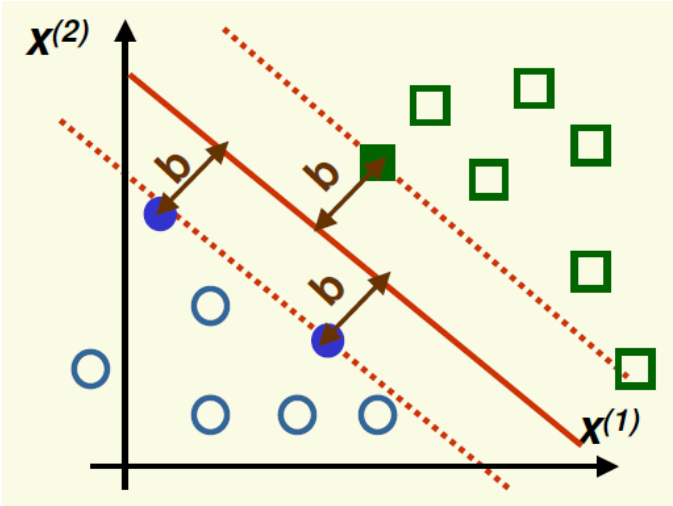
SVM: Linearly Separable Case

- 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

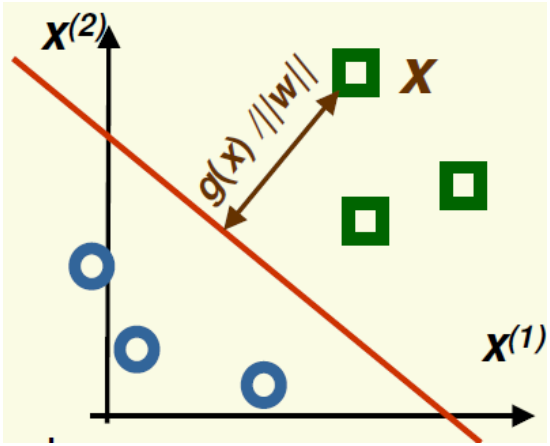
SVM: Linearly Separable Case



- ***Support vectors*** are the samples closest to the separating hyperplane
  - They are the most difficult patterns to classify
  - Recall perceptron update rule
- 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 **x** and the boundary **g(x) = 0**

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

- Distance is unchanged for hyperplane

$$g_1(\mathbf{x}) = \alpha g(\mathbf{x}) \qquad \frac{|\alpha\mathbf{w}^t\mathbf{x} + \alpha w_0|}{\|\alpha\mathbf{w}\|} = \frac{|\mathbf{w}^t\mathbf{x} + w_0|}{\|\mathbf{w}\|}$$

- Let $\mathbf{x}_i$ be an example closest to the boundary (on the positive side). Set: $|\mathbf{w}^t\mathbf{x}_i + w_0| = 1$

- Now the largest margin hyperplane is unique

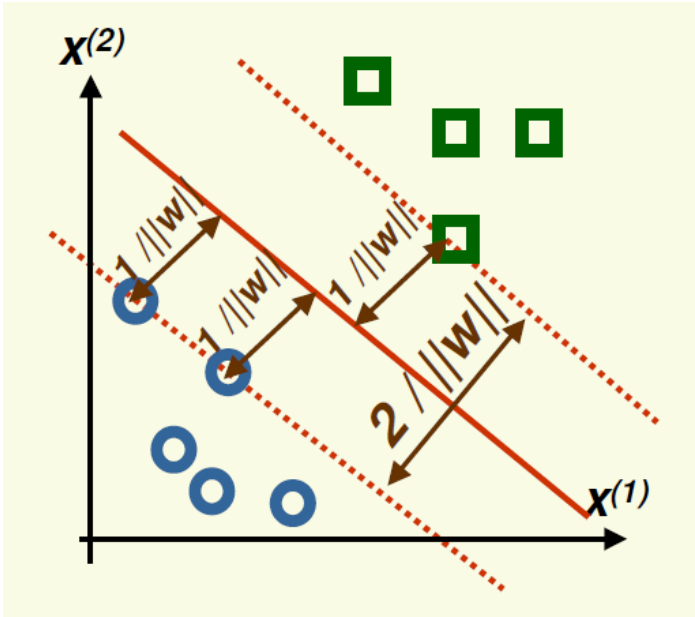
SVM: Formula for the Margin

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

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

- Thus the margin is

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



SVM: Optimal Hyperplane

- Maximize margin $\boxed{m = \frac{2}{\|w\|}}$
- Subject to constraints $\begin{cases} w^t x_i + w_0 \geq 1 & \text{if } x_i \text{ is positive example} \\ w^t x_i + w_0 \leq -1 & \text{if } x_i \text{ is negative example} \end{cases}$
- Let $\begin{cases} z_i = 1 & \text{if } x_i \text{ is positive example} \\ z_i = -1 & \text{if } x_i \text{ is negative example} \end{cases}$
- Can convert our problem to minimize

$$\boxed{\begin{array}{ll} \text{minimize} & J(w) = \frac{1}{2}\|w\|^2 \\ \text{constrained to} & z_i\left(w^t x_i + w_0\right) \geq 1 \quad \forall i \end{array}}$$

- **J(w)** is a quadratic function, thus there is a single global minimum

SVM: Optimal Hyperplane

- Use Kuhn-Tucker theorem to convert our problem to:

$$\text{maximize} \quad L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j z_i z_j x_i^t x_j$$

$$\text{constrained to} \quad \alpha_i \geq 0 \quad \forall i \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i z_i = 0$$

- $a = \{a_1, \ldots, a_n\}$ are new variables, one for each sample
- Optimized by quadratic programming

SVM: Optimal Hyperplane

- After finding the optimal $\boldsymbol{a} = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$
- Final discriminant function:

$$g(\mathbf{x}) = \left(\sum_{\mathbf{x}_i \in S} \alpha_i z_i \mathbf{x}_i \right)^t \mathbf{x} + w_0$$

- where ***S is the set of support vectors***

$$S = \{\mathbf{x}_i \mid \alpha_i \neq 0\}$$

SVM: Optimal Hyperplane

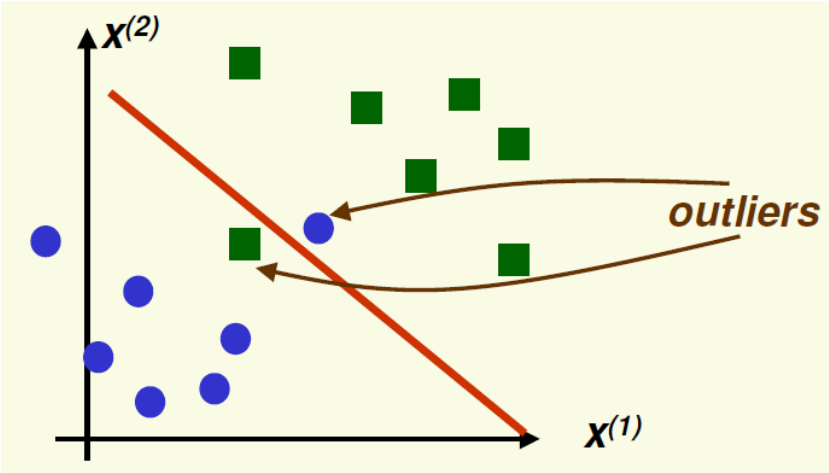
$$\text{maximize} \quad L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j z_i z_j \mathbf{x}_i^t \mathbf{x}_j$$

$$\text{constrained to} \quad \alpha_i \geq 0 \quad \forall i \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i z_i = 0$$

- $L_D(\mathbf{a})$ depends on the number of samples, not on dimension
  - samples appear only through the dot products $\mathbf{x}_j^t \mathbf{x}_i$
- This will become important when looking for a nonlinear discriminant function, as we will see soon

SVM: Non-Separable Case

- Data are most likely to be not linearly separable, but linear classifier may still be appropriate



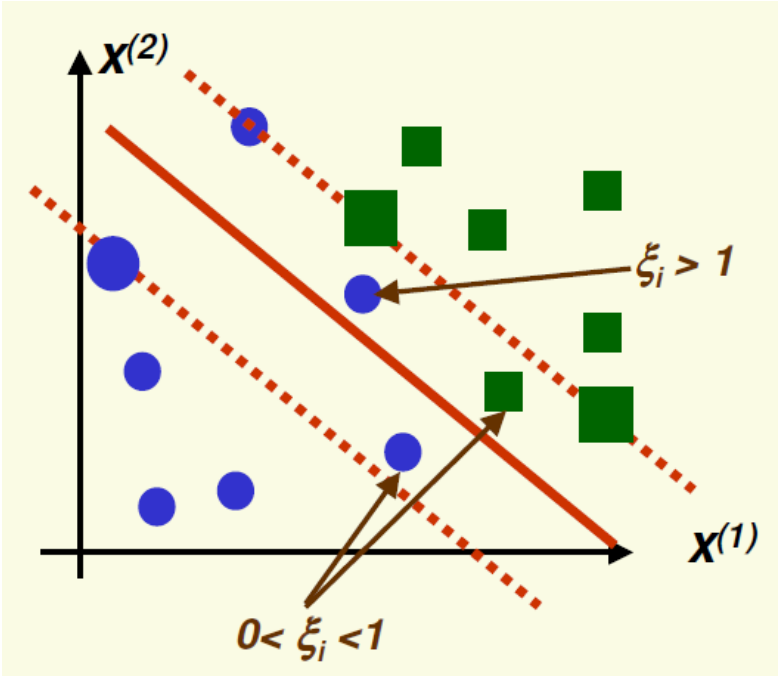
- Can apply SVM in non linearly separable case
- Data should be “almost” linearly separable for good performance

SVM: Non-Separable Case

- Use slack variables $\xi_1,\ldots,\xi_n$ (one for each sample)
- Change constraints from $z_i(w^t x_i + w_o) \geq 1 \quad \forall i$ to

  $z_i(w^t x_i + w_o) \geq 1 - \xi_i \quad \forall i$

- ξ_i is a measure of deviation from the ideal for x_i
  - $\xi_i > 1$: x_i is on the wrong side of the separating hyperplane
  - $0 < \xi_i < 1$: x_i is on the right side of separating hyperplane but within the region of maximum margin
  - $\xi_i < 0$: is the ideal case for x_i



SVM: Non-Separable Case

- We would like to minimize

$$J(\mathbf{w},\xi_1,\ldots,\xi_n)=\frac{1}{2}\|\mathbf{w}\|^2+\beta\sum_{i=1}^{n}I(\xi_i>0)$$

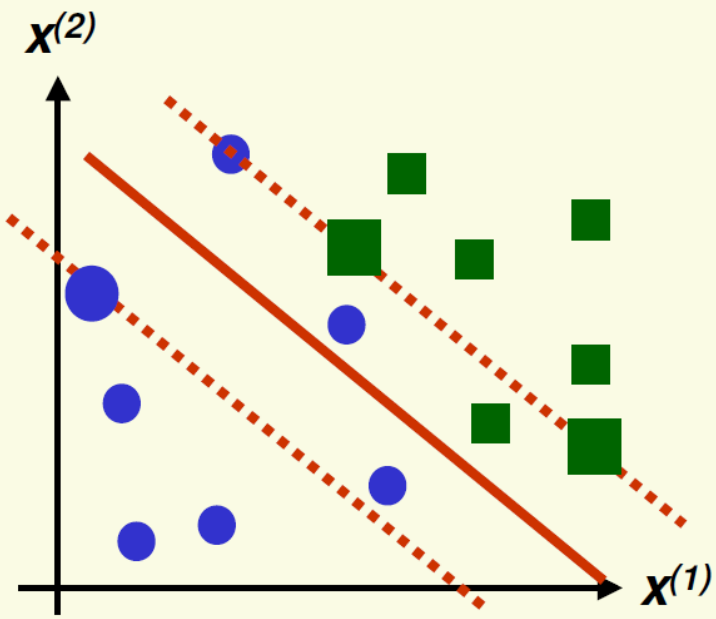
# of samples not in ideal location

- where $I(\xi_i>0)=\begin{cases}1 & \text{if } \xi_i>0\\ 0 & \text{if } \xi_i\le 0\end{cases}$
- Constrained to $z_i\left(\mathbf{w}^t\mathbf{x}_i+w_0\right)\ge 1-\xi_i$ and $\xi_i\ge 0 \quad \forall i$
- β is a constant that measures the relative weight of first and second term
  - If β is small, we allow a lot of samples to be in not ideal positions
  - If β is large, few samples can be in non-ideal positions

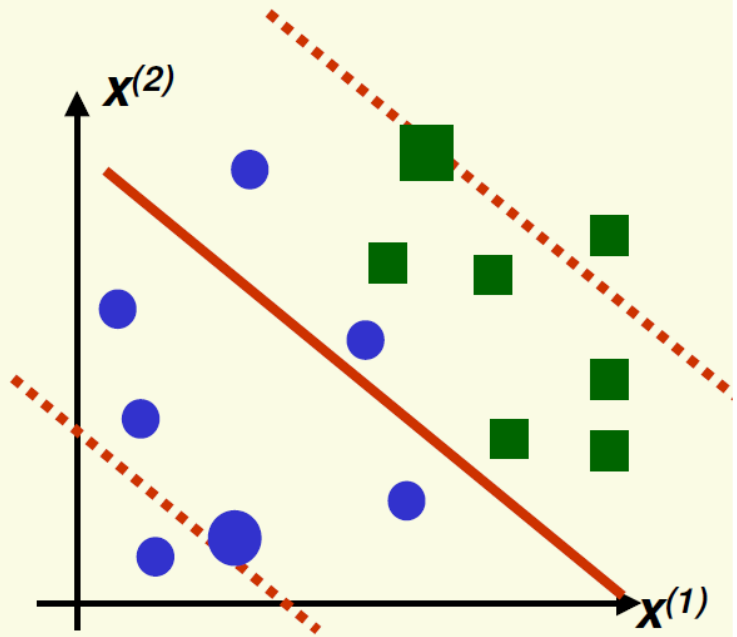
SVM: Non-Separable Case

$$J(w,\xi_1,\ldots,\xi_n)=\frac{1}{2}\|w\|^2+\beta\sum_{i=1}^{n} I(\xi_i>0)$$

# of examples not in ideal location



large β, few samples not in ideal position



small β, a lot of samples not in ideal position

SVM: Non-Separable Case

- Unfortunately this minimization problem is NP-hard due to the discontinuity of $I(\xi_i)$
- Instead, we minimize

$$J(\mathbf{w}, \xi_1, \ldots, \xi_n) = \frac{1}{2}\|\mathbf{w}\|^2 + \beta \sum_{i=1}^{n} \xi_i$$

a measure of # of misclassified examples

- Subject to

$$\begin{cases} z_i\left(\mathbf{w}^t \mathbf{x}_i + w_0\right) \geq 1 - \xi_i & \forall i \\ \xi_i \geq 0 & \forall i \end{cases}$$

SVM: Non-Separable Case

- Use Kuhn-Tucker theorem to convert to:

$$\text{maximize} \quad L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_i z_i z_j x_i^t x_j$$

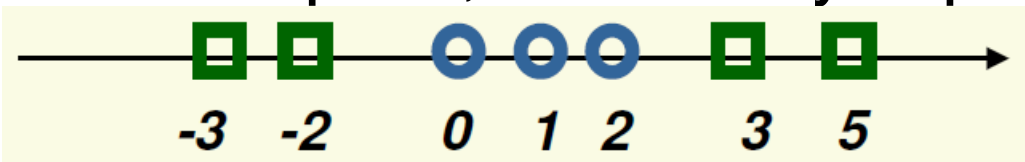
$$\text{constrained to} \quad 0 \le \alpha_i \le \beta \quad \forall i \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i z_i = 0$$

- **w** is computed using: $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i z_i \boldsymbol{x}_i$

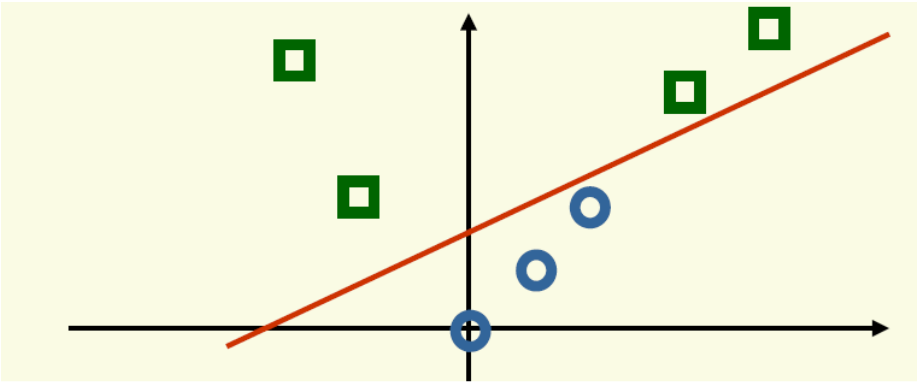
- Remember that $g(\boldsymbol{x}) = \left(\sum_{x_i \in S} \alpha_i z_i \boldsymbol{x}_i \right)^t \boldsymbol{x} + w_0$

Nonlinear Mapping

- Cover's theorem: "*a pattern-classification problem cast in a high dimensional space non-linearly is more likely to be linearly separable than in a low-dimensional space*"
- One dimensional space, not linearly separable



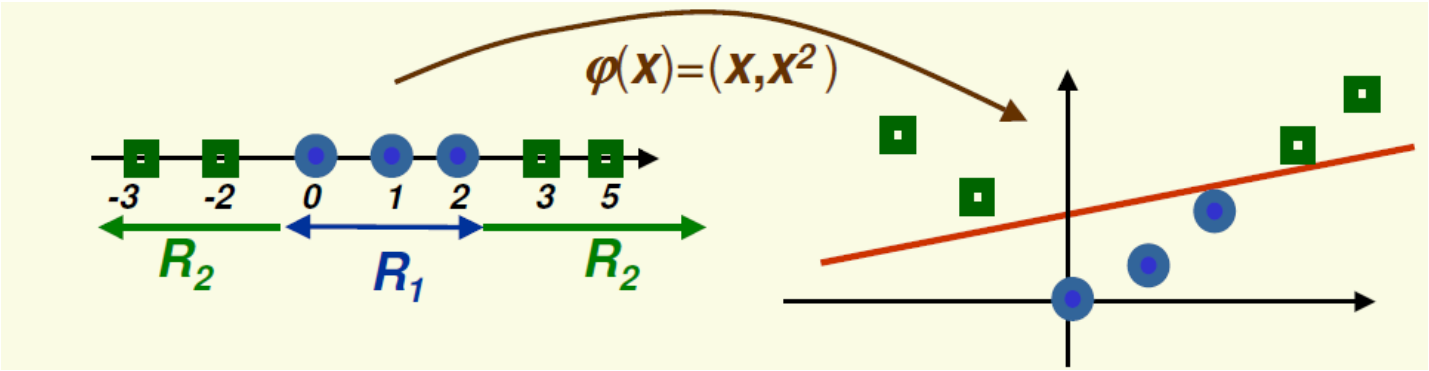
- Lift to two dimensional space with $\varphi(x)=(x,x^2)$



Nonlinear Mapping

- To solve a non linear classification problem with a linear classifier

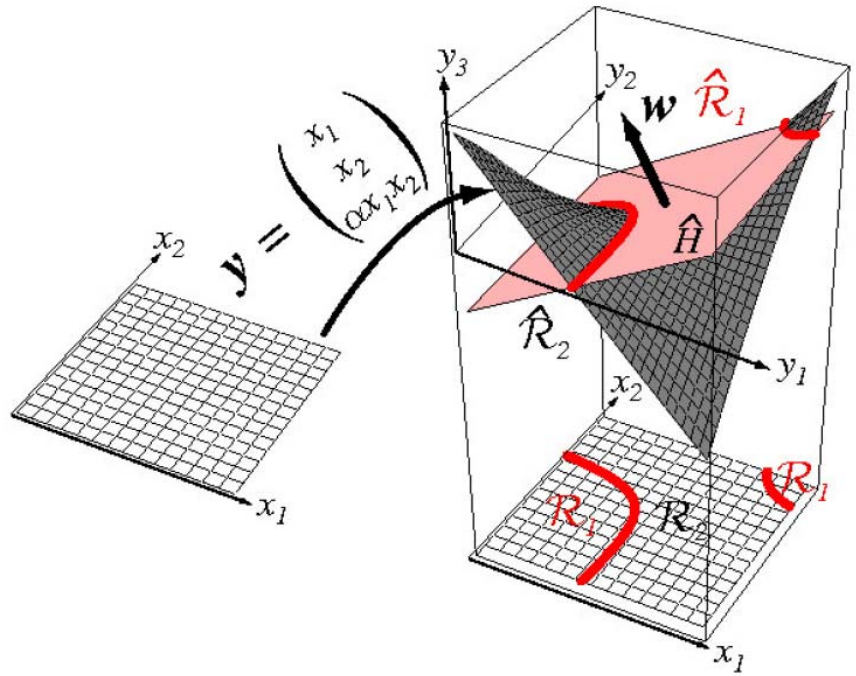
1. Project data $\boldsymbol{x}$ to high dimension using function $\varphi(x)$
2. Find a linear discriminant function for transformed data $\varphi(x)$
3. Final nonlinear discriminant function is $g(x) = w^t\varphi(x) + w_0$



- In 2D, the discriminant function is linear $g\left(\begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix}\right) = \begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} + w_0$

- In 1D, the discriminant function is not linear $g(x) = w_1 x + w_2 x^2 + w_0$

Nonlinear Mapping



- However, there always exists a mapping of N samples to an N-dimensional space in which the samples are separable by hyperplanes

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***

Kernels

- SVM optimization:

  maximize $$L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_i z_i z_j x_i^t x_j$$

- Note this optimization depends on samples x_i only through the dot product $x_i^t x_j$
- If we lift x_i to high dimension using $\varphi(x)$, we need to compute high dimensional product $\varphi(x_i)^t \varphi(x_j)$

  maximize $$L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_i z_i z_j \underbrace{\varphi(x_i)^t \varphi(x_j)}_{K(x_i, x_j)}$$

- Idea: find kernel function $K(x_i,x_j)$ s.t. $K(x_i, x_j) = \varphi(x_i)^t \varphi(x_j)$

Kernel Trick

- Then we only need to compute $K(\mathbf{x}_i,\mathbf{x}_j)$ instead of $\boldsymbol{\varphi}(\mathbf{x}_i)^t\,\boldsymbol{\varphi}(\mathbf{x}_j)$
- "kernel trick": do not need to perform operations in high dimensional space explicitly

Kernel Example

- Suppose we have two features and **K(x,y) = $(x^t y)^2$**
- Which mapping **φ(x)** does this correspond to?

$$K(x,y) = (x^t y)^2 = \left(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix} \begin{bmatrix} y^{(1)} \\ y^{(2)} \end{bmatrix} \right)^2 = \left(x^{(1)} y^{(1)} + x^{(2)} y^{(2)} \right)^2$$

$$= \left(x^{(1)} y^{(1)} \right)^2 + 2\left(x^{(1)} y^{(1)} \right)\left(x^{(2)} y^{(2)} \right) + \left(x^{(2)} y^{(2)} \right)^2$$

$$= \begin{bmatrix} \left(x^{(1)} \right)^2 & \sqrt{2} x^{(1)} x^{(2)} & \left(x^{(2)} \right)^2 \end{bmatrix} \begin{bmatrix} \left(y^{(1)} \right)^2 & \sqrt{2} y^{(1)} y^{(2)} & \left(y^{(2)} \right)^2 \end{bmatrix}^t$$

$$\varphi(x) = \begin{bmatrix} \left(x^{(1)} \right)^2 & \sqrt{2} x^{(1)} x^{(2)} & \left(x^{(2)} \right)^2 \end{bmatrix}$$

Choice of Kernel

- How to choose kernel function $K(x_i,x_j)$?
  - $K(x_i,x_j)$ should correspond to $\varphi(x_i)^t\ \varphi(x_j)$ in a higher dimensional space
  - Mercer's condition tells us which kernel function can be expressed as dot product of two vectors
  - If K and K' are kernels aK+bK' is a kernel
- Intuitively: Kernel should measure the similarity between x_i and x_j
  - As inner product measures similarity of unit vectors
  - May be problem-specific

Choice of Kernel

- Some common choices:
  - Polynomial kernel

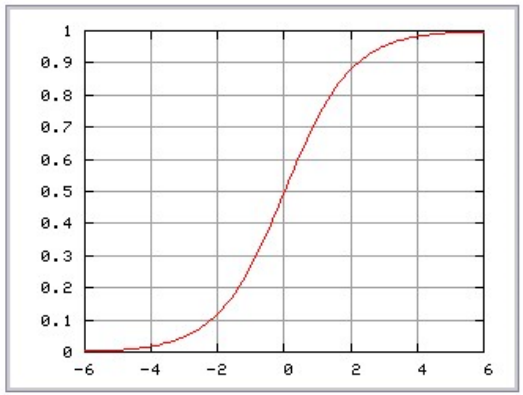
$$K(x_i, x_j) = (x_i^t x_j + 1)^p$$

  - Gaussian radial Basis kernel

$$K(x_i, x_j) = \exp\left(-\frac{1}{2\sigma^2}\|x_i - x_j\|^2\right)$$

  - Hyperbolic tangent (sigmoid) kernel

$$K(x_i,x_j) = \tanh(k\, x_i^t x_j + c)$$



- The mappings $\varphi(x_i)$ never have to be computed!!

Intersection Kernel

- Feature vectors are histograms

$$K(x_i, x_j) = \sum_{k=1}^{n} \min(x_{ik}, x_{jk})$$

- When $K(x_i,x_j)$ is small, x_i and x_j are dissimilar
- When $K(x_i,x_j)$ is large, x_i and x_j are similar

- The mapping $\varphi(x)$ does not exist

More Additive Kernels

- χ^2 kernel $K_{\chi^2} = \sum_{k=1}^{n} \frac{2x_k y_k}{x_k + y_k}$
- Hellinger's kernel $K_H = \sum_{k=1}^{n} \sqrt{x_k y_k}$
- Designed for feature vectors that are histograms
  - Can be used for other feature vectors
- Offer very large speed-ups

The Kernel Matrix

- a.k.a the Gram matrix

K=

K(1,1)	K(1,2)	K(1,3)	…	K(1,m)
K(2,1)	K(2,2)	K(2,3)	…	K(2,m)
…	…	…	…	…
K(m,1)	K(m,2)	K(m,3)	…	K(m,m)

- Contains all necessary information for the learning algorithm
- Fuses information about the data and the kernel (similarity measure)

Bad Kernels

- The kernel matrix is mostly diagonal
  - All points are orthogonal to each other
- Bad similarity measure
- Too many irrelevant features in high dimensional space

- We need problem-specific knowledge to choose appropriate kernel

Nonlinear SVM Step-by-Step

- Start with data $\mathbf{x}_1,\ldots,\mathbf{x}_n$ which live in feature space of dimension $\mathbf{d}$
- Choose kernel $\mathbf{K}(\mathbf{x}_i,\mathbf{x}_j)$ or function $\boldsymbol{\varphi}(\mathbf{x}_i)$ which lifts sample $\mathbf{x}_i$ to a higher dimensional space
- Find the maximum margin linear discriminant function in the higher dimensional space by using quadratic programming package to solve:

$$\text{maximize} \quad L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_i z_i z_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{constrained to} \quad 0 \le \alpha_i \le \beta \quad \forall i \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i z_i = 0$$

Nonlinear SVM Step-by-Step

- Weight vector **w** in the high dimensional space:

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

  – where **S** is the set of support vectors

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

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

- Non linear discriminant function in the original space:

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

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

Nonlinear SVM

- Nonlinear discriminant function

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

$$g(\boldsymbol{x}) = \sum$$

weight of support vector $\boldsymbol{x}_i$

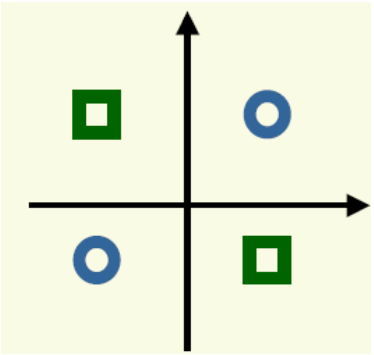
∓ 1

"inverse distance" from $\boldsymbol{x}$ to support vector $\boldsymbol{x}_i$

most important training samples, i.e. support vectors

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

SVM Example: XOR Problem



- Class 1: $x_1 = [1,-1]$, $x_2 = [-1,1]$
- Class 2: $x_3 = [1,1]$, $x_4 = [-1,-1]$
- Use polynomial kernel of degree 2:

$$K(x_i, x_j) = (x_i^t x_j + 1)^2$$

  – This kernel corresponds to the mapping

$$\varphi(x) = \left[1 \quad \sqrt{2}x^{(1)} \quad \sqrt{2}x^{(2)} \quad \sqrt{2}x^{(1)}x^{(2)} \quad \left(x^{(1)}\right)^2 \quad \left(x^{(2)}\right)^2\right]^t$$

- Need to maximize

$$L_D(\alpha) = \sum_{i=1}^{4} \alpha_i - \frac{1}{2}\sum_{i=1}^{4}\sum_{j=1}^{4} \alpha_i \alpha_j z_i z_j \left(x_i^t x_j + 1\right)^2$$

constrained to $0 \le \alpha_i \quad \forall i \quad \text{and} \quad \alpha_1 + \alpha_2 - \alpha_3 - \alpha_4 = 0$

SVM Example: XOR Problem

- After some manipulation …
- The solution is $a_1 = a_2 = a_3 = a_4 = 0.25$
  - satisfies the constraints

$$\forall i,\ \ 0 \le \alpha_i \ \ \textit{and} \ \ \alpha_1 + \alpha_2 - \alpha_3 - \alpha_4 = 0$$

- All samples are support vectors

SVM Example: XOR Problem

$$\varphi(\mathbf{x}) = \begin{bmatrix} 1 & \sqrt{2}x^{(1)} & \sqrt{2}x^{(2)} & \sqrt{2}x^{(1)}x^{(2)} & \left(x^{(1)}\right)^2 & \left(x^{(2)}\right)^2 \end{bmatrix}^t$$

- The weight vector **w** is:

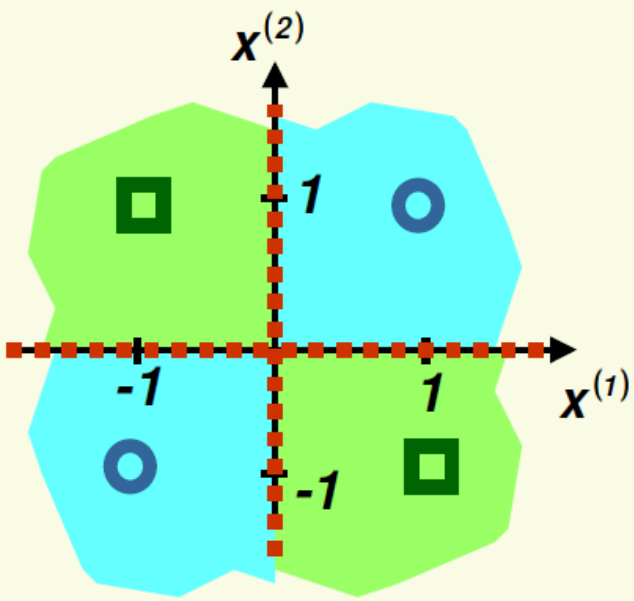
$$\begin{aligned} \mathbf{w} = \sum_{i=1}^{4} \alpha_i z_i \varphi(\mathbf{x}_i) &= 0.25\left(\varphi(\mathbf{x}_1) + \varphi(\mathbf{x}_2) - \varphi(\mathbf{x}_3) - \varphi(\mathbf{x}_4)\right) \\ &= \begin{bmatrix} 0 & 0 & 0 & -\sqrt{2} & 0 & 0 \end{bmatrix} \end{aligned}$$

- Thus the nonlinear discriminant function is:

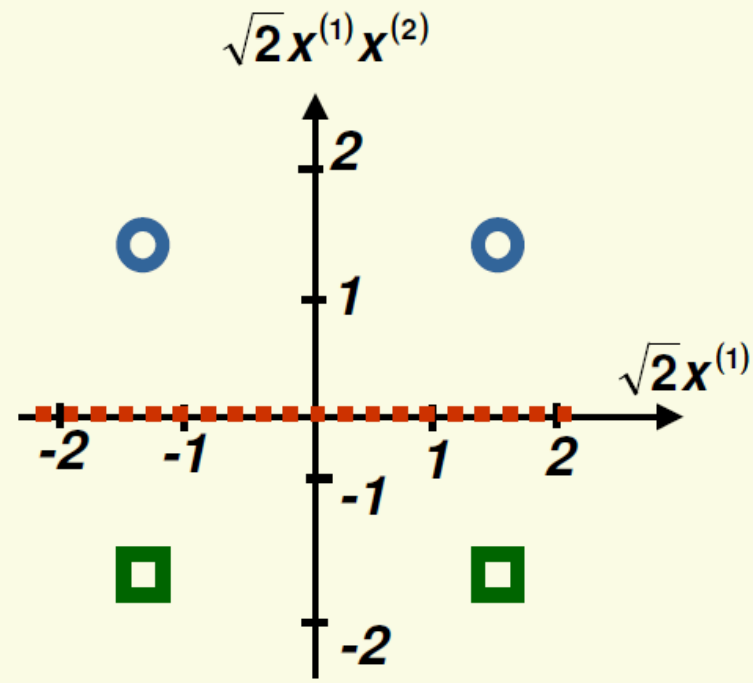
$$g(\mathbf{x}) = \mathbf{w}\varphi(\mathbf{x}) = \sum_{i=1}^{6} w_i \varphi_i(\mathbf{x}) = -\sqrt{2}\left(\sqrt{2}x^{(1)}x^{(2)}\right) = -2x^{(1)}x^{(2)}$$

SVM Example: XOR Problem

$$g(x) = -2x^{(1)}x^{(2)}$$



decision boundaries nonlinear



decision boundary is linear

SVM Summary

- Advantages:
  - Based on very strong 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:
  - Directly applicable to two-class problems
  - Quadratic programming is computationally expensive
  - Need to choose kernel

Multi-Class SVMs

- One against all
- Pairwise

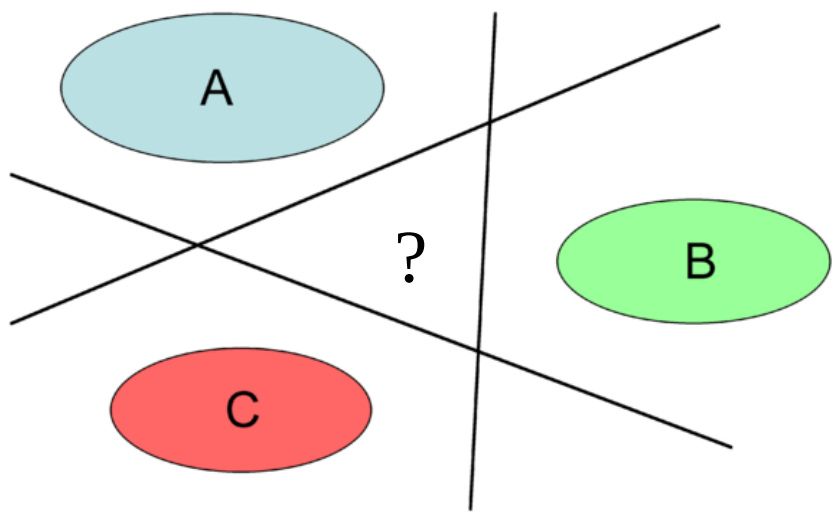
- These ideas apply to all binary classifiers when faced with multi-class problems

One-Against-All

- SVMs can only handle two-class outputs
- What can be done?
- Answer: learn N SVM's
  - SVM 1 learns "Output==1" vs "Output != 1"
  - SVM 2 learns "Output==2" vs "Output != 2"
  - …
  - SVM N learns "Output==N" vs "Output != N"

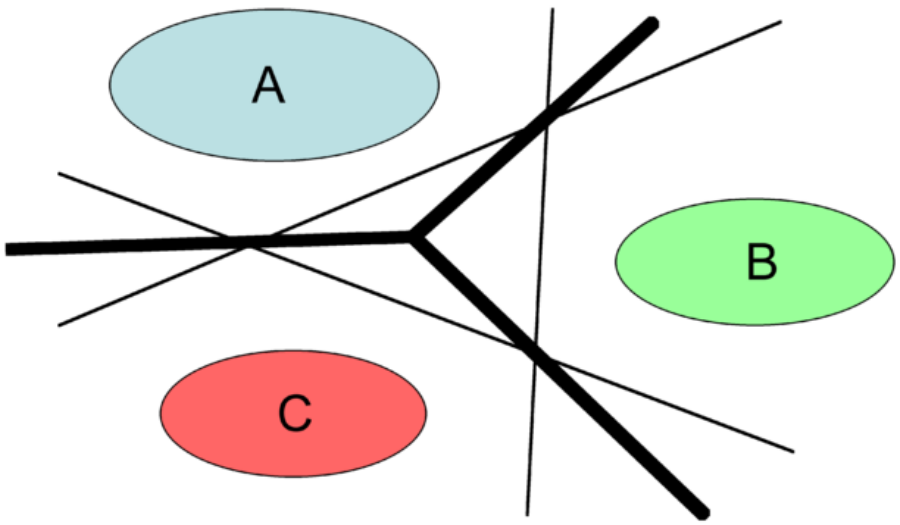
One-Against-All

- Original idea (Vapnik, 1995): classify x as ω_i if and only if the corresponding SVM accepts x and all other SVMs reject it



One-Against-All

- Modified idea (Vapnik, 1998): classify x according to the SVM that produces the highest value (use more than sign of decision function)



Pairwise SVMs

- Learn N(N-1)/2 SVM's
  - SVM 1 learns "Output==1" vs "Output == 2"
  - SVM 2 learns "Output==1" vs "Output == 3"
  - …
  - SVM M learns "Output==N-1" vs "Output == N"

Pairwise SVMs

- To classify a new input, apply each SVM and choose the label that “wins” most often