Linear Models

SHALA - 2020

https://shala2020.github.io/

COMPUTER SCIENCE & ENGINEERING
TEXAS A&M UNIVERSITY

Jyotikrishna Dass "JD"

HRBB 514A, Embedded Systems & Codesign Lab

dass dot jyotikrishna at tamu dot edu

Linear Models

Jyotikrishna Dass
Texas A&M University

$$x \xrightarrow{f} y$$

$$y = f(x)$$

linear in x (data)

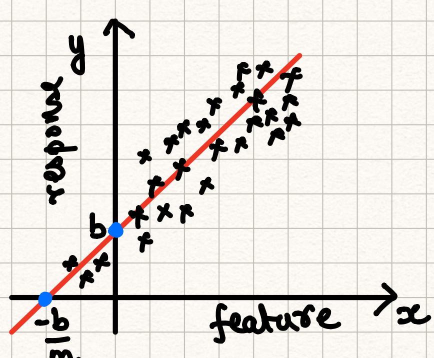
≡ Line or Hyperplane

≡ Polynomial of degree 1

$$y = b + mx$$

y: response variable (output)

x: predictor variable (input)



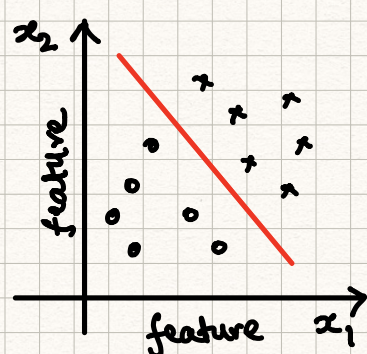
Linear Regression with 1D data

≡ Fit a line on response variable

$$y = \begin{bmatrix} 1 & x \end{bmatrix} \begin{bmatrix} b \\ m \end{bmatrix}$$

parameters of a line

$$\underset{1\times1}{y} = \underset{1\times2}{\vec{x}}\, \underset{2\times1}{\vec{w}} \equiv \mathbb{R}$$

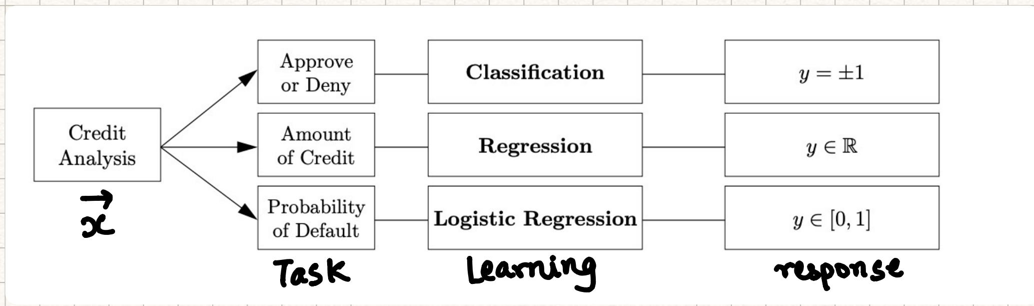


Linear Classification on 2D data

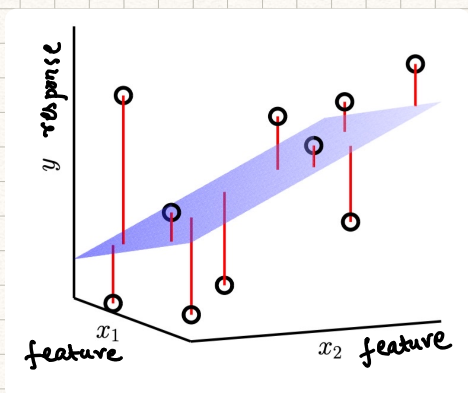
≡ Find a line that separates the data

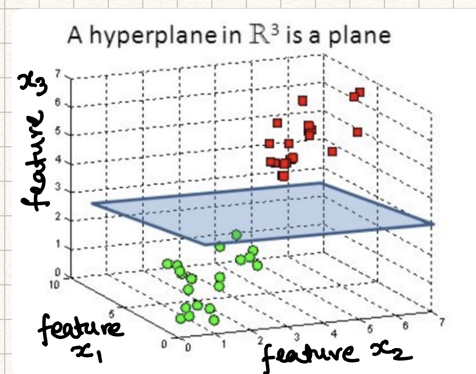
$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = 0$$

$$\{-1, +1\} \equiv \underset{1\times1}{y} = \text{sign}(\underset{1\times2}{\vec{x}}\, \underset{2\times1}{\vec{w}})$$



Source: Learning from Data





Linear Regression with 2D data (multiple features)

$\equiv$ Fit a plane

$$y = \begin{bmatrix} 1 & x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

Equation of a plane parameterized by w_0, w_1 and w_2

$$\underset{1\times1}{y} = \underset{1\times3}{\vec{x}}\ \underset{3\times1}{\vec{w}} \equiv \mathbb{R}$$

Linear Classification on 3D data

$\equiv$ Find a separating plane

$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

parameters of a plane

$$\{-1, +1\} \equiv \underset{1\times1}{y} = \mathrm{sign}\left(\underset{1\times3}{\vec{x}}\ \underset{3\times1}{\vec{w}}\right)$$

Linear Regression

Given : $\vec{x}_i$ i-th data point (predictor) $(1 \times d)$ vector

n number of data points, $i = 1 \ldots n$

d - dimensional feature

y_i real-valued response for i-th datapoint

Let,

$$X = \begin{bmatrix} 1 & \vec{x}_1 \\ 1 & \vec{x}_2 \\ \vdots & \vdots \\ 1 & \vec{x}_n \end{bmatrix}_{n \times (d+1)} \qquad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}_{n \times 1}$$

We wish to fit the model

$$y_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \ldots + w_d x_{id} + \epsilon_i$$

where, $w_0, w_1, w_2, \ldots, w_d$ = parameters of the hyperplane

ϵ_i = Gaussian noise
uncorrelated across measurements

$i = 1, \ldots, n$

→ System of linear equations n: equations
$(d+1)$: variables

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}_{n \times 1} \approx \begin{bmatrix} 1 & \vec{x}_1 \\ 1 & \vec{x}_2 \\ \vdots & \\ 1 & \vec{x}_n \end{bmatrix}_{n \times (d+1)} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}_{(d+1) \times 1}$$

$$Y_{n\times 1} \approx X_{n\times(d+1)} \, W_{(d+1)\times 1}$$

W → parameter / coefficient matrix

X → data / augmented matrix

Find best fitting W that approximates the above relationship between X and Y

|||

Solving a system of linear equations
with n : equations
$(d+1)$: unknown variables

CASE 1 : $n = d+1$

⇒ X is square matrix

⇒ is invertible (assuming datapoints are linearly independent)

⇒ $$W_{n\times 1} = X^{-1}_{n\times n} \, Y_{n\times 1}$$

⇒ exactly fits the data
⇒ poor generalization
⇒ need to regularize or subsample data into Training & Validation

CASE 2 : $n < d+1$

⇒ X is $\boxed{}_{n\times(d+1)}$

⇒ fewer datapoints than feature dimensions

⇒ less equations than unknown variables

⇒ Infinite number of solutions — under determined system

⇒ collect more data / feature selection and regularize

CASE 3: $n > d+1$

$\Rightarrow$ X is $\boxed{}_{n \times (d+1)}$

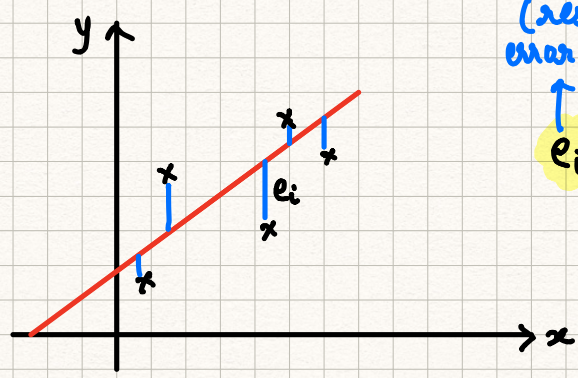
$\Rightarrow$ more datapoints than feature dimensions

$\Rightarrow$ more equations than unknown variables
overdetermined system

$\Rightarrow$ no solution exists

$\Rightarrow$ Look for W that is best fit

Best Fit Criterion : Least Squares



(residual)
error in prediction

$$e_i = \underbrace{y_i}_{\text{actual response}} - \underbrace{f(\vec{x}_i, W)}_{\text{predicted response}}$$

$$\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}$$

$$\underset{n \times 1}{e(W)} = \underset{n \times 1}{Y} - \underset{n \times (d+1)}{X} \underset{(d+1) \times 1}{W}$$

Optimization goal is to minimize $e_i \ \forall\, i = 1, .., n$

Mean Squared Error : Least Squares Objective

$$MSE(W) = \frac{1}{n}\sum_{i=1}^{n} e_i^2 = \frac{1}{n}\|e\|_2^2$$

$$\Rightarrow \quad J(W) = \frac{1}{n} e^T e \qquad \underset{n\times 1}{e} : e(W)$$

$$= \frac{1}{n}(Y - XW)^T (Y - XW)$$

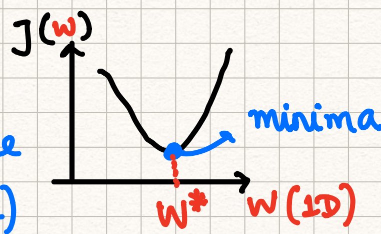
Homework

$$= \frac{1}{n}\left(Y^T Y - 2W^T X^T Y + W^T X^T X W\right)$$

$$\min_W J(W)$$

$$= \min_W \frac{1}{n} e^T e$$

Convex Surface (parabola)



$$= \min_W \frac{1}{n}(Y - XW)^T (Y - XW)$$

$$= \min_W \frac{1}{n}\left(Y^T Y - 2W^T X^T Y + W^T X^T X W\right)$$

Analytical Solution to Least Squares

$$\nabla J(W)\Big|_{W^*} = 0_{(d+1)\times 1}$$

W^* → optimum that minimizes MSE(W)

$$\frac{1}{n}\left(\nabla Y^T Y - 2\nabla W^T X^T Y + \nabla W^T X^T X W\right)_{(d+1)\times 1}$$

① $\nabla_w (W^T b) = b$

② $\nabla_w (W^T A W) = (A + A^T) W$

$$\frac{1}{n}\left(0 - 2X^T Y + 2X^T X W\right)\Big|_{W^*} = 0$$

$$\Rightarrow \quad X^T X W^* = X^T Y$$ ~ Normal Equation

$$\Rightarrow \quad \boxed{\underset{(d+1)\times 1}{W^*} = \underbrace{(X^T X)^{-1} X^T}_{\text{pseudo inverse of } X} Y}$$

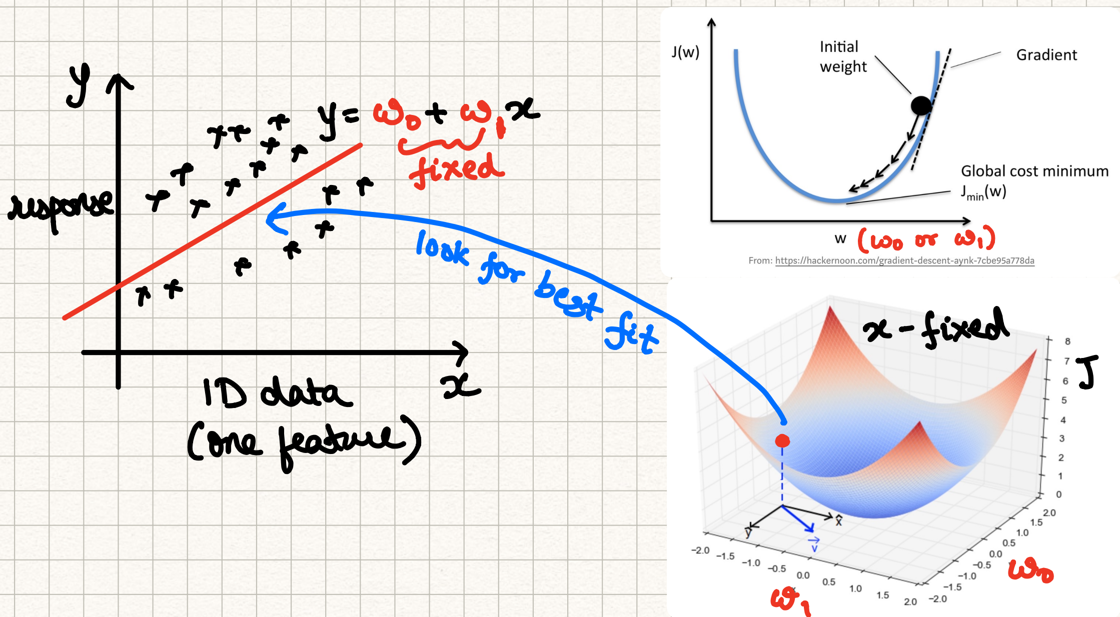
when $(X^T X)^{-1}$ exists (well-conditioned)

one-step solution

pseudo inverse of X

Gradient Descent

iterative steps to solve optimization problem



Bowl-shaped

for $\quad x_i \equiv$ 1D data $(d=1)$

Recall,
$$J(W) = \frac{1}{n}\sum_{i=1}^{n}\Big(y_i - \underbrace{f(x_i, W)}_{[1 \;\; x_i]_{1\times 2}\begin{bmatrix} w_0 \\ w_1 \end{bmatrix}_{2\times 1}}\Big)^2$$

$$J(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - w_0 - w_1 x_i\right)^2$$

$$= \frac{1}{n}\sum_{i=1}^{n} y_i^2 + w_0^2 + w_1^2 x_i^2 - 2 w_0 y_i + 2 w_0 w_1 x_i - 2 w_1 x_i y_i$$

$\equiv$ Quadratic function in w_0

$\equiv$ Quadratic function in w_1

$\equiv$ Quadratic surface in (w_0, w_1)

In Analytic Solution, recall

$$\nabla_{W} J(W)_{2\times 1} = 0_{2\times 1} \qquad W = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

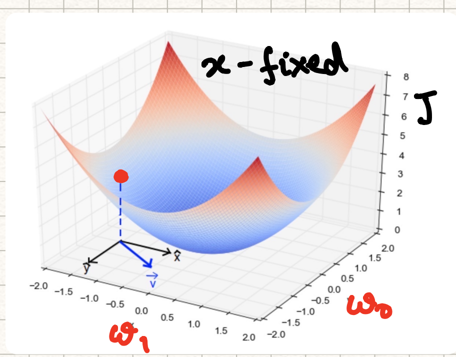
Alternatively, iteratively set

$$\nabla_{W} J(W)_{2\times 1} \rightarrow 0_{2\times 1}$$

$$\equiv \begin{bmatrix} \dfrac{\partial J(\omega_0)}{\partial \omega_0} \\[2ex] \dfrac{\partial J(\omega_1)}{\partial \omega_1} \end{bmatrix}_{2\times 1} \rightarrow \begin{bmatrix} 0 \\ 0 \end{bmatrix}_{2\times 1}$$

$\equiv$ Direction of Gradient is in the Direction of fastest increase in function value

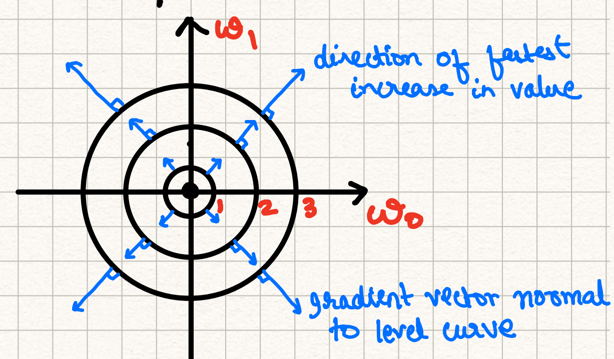
$\equiv$ steepest ascent | climb



Elliptic Paraboloid
(bowl-shaped surface)

eg: $J(\omega_0, \omega_1) = \omega_0^2 + \omega_1^2$

Gradient vector : $\nabla_{W} J = \begin{bmatrix} 2\omega_0 \\ 2\omega_1 \end{bmatrix}$



Level Curves
(equipotential levels)

eg: $\omega_0^2 + \omega_1^2 = 1$

$\omega_0^2 + \omega_1^2 = 4$

$\omega_0^2 + \omega_1^2 = 9$

If we are at any random $(w_0, w_1, J(w_0, w_1))$ on the bowl surface, we need to roll down into the deepest point in the valley i.e. minima where, $J(w_0, w_1)$ is minimum.

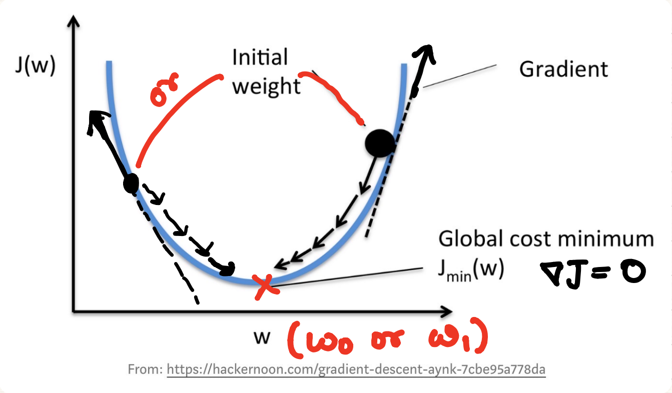
≡ This rolling down needs to be in the direction of steepest descent to save time

≡ In level-curves, we need to move from outer levels to innermost levels until we reach the center with minimum value of J.

≡ Follow direction of fastest decrease

≡ Follow Negative Gradient direction until you reach zero gradient

i.e. $\nabla_w J(W) \to 0$

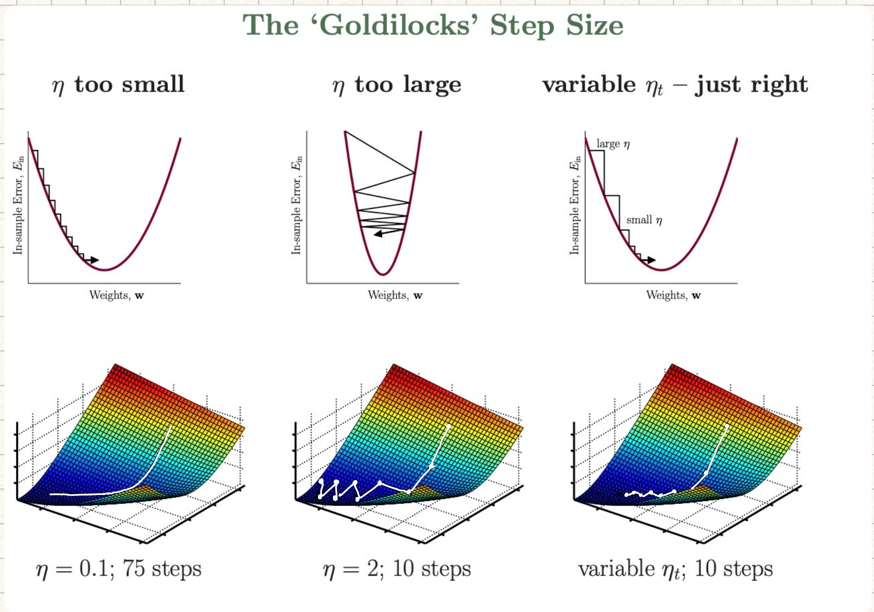


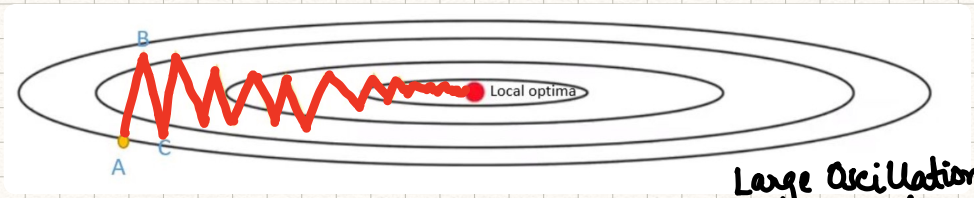
Iteratively,

$$W^{(t+1)} = W^{(t)} - \eta \nabla J(W^{(t)})$$

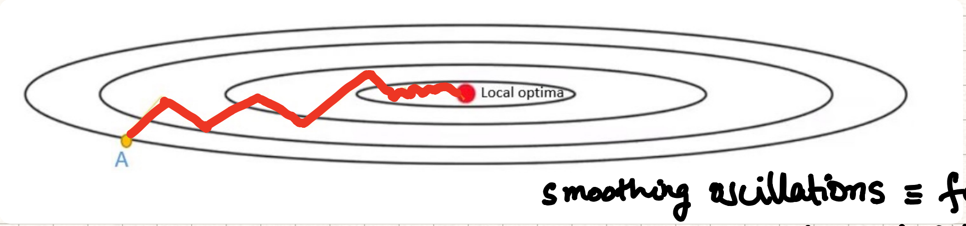
where, η : learning rate / step size

i.e. take a step size of η in the direction of negative gradient





Large Oscillations
= more iterations = slow convergence



smoothing oscillations = fast convergence
= less iterations

Recall, linear regression cost function for 1D data

$$J(w_0, w_1) = \frac{1}{2}\left(\frac{1}{n}\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i)^2\right)$$

$$= \frac{1}{2n}\sum_{i=1}^{n} y_i^2 + w_0^2 + w_1^2 x_i^2 - 2w_0 y_i + 2w_0 w_1 x_i - 2w_1 x_i y_i$$

$$\nabla J(w_0, w_1) = \begin{bmatrix} \frac{\partial J(w_0)}{\partial w_0} \\ \frac{\partial J(w_1)}{\partial w_1} \end{bmatrix}_{2\times 1}$$

$$J(w_0) = \frac{1}{2n}\sum_{i=1}^{n} w_0^2 - 2w_0 y_i + 2w_0 w_1 x_i + \text{constant}$$

$$J(w_1) = \frac{1}{2n}\sum_{i=1}^{n} w_1^2 x_i^2 + 2w_0 w_1 x_i - 2w_1 x_i y_i + \text{constant}$$

$$\frac{\partial J(w_0)}{\partial w_0} = \frac{1}{2n}\sum_{i=1}^{n} 2w_0 - 2y_i + 2w_1 x_i$$

$$= \frac{1}{n}\sum_{i=1}^{n} \begin{bmatrix} 1 & x_i \end{bmatrix}\begin{bmatrix} w_0 \\ w_1 \end{bmatrix} - y_i$$

$$= \frac{1}{n}\sum_{i=1}^{n} (X_i W - Y_i)_{1\times 1}$$

(scalar)

$$\frac{\partial J(w_1)}{\partial w_1} = \frac{1}{2n}\sum_{i=1}^{n} 2w_1 x_i^2 + 2w_0 x_i - 2x_i y_i$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(\left(w_1 x_i + w_0 1\right) - y_i\right) x_i$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(\underset{1\times 2}{X_i}\,\underset{2\times 1}{W} - \underset{1\times 1}{Y_i}\right)\underset{1\times 1}{x_i}$$

$$\underset{2\times 1}{\nabla J(W)} = \frac{1}{n}\sum_{i=1}^{n}\begin{bmatrix}(X_i W - Y_i)\,1 \\ (X_i W - Y_i)\,x_i\end{bmatrix}_{2\times 1}$$

$$= \frac{1}{n}\sum_{i=1}^{n}(X_i W - Y_i)\begin{bmatrix}1 \\ x_i\end{bmatrix}_{2\times 1}$$

$$= \frac{1}{n}\sum_{i=1}^{n}(\underbrace{X_i W - Y_i}_{\text{scalar}})\underbrace{X_i^T}_{2\times 1}$$

$$\nabla J(W) = \frac{1}{n}\underset{2\times n}{X^T}\left(\underset{n\times 2}{X}\,\underset{2\times 1}{W} - \underset{n\times 1}{Y}\right)_{n\times 1}$$

Parameter update step for Linear Regression (multivariate)

$$\underset{(d+1)\times 1}{W^{(t+1)}} = W^{(t)} - \eta\,\frac{1}{n}X^T(XW - Y)$$

step size.

#training samples

Gradient Descent Algorithm (batch)

1: Initialize at step $t=0$ to $W^{(0)}$

2: for $t = 0, 1, 2, \dots$ do

3: Compute the gradient

$$g^{(t)} = \nabla J(W^{(t)})$$

takes all training samples for each iteration

4: Update the weights

$$W^{(t+1)} = W^{(t)} - \eta\, g^{(t)}$$

5: Iterate until it is time to stop

6: end for

7: Return the final weights $W^{(t=t^*)}$

Regularized Linear Regression

- To avoid overfitting training data
- To avoid fitting the noisy samples
- To generalize well on unseen test data
- Aim is to be guarded / defensive in how we predict
- Keep W small by constraints

Recall, $MSE(W) = \frac{1}{2n} \sum_{i=1}^{n} e_i(w)^2 = \frac{1}{2n}(Y - XW)^T (Y - XW)$

$$\min_{W} \; MSE(W)$$

subject to $\|W\|_1 \leq C$

for some $C > 0$

LASSO Regression

$$\min_{W} \; MSE(W)$$

subject to $W^T W \leq C$

for some $C > 0$

Ridge Regression

$$\min_{W} \; \underbrace{MSE(W) + \lambda \|W\|_1}_{J(W)} \qquad \lambda > 0$$

$$\min_{W} \; \underbrace{MSE(W) + \frac{\lambda}{2n} \|W\|_2^2}_{J(W)} \qquad \lambda > 0$$

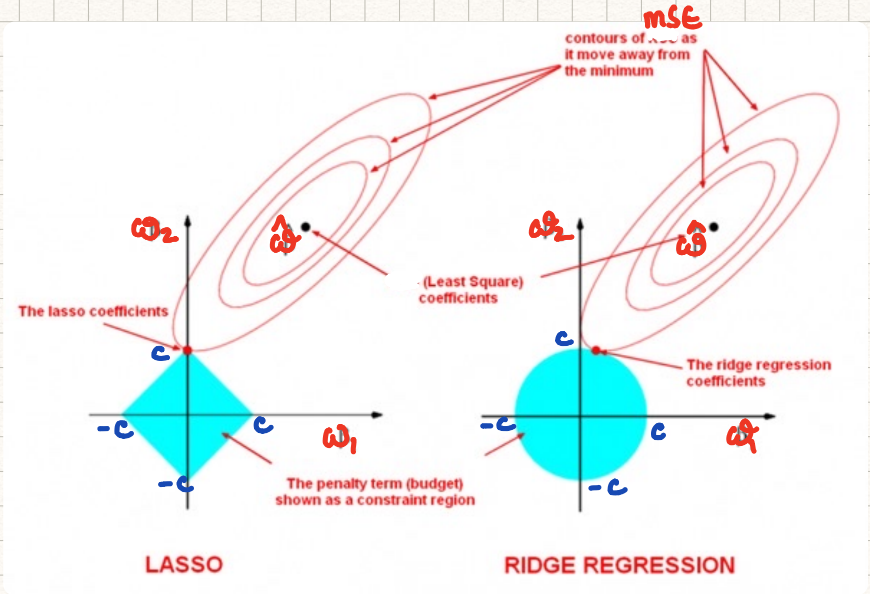
$\lambda \to 0$

$\Rightarrow$ Relax constraints on W

$\Rightarrow$ constrained region grows & hits ellipse center

$\Rightarrow$ LASSO, Ridge $\longrightarrow$ Linear Regression (Least Squares Fit)

$\Rightarrow$ Model complexity increases (overfitting)



- Can lead to 0 parameters
- Feature selection

- shrinks the parameters
- reduces model complexity

Solving Ridge Regression

$$J(W) = \frac{1}{2n}(Y - XW)^T(Y - XW) + \frac{\lambda}{2n}W^TW$$

$$= \frac{1}{2n}\left(Y^TY - 2W^TX^TY + W^TX^TXW\right) + \frac{\lambda}{2n}W^TW$$

$$\frac{\partial J(W)}{\partial W} = \frac{1}{2n}\left(0 - 2X^TY + 2X^TXW\right) + \frac{2\lambda}{2n}W$$

$$= \frac{1}{n}X^T(XW - Y) + \frac{\lambda}{n}W$$

Least Squares Fit with Regression

$$\frac{\partial J(W)}{\partial W} = 0 \quad \text{at} \quad W^*$$

$$\Rightarrow \left(\frac{X^T X}{n} + \frac{\lambda I}{n} \right) W^* = \frac{1}{n} X^T Y$$

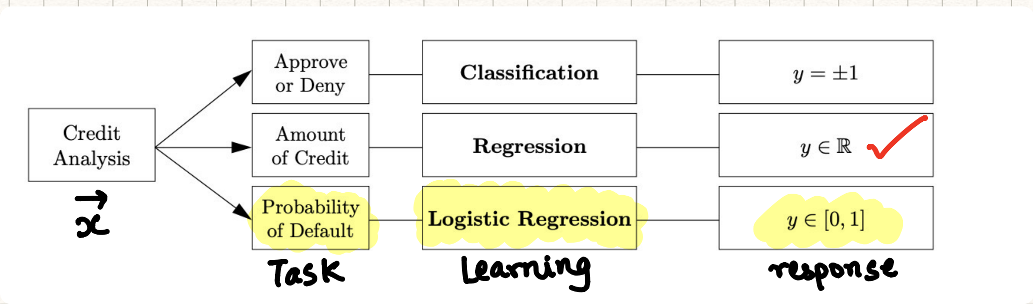
$$\Rightarrow (X^T X + \lambda I) W^* = X^T Y$$

$$\Rightarrow \quad W^* = (X^T X + \underbrace{\lambda I}_{})^{-1} X^T Y$$

extra positive term that makes $(XX^T + \lambda I)$ invertible

- eigen values of $X^T X \geq 0$
- eigen values of $(X^T X + \lambda I) > 0$ as $\lambda > 0$

$\Rightarrow$ Inverse exists ALWAYS



Source: Learning from Data

Logistic Regression

Goal is to model Probability of y belonging to an event or not given the data $\vec{x}$

$$x \xrightarrow{f_w} y \in [0,1]$$

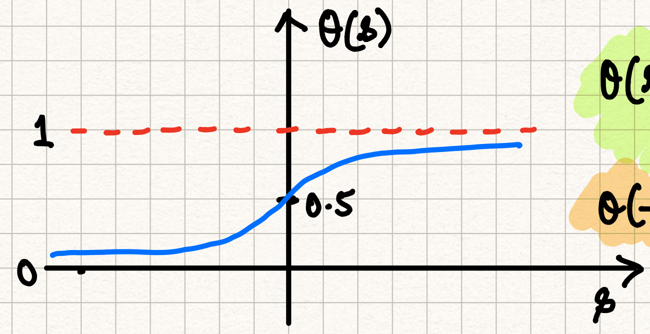
discrete values

eg: Given the patient health data, what is the likelihood of heart attack over the next year.

$$f_w(\vec{x}) = \theta\left(\sum_{k=0}^{d} w_k x_k\right) = \theta\left(W^T \vec{x}\right)$$

where, $\vec{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}_{(d+1)\times 1}$ $\quad W = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}_{(d+1)\times 1}$

Logistic function (sigmoid)



$$\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$$

$$\theta(-s) = \frac{1}{1+e^s} = 1-\theta(s)$$

$$\theta'(s) = \frac{\partial \theta(s)}{\partial s} = +\frac{e^{-s}}{(1+e^{-s})^2} = \frac{1}{1+e^{-s}} \times \frac{e^{-s}}{1+e^{-s}} = \theta(s)\left(1-\theta(s)\right)$$

Dataset : $(\vec{x}_1, y_1), \ldots . (\vec{x}_n, y_n)$

$\vec{x}_i$ patient i's health information $(d+1) \times 1$ vector

$y_i =$ did they have a heart attack or not

Let us assume that

$$\begin{cases} P(y=1 \mid \vec{x}, W) = f_W(\vec{x}) \\ P(y=0 \mid \vec{x}, W) = 1 - f_W(\vec{x}) \end{cases}$$

compact notation

$$p(y \mid \vec{x}, W) = \left(f_W(\vec{x})\right)^{y} \left(1 - f_W(\vec{x})\right)^{1-y}$$

Assume n training examples generated independently,

$L(W) = p(Y \mid X, W)$ is likelihood of parameters

$$= \prod_{i=1}^{n} p(y_i \mid \vec{x}_i, W)$$

$$= \prod_{i=1}^{n} \left(f_W(\vec{x}_i)\right)^{y_i} \left(1 - f_W(\vec{x}_i)\right)^{1-y_i}$$

Goal is to Maximize this likelihood

$\equiv$ maximize the log likelihood

$$\ell(W) = \log \mathcal{L}(W)$$

$$\max_{W} \sum_{i=1}^{n} y_i \log\left(f_W(\vec{x}_i)\right) + (1-y_i) \log\left(1 - f_W(\vec{x}_i)\right)$$

$$\equiv \min_{W} -\ell(W)$$

cross-entropy loss

Optimize using Gradient Ascent

where,

$$W^{(t+1)} = W^{(t)} + \eta \nabla_W \ell(W^{(t)})$$

positive as we are maximizing

Let us find update for 1 training sample $(\vec{x}_i, y_i)$

$$\frac{\partial \ell_i(W_k)}{\partial W_k} = \left(y_i \frac{1}{f_W(\vec{x}_i)} - (1-y_i) \frac{1}{1 - f_W(\vec{x}_i)} \right) \frac{\partial f_W(\vec{x}_i)}{\partial W_k}$$

Recall, $f_W(\vec{x}_i) = \theta(W^T \vec{x}_i)$, where, $\theta(s)$ is sigmoid/logistic function $= \frac{1}{1+e^{-s}}$

and $\theta'(s) = \theta(s)\left(1 - \theta(s)\right)$

$$= \left(y_i \frac{1}{f_W(\vec{x}_i)} - (1-y_i) \frac{1}{1 - f_W(\vec{x}_i)} \right) f_W(\vec{x}_i)\left(1 - f_W(\vec{x}_i)\right) \frac{\partial (W^T \vec{x}_i)}{\partial W_k}$$

$$= \left(y_i \left(1 - f_w(\vec{x}_i)\right) - (1 - y_i) f_w(\vec{x}_i) \right) \vec{x}_{ik}$$

$$= \left(y_i - f_w(\vec{x}_i) \right) \vec{x}_{ik}$$

$$W_k^{(t+1)} = W_k^{(t)} + \eta \frac{\partial L_i(W^{(t)})}{\partial W_k^{(t)}}$$

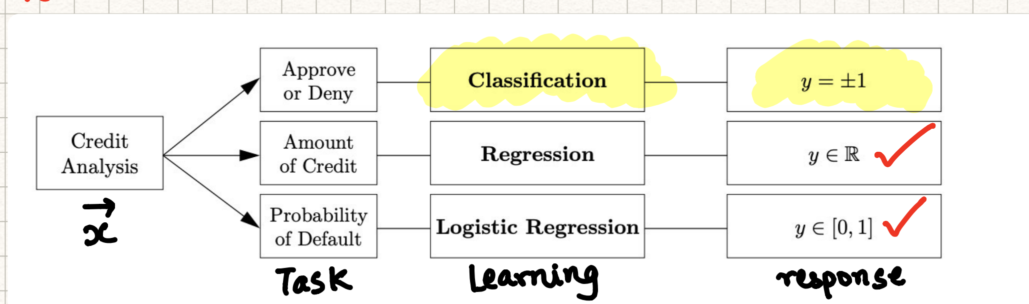
$k = 1 \ldots (d+1)$
$k \neq 0$

contribution from single sample $\vec{x}_i \equiv L_i(\cdot)$

$\equiv$ Stochastic Gradient Ascent

$$W_k^{(t+1)} = W_k^{(t)} + \eta \left(y_i - \frac{1}{(1 + e^{-W^T \vec{x}_i})} \right) \vec{x}_{ik}$$

$k = 1, \ldots, (d+1)$
$k \neq 0$

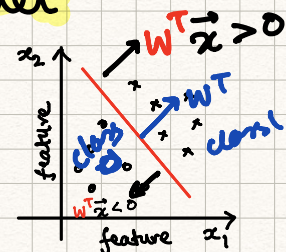


Source: Learning from Data

Perceptron Classifier Model

$x \xrightarrow{f} y \in \{0,1\}$

binary class

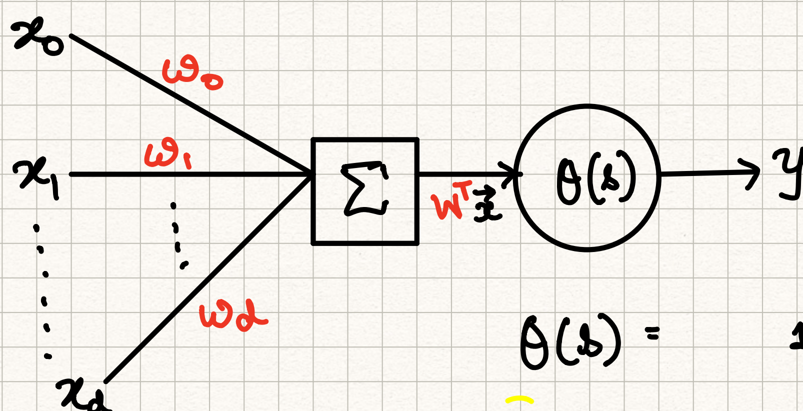


Linear Classification on 2D data

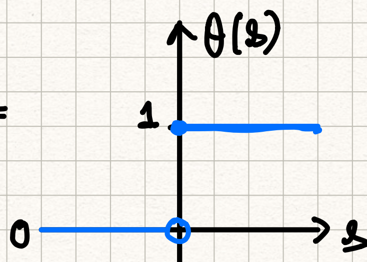
= Find a line that separates the data

$$f_W(\vec{x}) = \theta(W^T\vec{x})$$

where, $\vec{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}_{(d+1)\times 1}$ $W = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}_{(d+1)\times 1}$



$\theta(s) =$



Rough model of how human brain works

$$= step(s) = \begin{cases} 1 & \text{if } s \geq 0 \\ 0 & \text{if } s < 0 \end{cases}$$

Update Rule:

$$W_k^{(t+1)} = W_k^{(t)} + \eta\underbrace{\left(y_i - f_W(\vec{x}_i)\right)\vec{x}_{ik}}$$

$k \neq 0$

$k = 1, \ldots, (d+1)$

if mistake on (+1) example

$W_k^{t+1} = W_k^t + \eta(1-0)\vec{x}_{ik}$

Online : $W_k^{t+1} = W_k^t + \eta\vec{x}_{ik}$

if mistake on (0) example

$W_k^{t+1} = W_k^t + \eta(0-1)\vec{x}_{ik}$

$W_k^{t+1} = W_k^t - \eta\vec{x}_{ik}$

Lecture 2: The SVM classifier

C19 Machine Learning Hilary 2015 A. Zisserman

- Review of linear classifiers
    - Linear separability
    - Perceptron

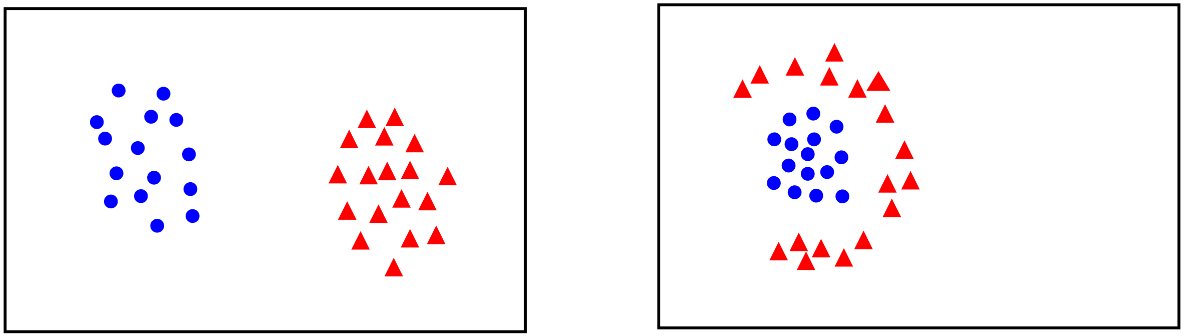
- Support Vector Machine (SVM) classifier
    - Wide margin
    - Cost function
    - Slack variables
    - Loss functions revisited
    - Optimization

Binary Classification

Given training data $(\mathbf{x}_i, y_i)$ for $i = 1 \ldots N$, with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$, learn a classifier $f(\mathbf{x})$ such that

$$f(\mathbf{x}_i) \begin{cases} \geq 0 & y_i = +1 \\ < 0 & y_i = -1 \end{cases}$$

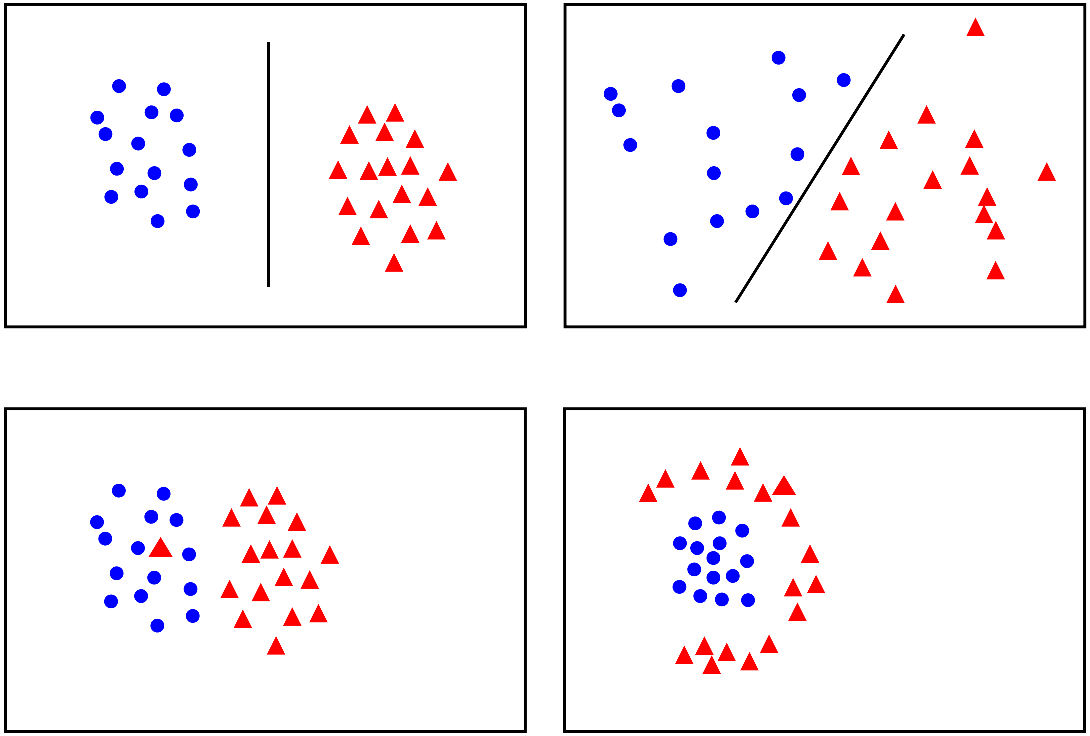
i.e. $y_i f(\mathbf{x}_i) > 0$ for a correct classification.



Linear separability

linearly separable

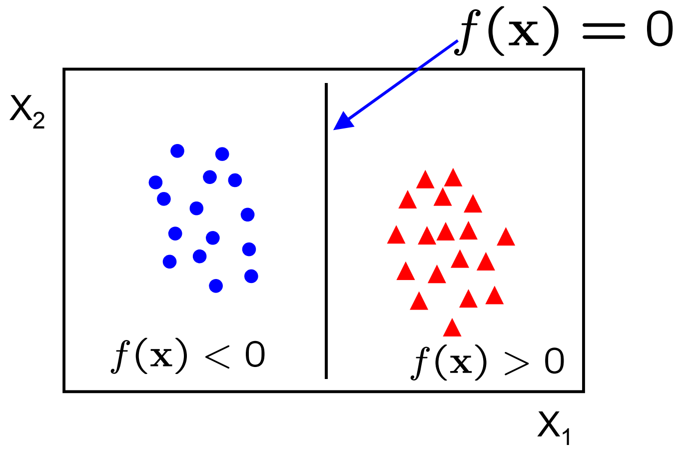
not linearly separable



Linear classifiers

A linear classifier has the form

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

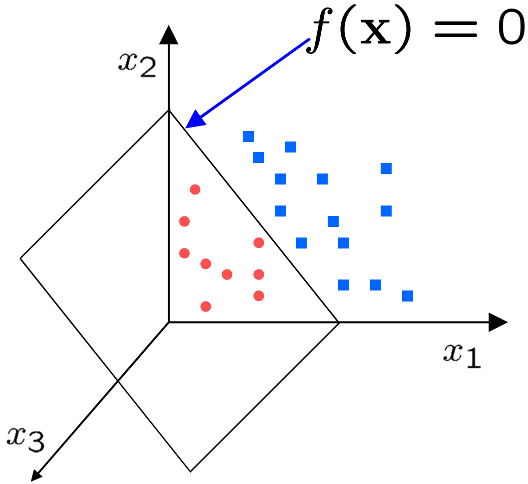


- in 2D the discriminant is a line
- $\mathbf{w}$ is the normal to the line, and b the bias
- $\mathbf{w}$ is known as the weight vector

Linear classifiers

A linear classifier has the form

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$



- in 3D the discriminant is a plane, and in nD it is a hyperplane

For a K-NN classifier it was necessary to `carry' the training data

For a linear classifier, the training data is used to learn **w** and then discarded

Only **w** is needed for classifying new data

The Perceptron Classifier

Given linearly separable data $\mathbf{x}_i$ labelled into two categories $y_i = \{-1,1\}$, find a weight vector $\mathbf{w}$ such that the discriminant function

$$f(\mathbf{x}_i) = \mathbf{w}^\top \mathbf{x}_i + b$$

separates the categories for i = 1, .., N

- how can we find this separating hyperplane ?

The Perceptron Algorithm

Write classifier as $f(\mathbf{x}_i) = \tilde{\mathbf{w}}^\top \tilde{\mathbf{x}}_i + w_0 = \mathbf{w}^\top \mathbf{x}_i$

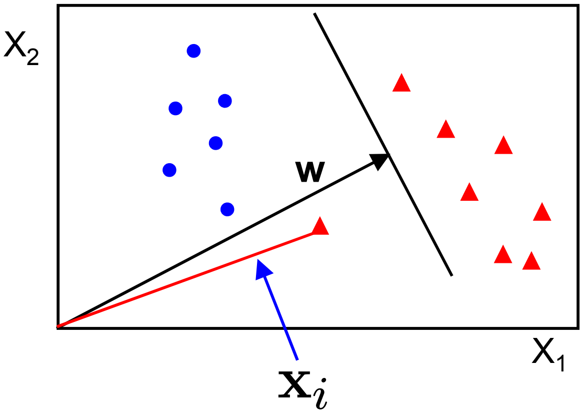
$$\text{where } \mathbf{w} = (\tilde{\mathbf{w}}, w_0), \mathbf{x}_i = (\tilde{\mathbf{x}}_i, 1)$$

- Initialize $\mathbf{w} = 0$
- Cycle though the data points $\{ \mathbf{x}_i, y_i \}$
  - if $\mathbf{x}_i$ is misclassified then $\mathbf{w} \leftarrow \mathbf{w} + \alpha \, \mathrm{sign}(f(\mathbf{x}_i)) \, \mathbf{x}_i$
- Until all the data is correctly classified

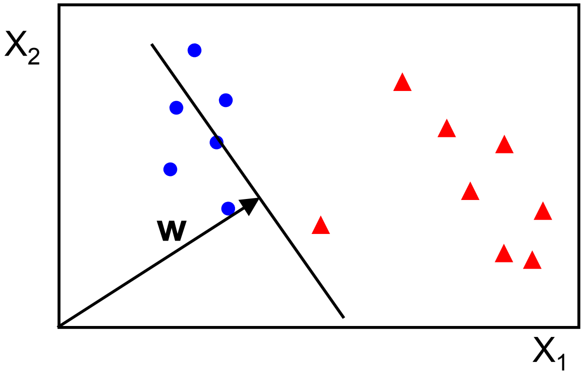
For example in 2D

- Initialize **w** = 0
- Cycle though the data points { $\mathbf{x}_i$, y_i }
  - if $\mathbf{x}_i$ is misclassified then $\mathbf{w} \leftarrow \mathbf{w} + \alpha \, \text{sign}(f(\mathbf{x}_i)) \, \mathbf{x}_i$
- Until all the data is correctly classified

before update



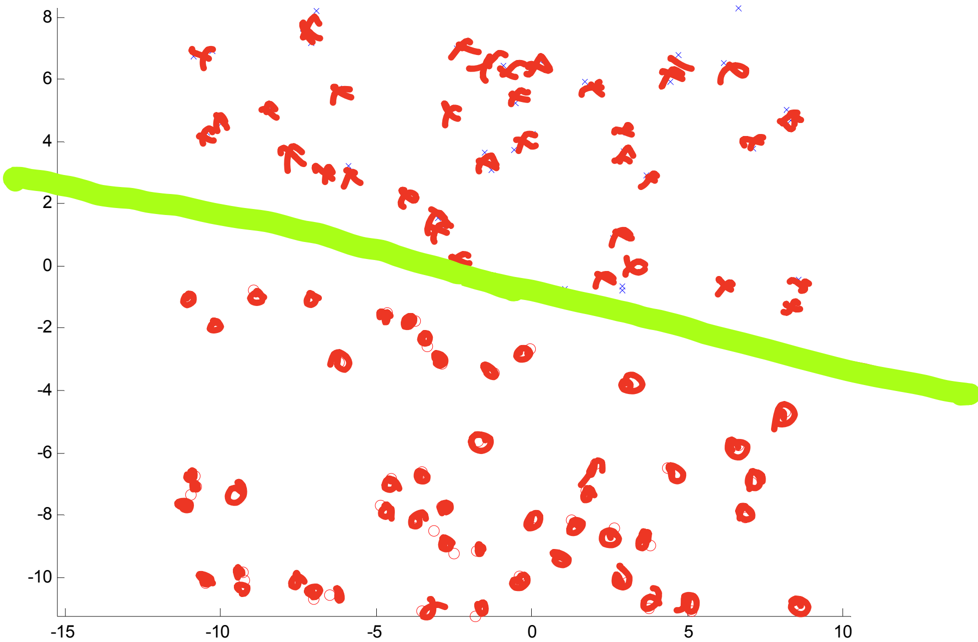
after update



$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \, \mathbf{x}_i$$

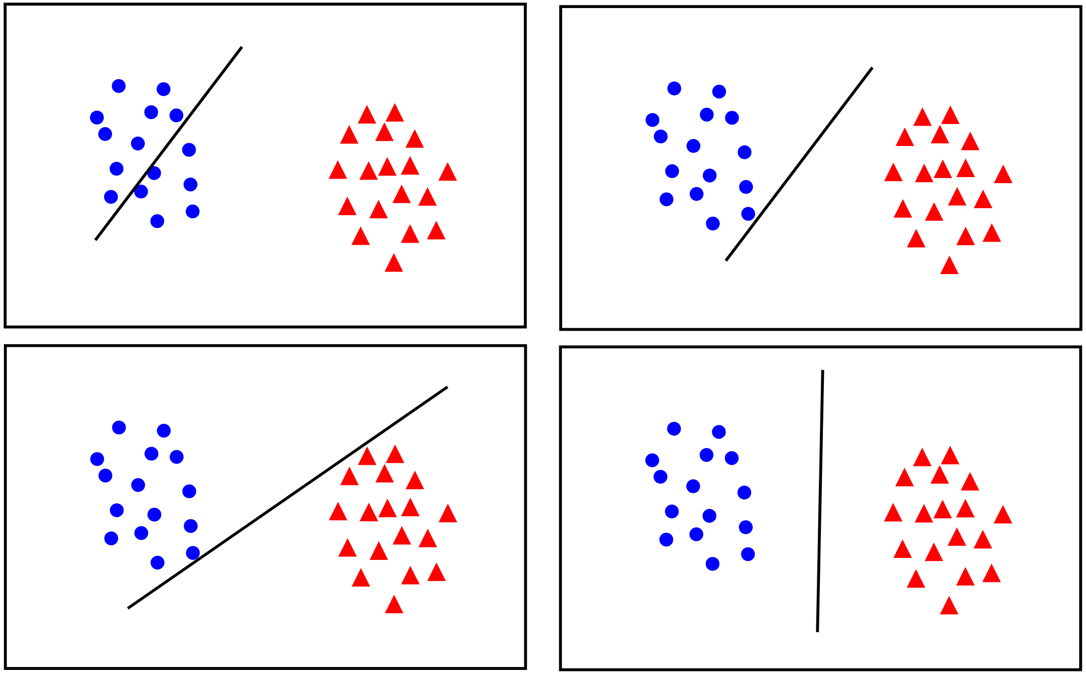
NB after convergence $\mathbf{w} = \sum_i^N \alpha_i \mathbf{x}_i$

Perceptron example



- if the data is linearly separable, then the algorithm will converge
- convergence can be slow …
- separating line close to training data
- we would prefer a larger margin for generalization

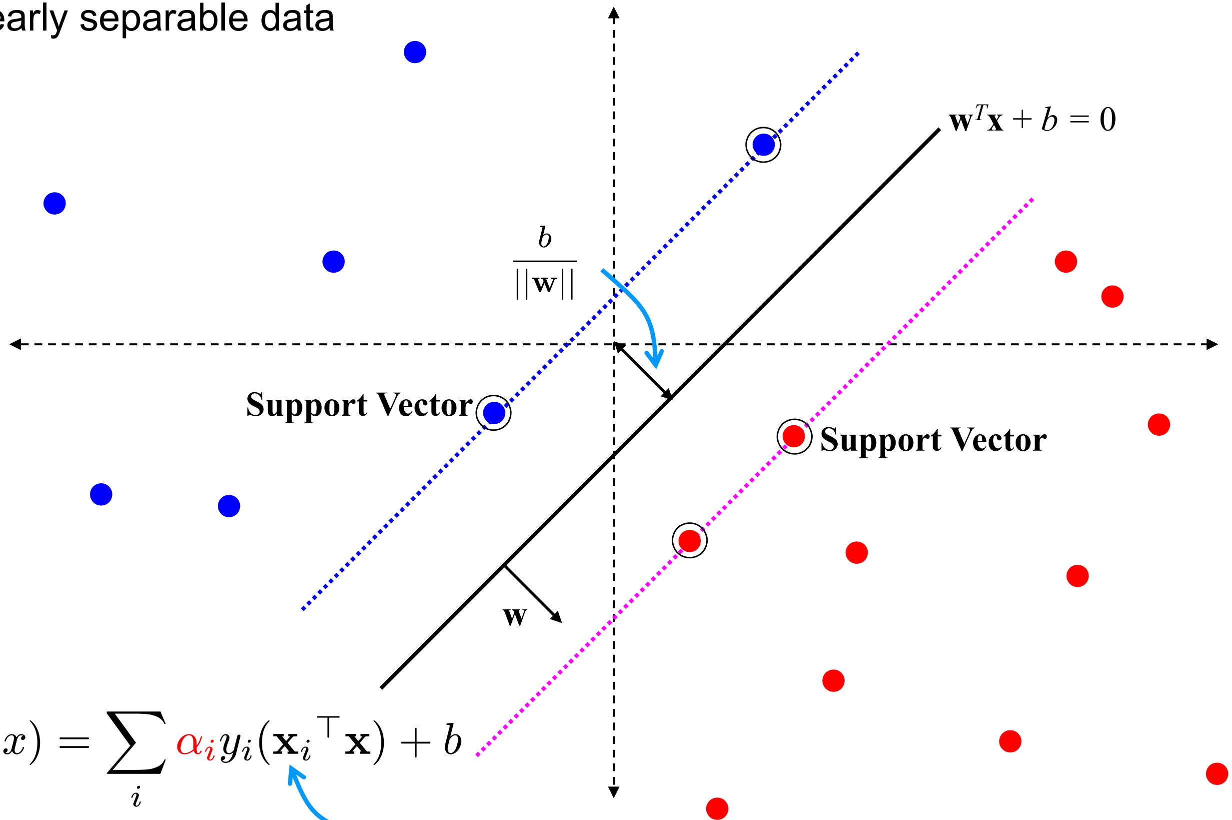
What is the best w?



- maximum margin solution: most stable under perturbations of the inputs

Support Vector Machine

linearly separable data



$$f(x) = \sum_i \alpha_i y_i ({\mathbf{x}_i}^\top \mathbf{x}) + b$$

support vectors

SVM – sketch derivation

- Since $\mathbf{w}^\top \mathbf{x} + b = 0$ and $c(\mathbf{w}^\top \mathbf{x} + b) = 0$ define the same plane, we have the freedom to choose the normalization of $\mathbf{w}$

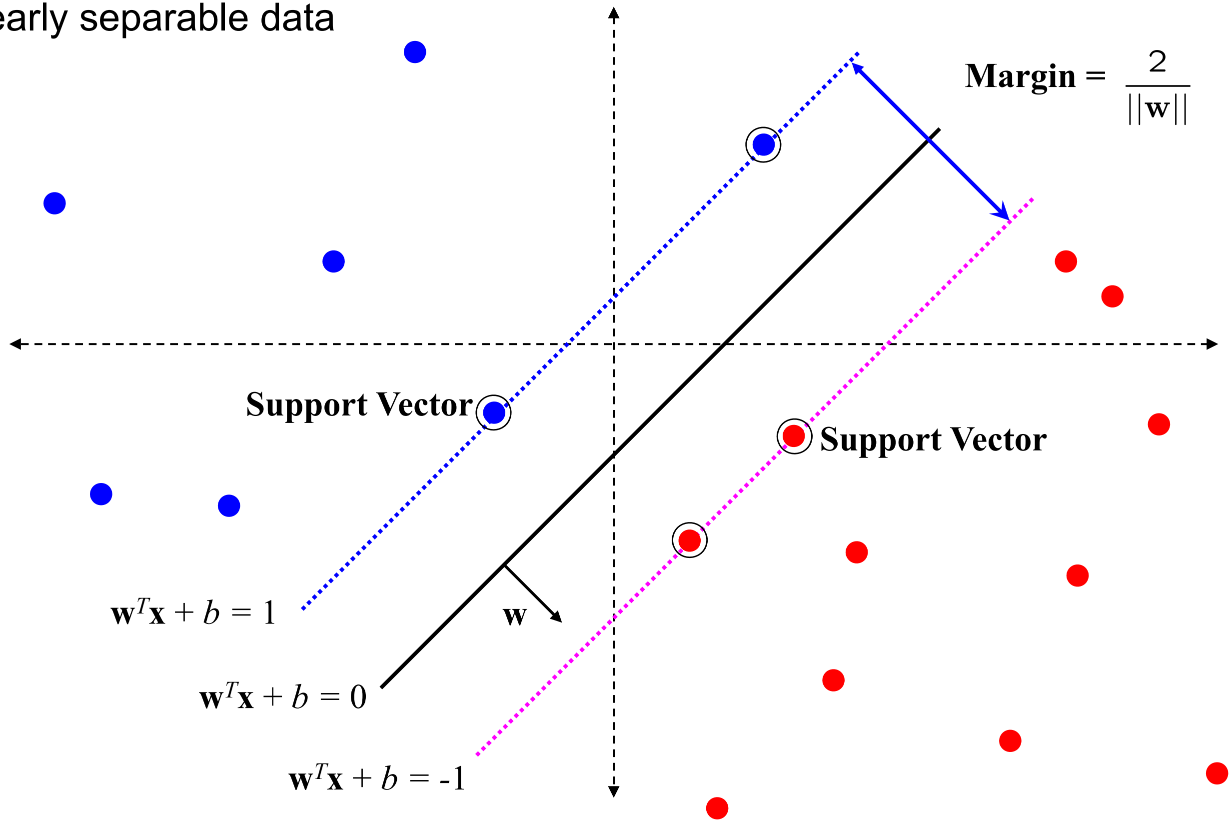
- Choose normalization such that $\mathbf{w}^\top \mathbf{x}_+ + b = +1$ and $\mathbf{w}^\top \mathbf{x}_- + b = -1$ for the positive and negative support vectors respectively

- Then the margin is given by

$$\frac{\mathbf{w}}{||\mathbf{w}||} \cdot \left(\mathbf{x}_+ - \mathbf{x}_-\right) = \frac{\mathbf{w}^\top \left(\mathbf{x}_+ - \mathbf{x}_-\right)}{||\mathbf{w}||} = \frac{2}{||\mathbf{w}||}$$

Support Vector Machine

linearly separable data



SVM – Optimization

- Learning the SVM can be formulated as an optimization:

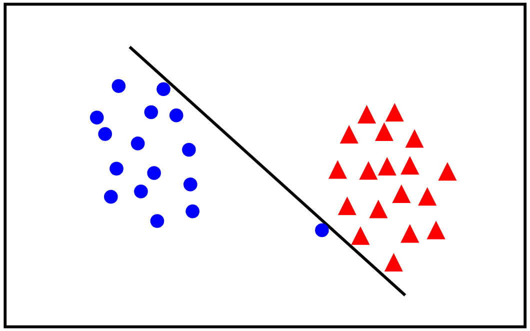
$$\max_{\mathbf{w}} \frac{2}{||\mathbf{w}||} \text{ subject to } \mathbf{w}^\top \mathbf{x}_i + b \begin{array}{ll} \geq 1 & \text{if } y_i = +1 \\ \leq -1 & \text{if } y_i = -1 \end{array} \text{ for } i = 1 \dots N$$

- Or equivalently

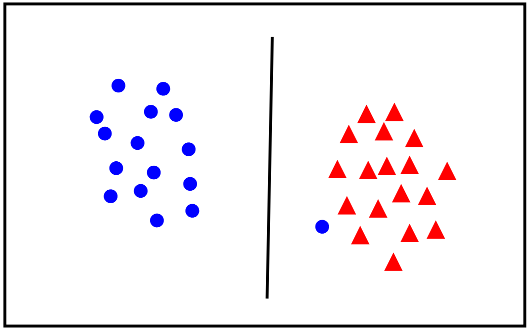
$$\min_{\mathbf{w}} ||\mathbf{w}||^2 \text{ subject to } y_i \left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1 \text{ for } i = 1 \dots N$$

- This is a quadratic optimization problem subject to linear constraints and there is a unique minimum

Linear separability again: What is the best w?



• the points can be linearly separated but there is a very narrow margin



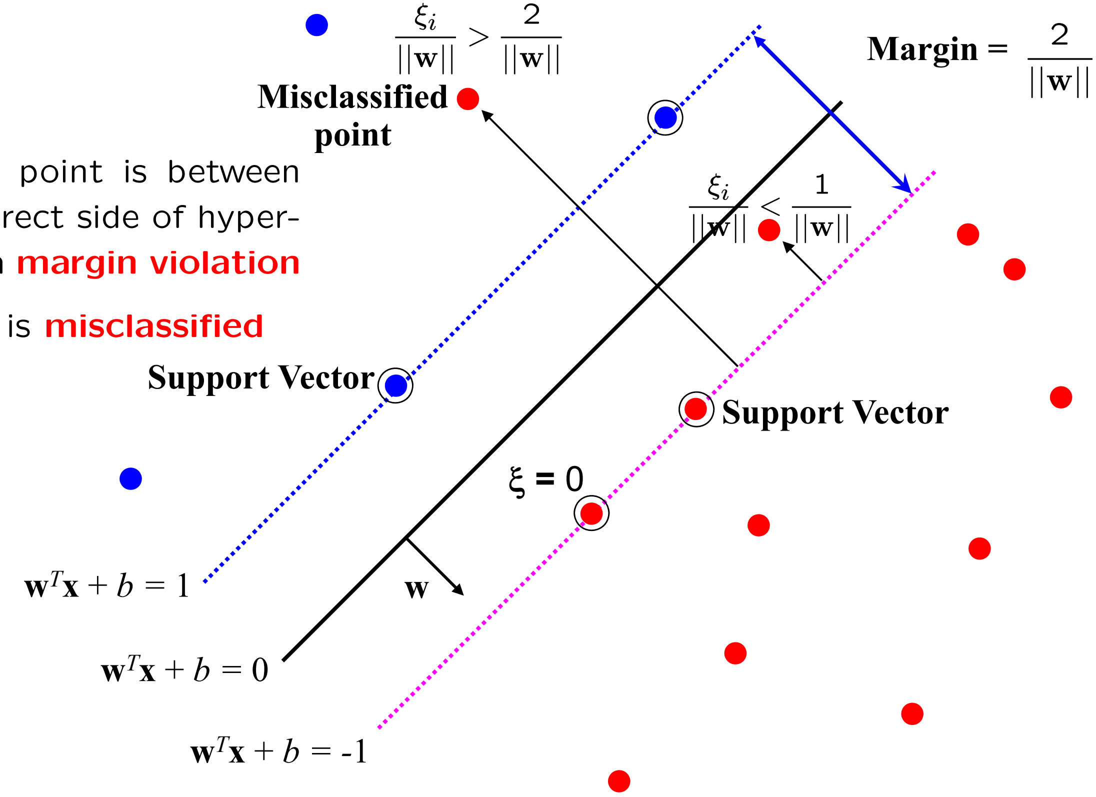
• but possibly the large margin solution is better, even though one constraint is violated

In general there is a trade off between the margin and the number of mistakes on the training data

Introduce "slack" variables

$$\xi_i \geq 0$$

- for $0 < \xi \leq 1$ point is between margin and correct side of hyper-plane. This is a **margin violation**
- for $\xi > 1$ point is **misclassified**



“Soft” margin solution

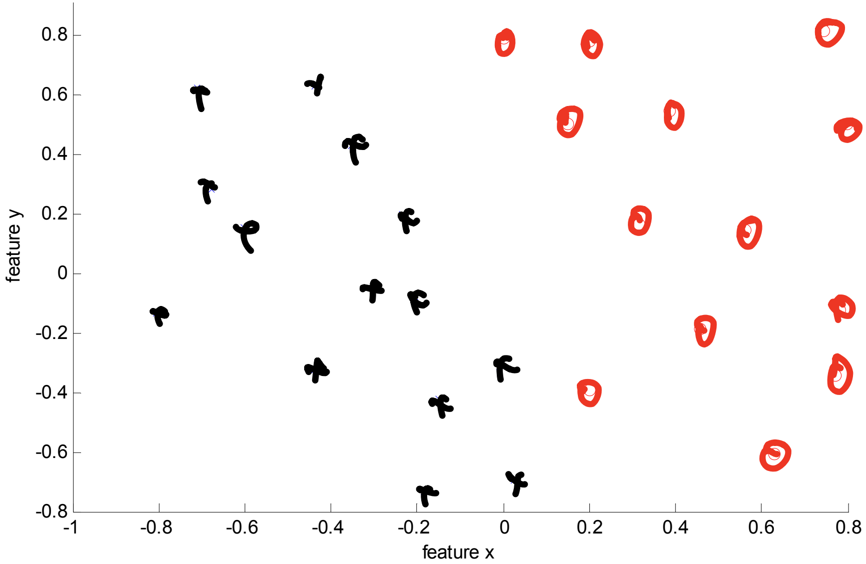
The optimization problem becomes

$$\min_{\mathbf{w}\in\mathbb{R}^d, \xi_i \in \mathbb{R}^+} ||\mathbf{w}||^2 + C \sum_i^N \xi_i$$

subject to

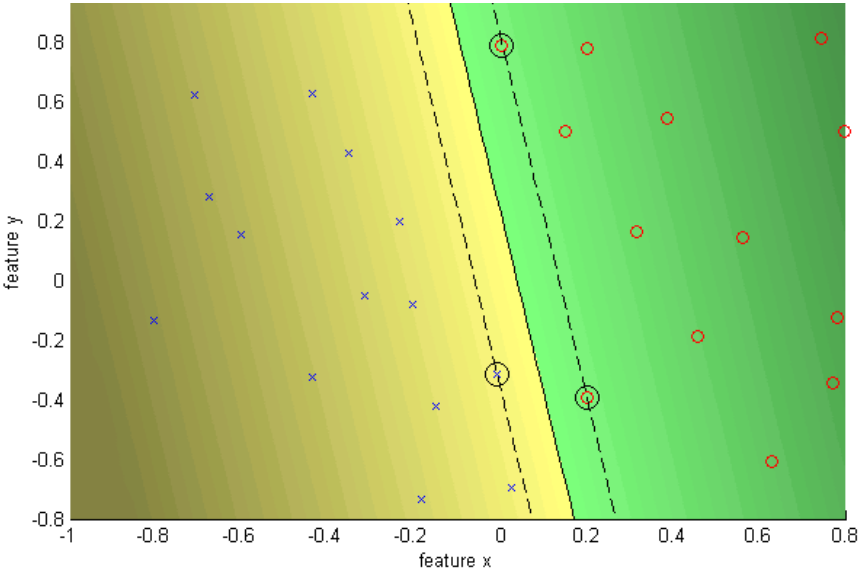
$$y_i \left(\mathbf{w}^\top \mathbf{x}_i + b \right) \geq 1 - \xi_i \text{ for } i = 1 \ldots N$$

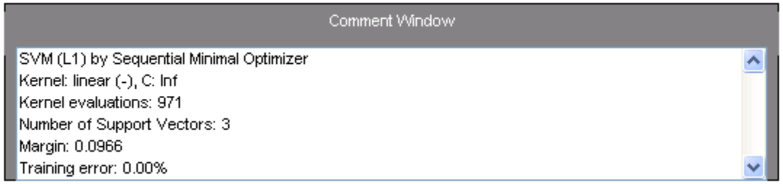
- Every constraint can be satisfied if ξ_i is sufficiently large
- C is a regularization parameter:
  - small C allows constraints to be easily ignored $\rightarrow$ large margin
  - large C makes constraints hard to ignore $\rightarrow$ narrow margin
  - $C = \infty$ enforces all constraints: hard margin
- This is still a quadratic optimization problem and there is a unique minimum. Note, there is only one parameter, C.



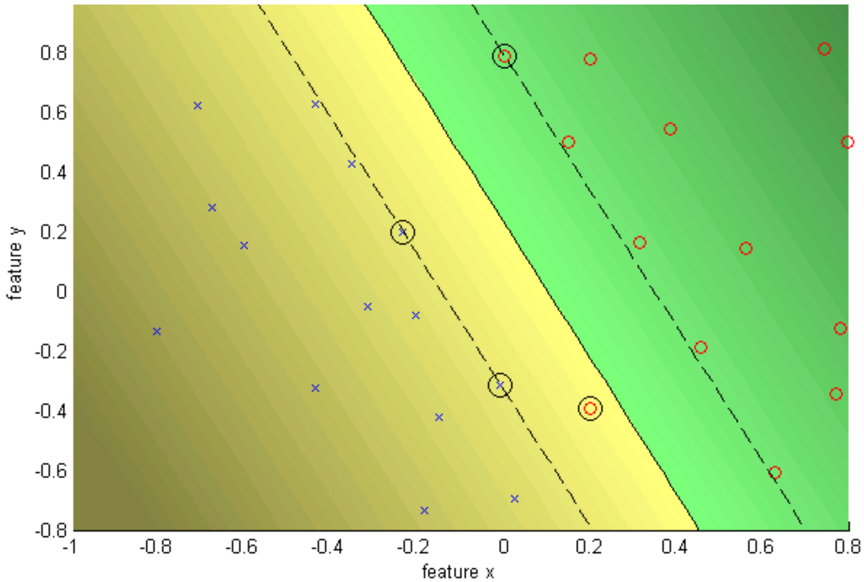
- data is linearly separable
- but only with a narrow margin

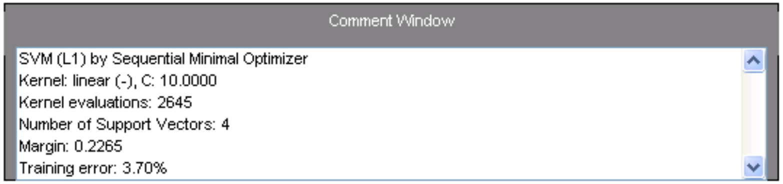
C = Infinity   hard margin





C = 10 soft margin





Optimization

Learning an SVM has been formulated as a constrained optimization problem over $\mathbf{w}$ and ξ

$$\min_{\mathbf{w}\in\mathbb{R}^d,\xi_i\in\mathbb{R}^+} ||\mathbf{w}||^2 + C\sum_i^N \xi_i \text{ subject to } y_i\left(\mathbf{w}^\top\mathbf{x}_i + b\right) \geq 1 - \xi_i \text{ for } i = 1\ldots N$$

The constraint $y_i\left(\mathbf{w}^\top\mathbf{x}_i + b\right) \geq 1 - \xi_i$, can be written more concisely as

$$y_i f(\mathbf{x}_i) \geq 1 - \xi_i$$

which, together with $\xi_i \geq 0$, is equivalent to

$$\xi_i = \max\left(0, 1 - y_i f(\mathbf{x}_i)\right)$$

Hence the learning problem is equivalent to the unconstrained optimization problem over $\mathbf{w}$

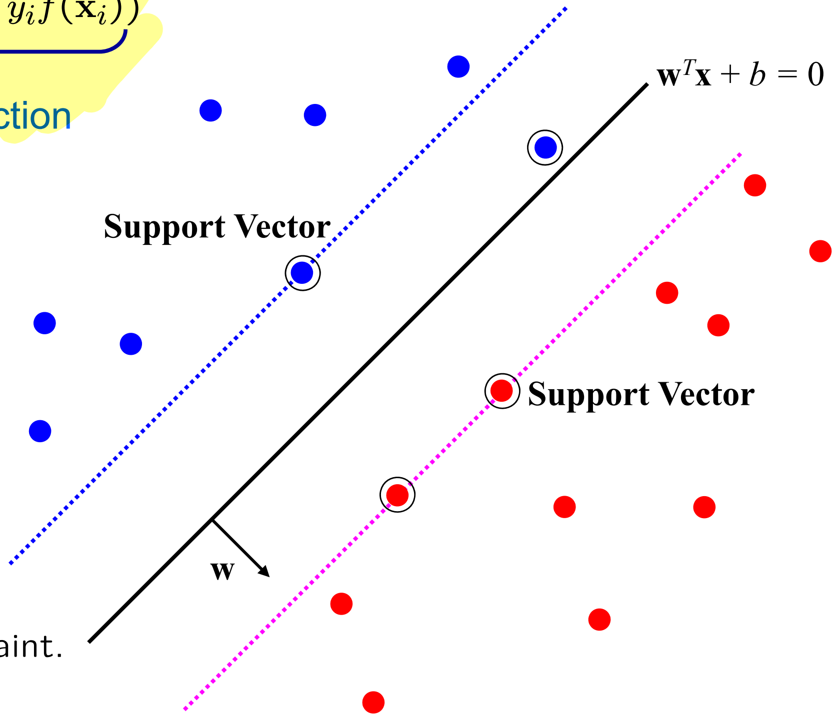
$$\min_{\mathbf{w}\in\mathbb{R}^d} \underbrace{||\mathbf{w}||^2}_{\text{regularization}} + C\sum_i^N \underbrace{\max\left(0, 1 - y_i f(\mathbf{x}_i)\right)}_{\text{loss function}}$$

Loss function

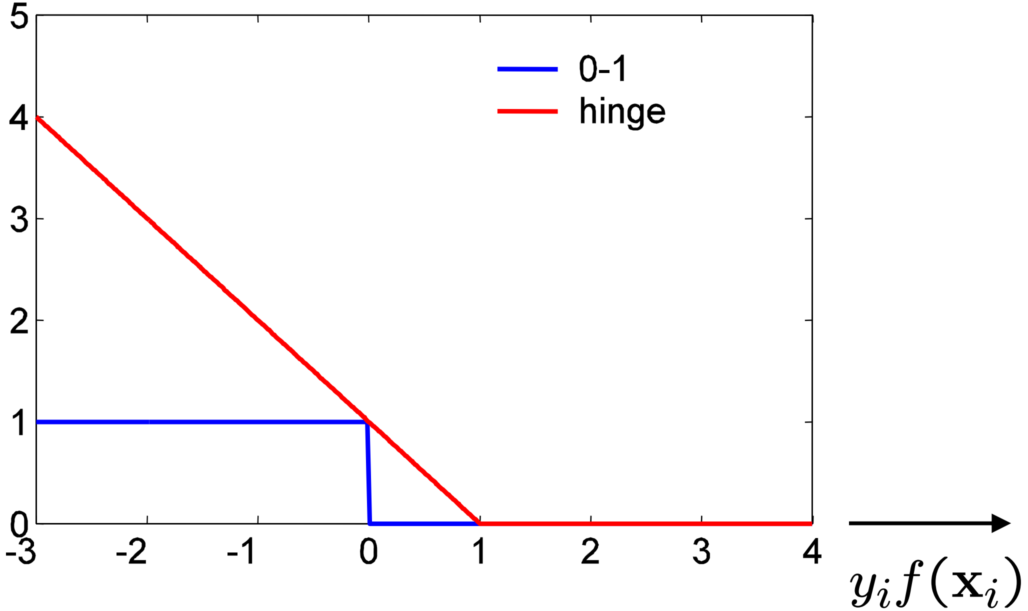
$$\min_{\mathbf{w}\in\mathbb{R}^d} ||\mathbf{w}||^2 + C\sum_i^N \underbrace{\max\left(0, 1 - y_i f(\mathbf{x}_i)\right)}_{\text{loss function}}$$

Points are in three categories:

1. $y_i f(x_i) > 1$
   Point is outside margin.
   No contribution to loss
2. $y_i f(x_i) = 1$
   Point is on margin.
   No contribution to loss.
   As in hard margin case.
3. $y_i f(x_i) < 1$
   Point violates margin constraint.
   Contributes to loss



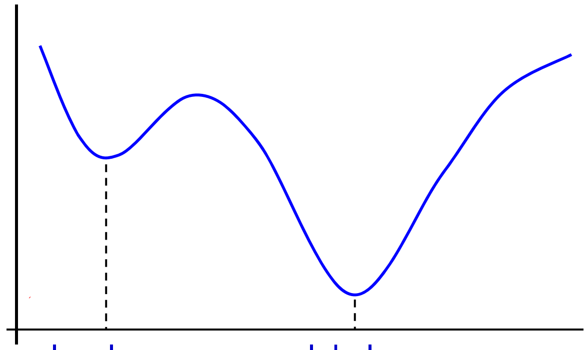
Loss functions



- SVM uses “hinge” loss $\max(0, 1 - y_i f(\mathbf{x}_i))$
- an approximation to the 0-1 loss

Optimization continued

$$\min_{\mathbf{w}\in\mathbb{R}^d} C\sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i)) + ||\mathbf{w}||^2$$



- Does this cost function have a unique solution?
- Does the solution depend on the starting point of an iterative optimization algorithm (such as gradient descent)?

If the cost function is convex, then a locally optimal point is globally optimal (provided the optimization is over a convex set, which it is in our case)

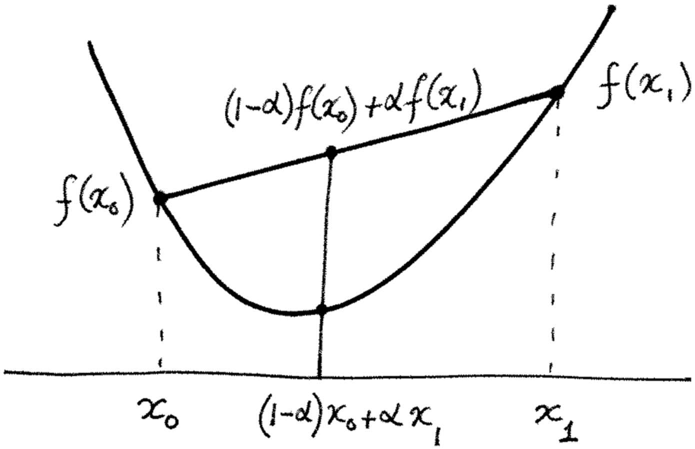
Convex functions

D – a domain in $\mathbb{R}^n$.

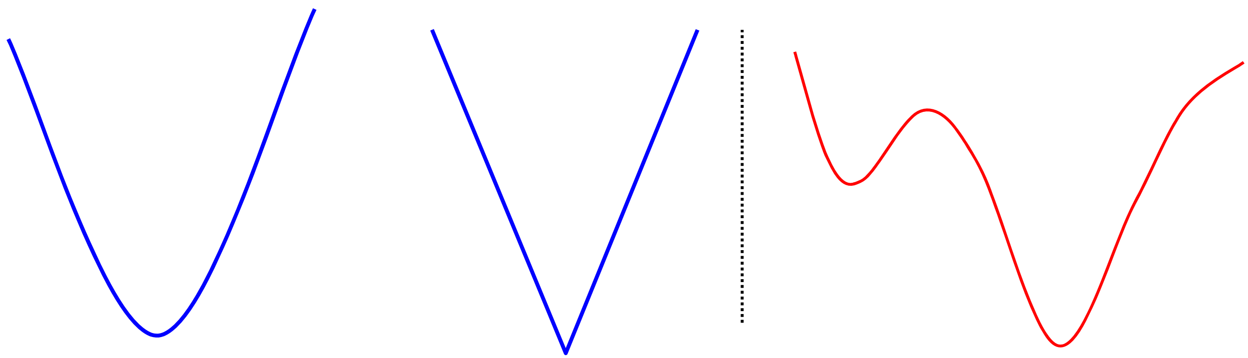
A convex function $f : D \to \mathbb{R}$ is one that satisfies, for any $\mathbf{x}_0$ and $\mathbf{x}_1$ in D:

$$f((1-\alpha)\mathbf{x}_0 + \alpha\mathbf{x}_1) \leq (1-\alpha)f(\mathbf{x}_0) + \alpha f(\mathbf{x}_1) \quad .$$

Line joining $(\mathbf{x}_0, f(\mathbf{x}_0))$ and $(\mathbf{x}_1, f(\mathbf{x}_1))$ lies above the function graph.



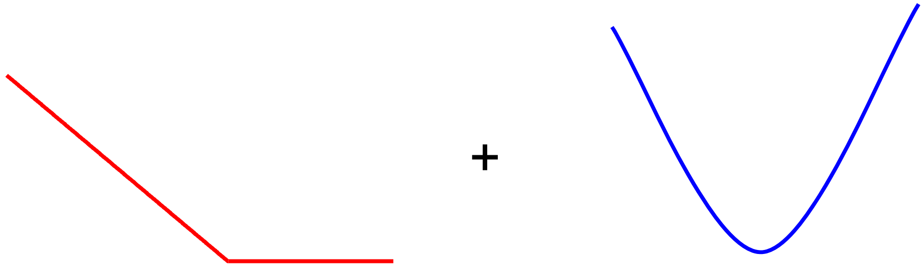
Convex function examples



convex

Not convex

A non-negative sum of convex functions is convex



SVM

$$\min_{\mathbf{w}\in\mathbb{R}^d} C\sum_{i}^{N} \max\left(0, 1 - y_i f(\mathbf{x}_i)\right) + ||\mathbf{w}||^2$$

convex

Gradient (or steepest) descent algorithm for SVM

To minimize a cost function $\mathcal{C}(\mathbf{w})$ use the iterative update

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} \mathcal{C}(\mathbf{w}_t)$$

where η is the learning rate.

First, rewrite the optimization problem as an average

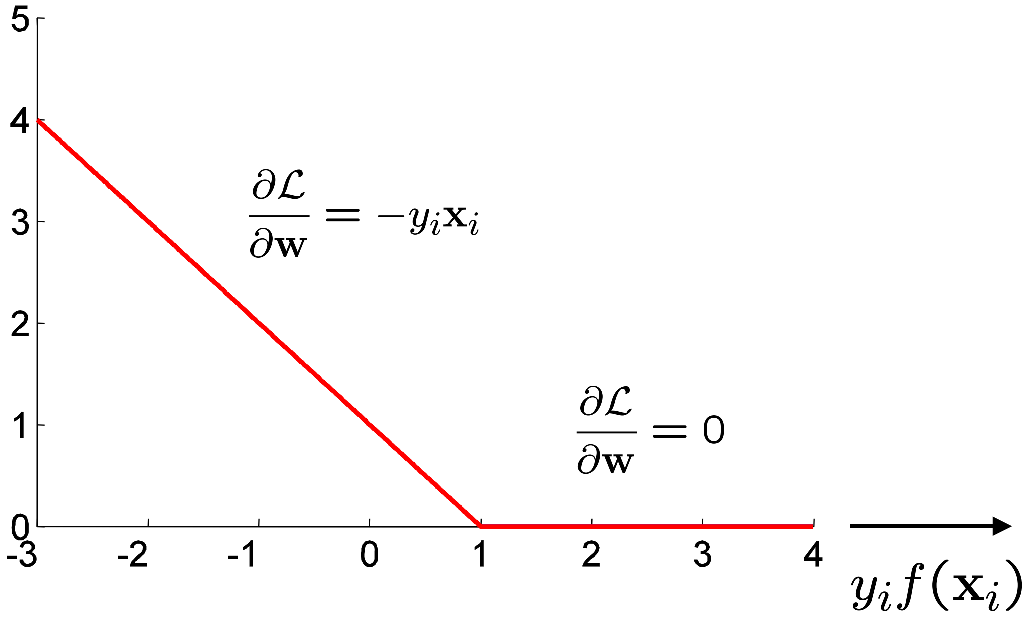
$$\begin{aligned}\min_{\mathbf{w}} \mathcal{C}(\mathbf{w}) &= \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{N}\sum_i^N \max\left(0, 1 - y_i f(\mathbf{x}_i)\right) \\ &= \frac{1}{N}\sum_i^N \left(\frac{\lambda}{2}||\mathbf{w}||^2 + \max\left(0, 1 - y_i f(\mathbf{x}_i)\right)\right)\end{aligned}$$

(with $\lambda = 2/(NC)$ up to an overall scale of the problem) and $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$

Because the hinge loss is not differentiable, a sub-gradient is computed

Sub-gradient for hinge loss

$$\mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}) = \max(0, 1 - y_i f(\mathbf{x}_i)) \qquad f(\mathbf{x}_i) = \mathbf{w}^\top \mathbf{x}_i + b$$



Sub-gradient descent algorithm for SVM

$$\mathcal{C}(\mathbf{w}) = \frac{1}{N}\sum_i^N \left(\frac{\lambda}{2}||\mathbf{w}||^2 + \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w})\right)$$

The iterative update is

$$\begin{aligned} \mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta \nabla_{\mathbf{w}_t} \mathcal{C}(\mathbf{w}_t) \\ &\leftarrow \mathbf{w}_t - \eta \frac{1}{N}\sum_i^N (\lambda \mathbf{w}_t + \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}_t)) \end{aligned}$$

where η is the learning rate.

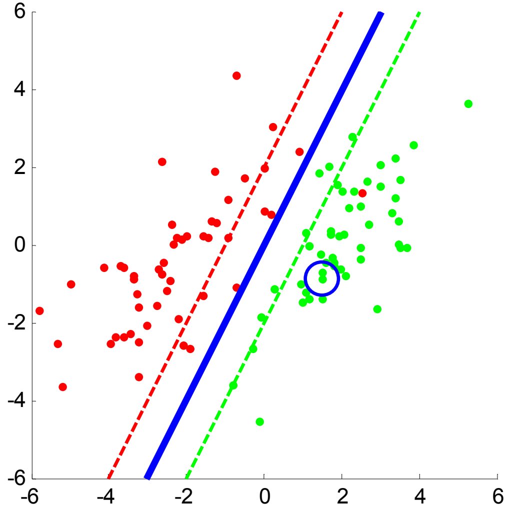
Then each iteration t involves cycling through the training data with the updates:

$$\begin{aligned} \mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta(\lambda \mathbf{w}_t - y_i \mathbf{x}_i) & \text{if } y_i f(\mathbf{x}_i) < 1 \\ &\leftarrow \mathbf{w}_t - \eta \lambda \mathbf{w}_t & \text{otherwise} \end{aligned}$$

In the Pegasos algorithm the learning rate is set at $\eta_t = \frac{1}{\lambda t}$

Pegasos – Stochastic Gradient Descent Algorithm

Randomly sample from the training data



Background reading and more …

• Next lecture – see that the SVM can be expressed as a sum over the support vectors:

$$f(x) = \sum_i \alpha_i y_i (\mathbf{x}_i{}^\top \mathbf{x}) + b$$

support vectors

• On web page:

http://www.robots.ox.ac.uk/~az/lectures/ml

• links to SVM tutorials and video lectures

• MATLAB SVM demo