KIOS
Research and Innovation Center of Excellence

MSc on Intelligent Critical Infrastructure Systems

Machine Learning

Lectures 5 and 6

Marios Polycarpou

Director, KIOS Research and Innovation Center of Excellence

Professor, Electrical and Computer Engineering

University of Cyprus

University of Cyprus

Imperial College London

funded by:

Linear Regression

Given a dataset of labeled examples: $\{(x_j, y_j)\}_{j=1}^{N}$

where $x_j \in \mathbb{R}^n$ $y_j \in \mathbb{R}$ and N is the size of the dataset.

Interpolation is a method for finding new input-output data within $\{(x_j, y_j)\}_{j=1}^{N}$

Function Interpolation is the problem of defining a function $\hat{f} : \mathbb{R}^n \to \mathbb{R}$

such that $\hat{f}(x_j) = y_j$ for all $j = 1, 2, \ldots N$

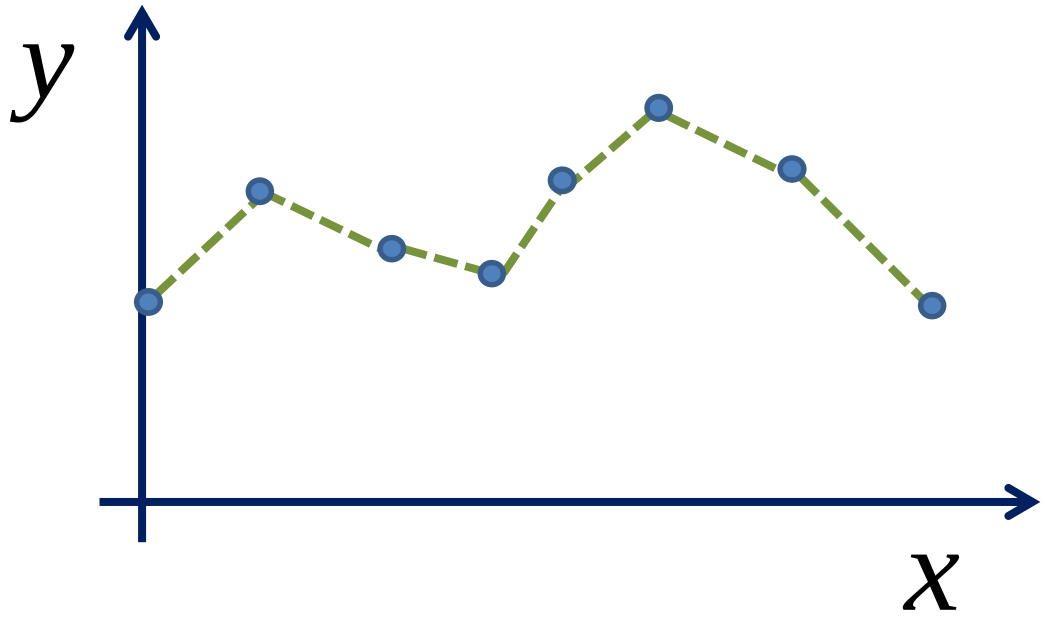
Linear Regression

If we use linear interpolation then that results in linear approximation

Suppose (x, y) is a linear interpolation between two points: (x_a, y_a) (x_b, y_b)

$$y = y_a + \left(\frac{y_a - y_b}{x_a - x_b} \right)(x - x_a)$$

- Simple
- Not very accurate
- Not differentiable

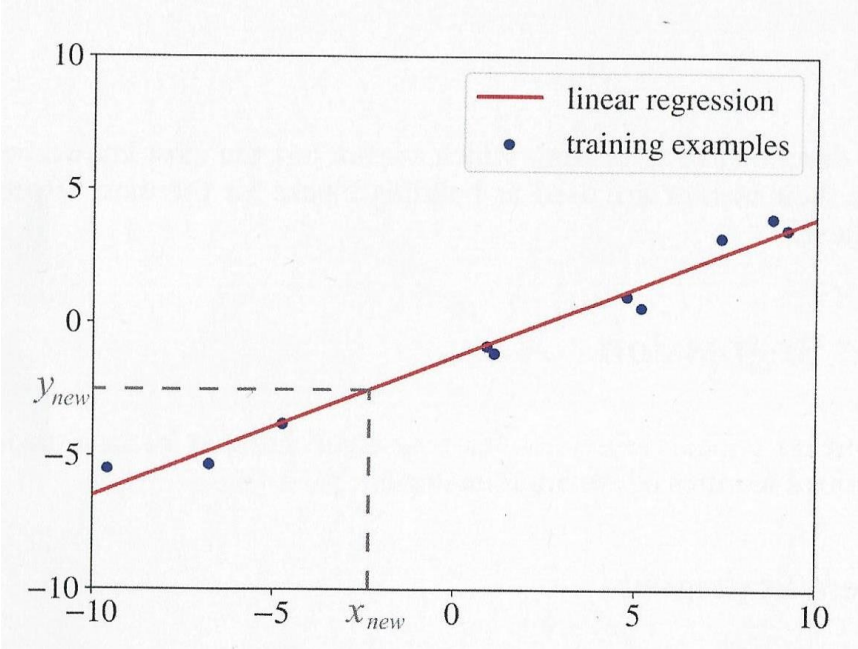


Linear Regression

If we use a linear combination of features,

we obtain:

$$\hat{f}(x) = \sum_{j=1}^{n} \theta_j x_j + \theta_0 = \theta^T x + \theta_0 = \bar{\theta}^T \bar{x}$$



$$\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^n \qquad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n \qquad \bar{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \bar{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

parameters

Linearly Parameterized Regression

→ Function interpolation where $\hat{f}(x)$ is an element of a finite dimensional linear space:

$$\hat{f}(x) = \sum_{i=1}^{M} \theta_i \phi_i(x) = \theta^T \phi(x) = \phi(x)^T \theta$$

N : number of examples in dataset

n : number of features

M : number of weights (adjustable parameters)

$$\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_M \end{bmatrix} \in \mathbb{R}^M$$

parameters

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \vdots \\ \phi_M(x) \end{bmatrix} : \mathbb{R}^n \to \mathbb{R}^M$$

Basis functions

Linearly Parameterized Regression

→ To satisfy the input-output pair: $\{(x_j, y_j)\}_{j=1}^{N}$

$$x_j \in \mathbb{R}^n$$

$$y_j \in \mathbb{R}$$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} \phi(x_1)^T \\ \vdots \\ \phi(x_N)^T \end{bmatrix} \cdot \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_M \end{bmatrix}$$

$$Y = \Phi^T \theta$$

Interpolation matrix
or collocation matrix
of dimension N x M

- n = 1 : scalar interpolation
- n > 1 : scattered data interpolation (if not on a grid)

→ To achieve interpolation we require $M \geq N$

Linearly Parameterized Regression

→ If $M = N$ and Φ is nonsingular, we have a unique solution:

$$Y = \Phi^T \theta \quad \Longrightarrow \quad \boxed{\theta = (\Phi^T)^{-1} Y = \Phi^{-T} Y}$$

REMARKS

- When Φ is nonsingular, the linear space spanned by the corresponding basis functions is called a *Chebyshev space* or a *Haar space*.
- For a unique solution, the number of parameters must be equal to the number of sample points.
- As the number of samples increases, eventually we will end up trying to fit measurement noise, which is undesirable (overfitting)
- In on-line applications (real-time learning), N becomes quite large (unbounded!)
- In practice, M is kept fixed.

Linearly Parameterized Regression

Given an adaptive approximator: $\hat{f}(x;\theta)$

Find a set of parameters θ such that $\hat{f}(x;\theta)$

is close to $f(x)$

Key Issues:

- How to select the adaptive approximator? How to choose the basis functions?
- How to select closeness?
- How to obtain the optimal parameters?

Linearly Parameterized Approximator (batch learning)

Over-constrained solution $(M < N)$

Cannot fit the data perfectly, so the objective is to find approximation parameters θ that minimize some measure of the approximation error.

$$J(\theta) = \frac{1}{2}\left(Y - \Phi^T\theta\right)^T W \left(Y - \Phi^T\theta\right) = \left\|Y - \Phi^T\theta\right\|_W^2$$

Weighted 2-norm (Euclidean norm)

W: symmetric and positive definite

Linearly Parameterized Approximator (batch learning)

Over-constrained solution $(M < N)$

To find optimal θ*: $\left.\frac{\partial J}{\partial \theta}\right|_{\theta=\theta^*} = 0$ $\left.\frac{\partial J}{\partial \theta}\right|_{\theta=\theta^*} = \Phi W\left(\Phi^T\theta^* - Y\right) = 0$

$$\left(\Phi W \Phi^T\right)\theta^* - \Phi W Y = 0$$

$$\theta^* = \left(\Phi W \Phi^T\right)^{-1} \Phi W Y$$

Weighted Least Squares solution
(assumes rank(Φ) = M)

If W = μI, where μ > 0 is a scalar:

Least Squares solution

$$\theta^* = \left(\Phi\,\Phi^T\right)^{-1} \Phi Y$$

Linearly Parameterized Approximator (batch learning)

Under-constrained solution $(M > N)$

→ In this case, we either have no solution or an infinite number of solutions.

→ For the case of infinite number of solutions, we can consider the minimum norm solution using Lagrange multipliers.

$$J(\theta) = \frac{1}{2}\theta^T\theta + \lambda^T\left(Y - \Phi^T\theta\right)$$

$$\left.\begin{aligned} \left.\frac{\partial J}{\partial \theta}\right|_{\theta=\theta^*} &= \theta^{*T} - \lambda^{*T}\Phi^T = 0 \\ \left.\frac{\partial J}{\partial \lambda}\right|_{\lambda=\lambda^*} &= Y - \Phi^T\theta^* = 0 \end{aligned}\right\}$$

$$\lambda^* = \left(\Phi^T\Phi\right)^{-1}Y$$

$$\theta^* = \underbrace{\Phi\left(\Phi^T\Phi\right)^{-1}}Y$$

pseudo-inverse

Linear Regression using Gradient Descent (LMS)

$$\hat{f}(x) = \sum_{j=1}^{n} \theta_j x_j + \theta_0 = \theta^T x + \theta_0 = \bar{\theta}^T \bar{x}$$

$$J(\theta, \theta_0) = \frac{1}{N} \sum_{i=1}^{N} (\theta^T x_i + \theta_0 - y_i)^2$$

Given $\{(x_j, y_j)\}_{j=1}^{N}$ **find parameters** θ, θ_0 **to minimize cost function** $J(\theta, \theta_0)$

$$\theta(k+1) = \theta(k) - \alpha \frac{\partial J}{\partial \theta}$$

$$\theta_0(k+1) = \theta_0(k) - \alpha \frac{\partial J}{\partial \theta_0}$$

$$\theta(k+1) = \theta(k) - \alpha \left(\frac{2}{N} \sum_{i=1}^{N} (\theta^T(k) x_i + \theta_0(k) - y_i) x_i \right)$$

$$\theta_0(k+1) = \theta_0(k) - \alpha \left(\frac{2}{N} \sum_{i=1}^{N} (\theta^T(k) x_i + \theta_0(k) - y_i) \right)$$

Least Mean Squares (LMS) rule
(Widrow-Hoff learning rule)

$$\theta(k+1) = \theta(k) - \alpha \left(\theta^T(k) x(k) + \theta_0(k) - y(k) \right) x(k)$$

$$\theta_0(k+1) = \theta_0(k) - \alpha \left(\theta^T(k) x(k) + \theta_0(k) - y(k) \right)$$

Linear Regression using Gradient Descent (LMS)

- Batch gradient descent vs. stochastic gradient descent
- Batch gradient descent has to go through the entire training set. This is called an **epoch**.
- Stochastic gradient descent often gets “close” to the optimal parameters much faster than the batch gradient descent

Linear Regression using Gradient Descent

Linearly parametrized approximation models

$$\hat{f}(x) = \sum_{i=1}^{M} \theta_i \phi_i(x) = \theta^T \phi(x) = \phi(x)^T \theta$$

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} (\theta^T \phi(x) - y_i)^2$$

$$\theta(k+1) = \theta(k) - \alpha \left(\theta^T(k) \phi(x(k)) - y(k) \right) \phi(x(k))$$

Linearly Parameterized Approximator (online learning)

$$\theta^* = \left(\Phi W \Phi^T\right)^{-1} \Phi W Y$$

Weighted Least Squares solution
(assumes rank(Φ) = M)

$$\theta_k = \left(\Phi_k W_k \Phi_k^T\right)^{-1} \Phi_k W_k Y_k$$

Recursive Weighted Least Squares (2nd order optimization)

$$\theta_{k+1} = \theta_k + A_k^{-1}\left(\phi_{k+1}^T A_k^{-1} \phi_{k+1} + w_{k+1}^{-1}\right)^{-1} \phi_{k+1}\left(y_{k+1} - \phi_{k+1}^T \theta_k\right)$$

$$A_k^{-1} = A_{k-1}^{-1} - A_{k-1}^{-1}\phi_k\left(\phi_k^T A_{k-1}^{-1}\phi_k + w_k^{-1}\right)^{-1}\phi_k^T A_{k-1}^{-1}$$

$$\theta_{k+1} = \theta_k + \Omega_k \phi_{k+1}\left(y_{k+1} - \phi_{k+1}^T \theta_k\right)$$

$$\Omega_k = A_k^{-1}\left(\phi_{k+1}^T A_k^{-1}\phi_{k+1} + w_{k+1}^{-1}\right)^{-1}$$

- **Another way to write it**
- **Several simplifications exist**

Linearly Parameterized Approximator (online learning)

Continuous-time Recursive Least Squares

$$\dot{\theta}(t) = P(t)\phi(t)\left(y(t) - \phi(t)^T \theta(t)\right)$$

$$\dot{P}(t) = -P(t)\phi(t)\phi(t)^T P(t)$$

For further reading see:

- G. C. Goodwin and K. S. Sin, “Adaptive Filtering Prediction and Control” Prentice Hall, 1984
- C. R. Johnson Jr., “Lectures on Adaptive Parameter Estimation” Prentice Hall, 1988.

Logistic Regression

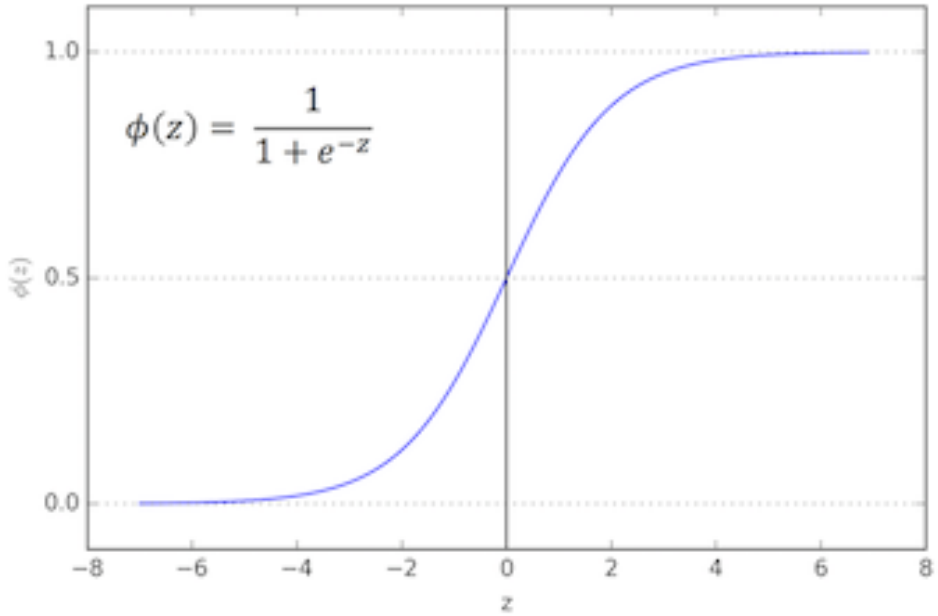
Sigmoidal combination of features:

$$\hat{f}(x) = \frac{1}{1+e^{-(\theta^T x+\theta_0)}} = \frac{1}{1+e^{-\bar{\theta}^T \bar{x}}}$$

$$\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^n \qquad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$$

$$\bar{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \bar{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

parameters



Logistic Regression

- Logistic regression is a classification learning algorithm (not regression!)
- Solution cannot be obtained in closed-form.
- Maximum Likelihood formulation and then use stochastic gradient descent.

$$\hat{f}(x) = \frac{1}{1+e^{-(\theta^T x+\theta_0)}}$$

Sigmoidal function

$$\theta(k+1) = \theta(k) - \alpha\left(\frac{1}{1+e^{-(\theta^T(k)x(k)+\theta_0(k))}} - y(k)\right)x(k)$$

$$\theta_0(k+1) = \theta_0(k) - \alpha\left(\frac{1}{1+e^{-(\theta^T(k)x(k)+\theta_0(k))}} - y(k)\right)$$

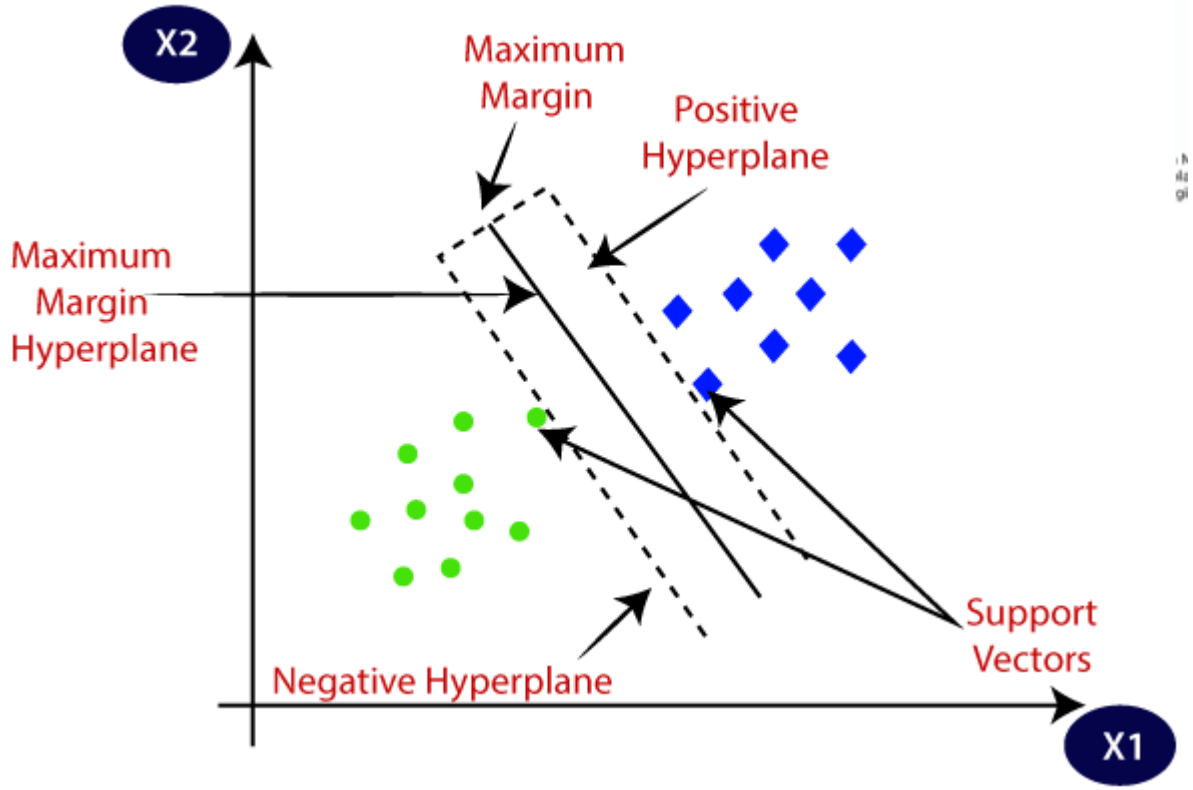
Support Vector Machines (SVM)

$$\theta^T x_i - \theta_0 \geq +1 \quad \text{if} \quad y_i = +1$$
$$\theta^T x_i - \theta_0 \leq -1 \quad \text{if} \quad y_i = -1$$

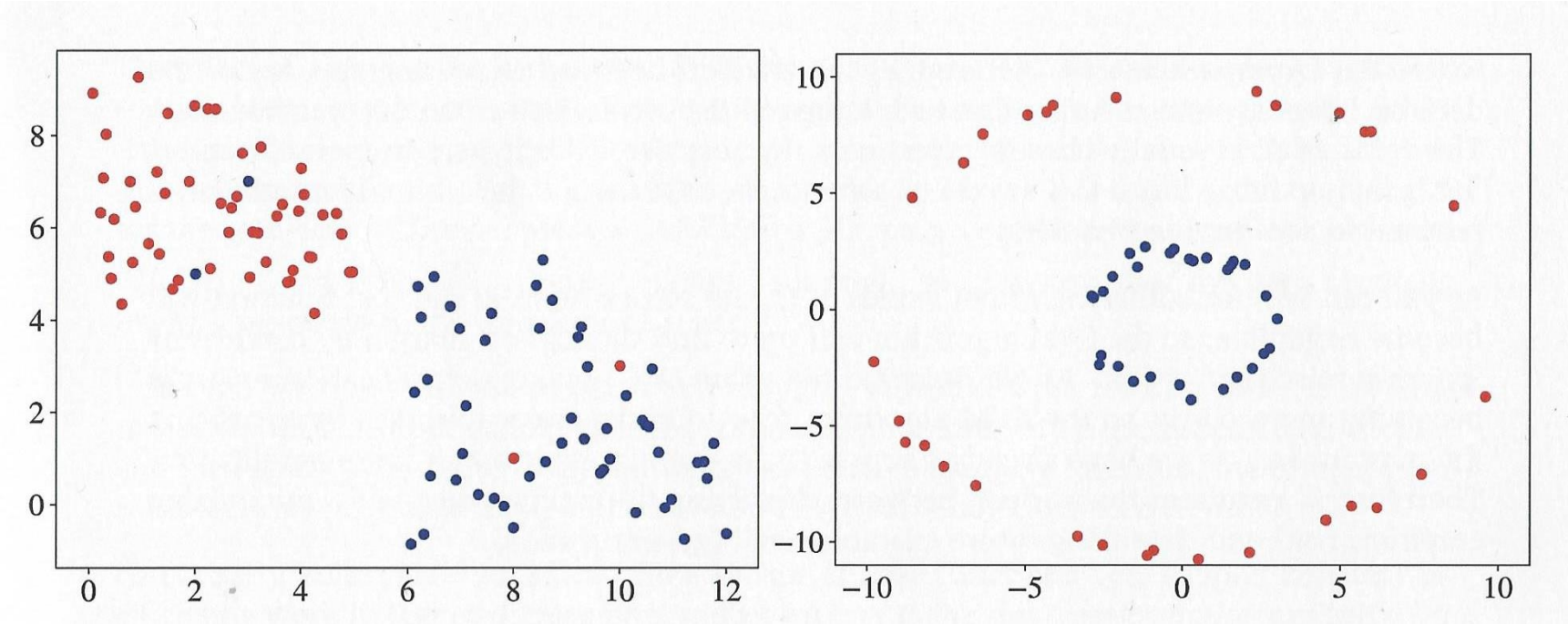
$$\text{Minimize} \quad \frac{1}{2}\|\theta\|^2$$

$$\text{subject to} \quad y_i(\theta^T x_i - \theta_0) \geq 1$$

$$\text{for} \quad i = 1, 2, \ldots N$$



Support Vector Machines (SVM)



Dealing with noise

Hinge loss function

Minimize $$C\|\theta\|^2 + \frac{1}{N}\sum_{i=1}^{N}\max\left(0,\ 1 - y_i(\theta^T x_i - \theta_0)\right)$$

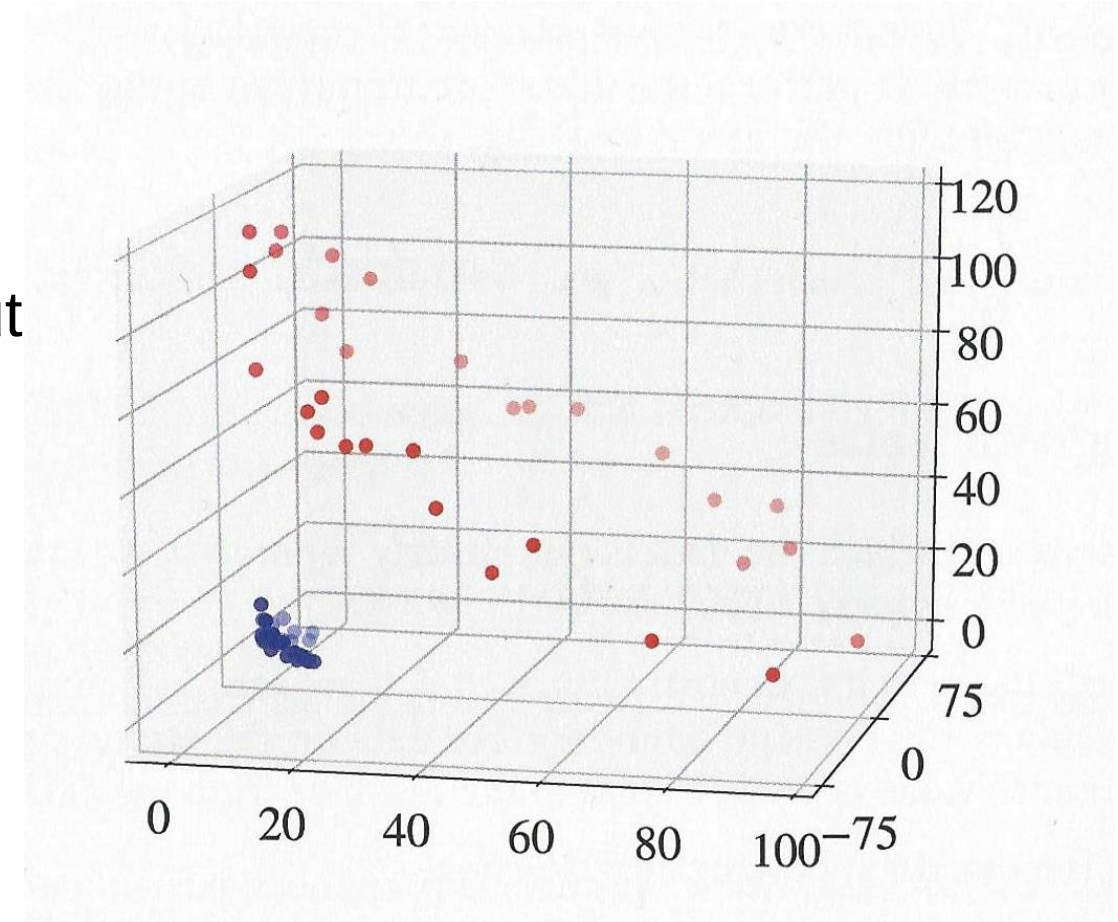
Support Vector Machines (SVM)

Dealing with inherent nonlinearity

- Transform the original space to a higher dimensional space so that the data becomes linearly separable.
- Kernel functions provide a systematic way to work in higher dimension without needing to come up with a transformation explicitly.

$$\varphi : x \mapsto \varphi(x)$$

$$\varphi([q, p]) = \left(q^2, \sqrt{2}qp, p^2\right)$$



Feature Engineering

- Feature engineering is the process of using domain knowledge of the data to create features that enhance machine learning algorithms. Basically, transforms raw data into a dataset.
- If feature engineering is done correctly, it increases the predictive ability of machine learning algorithms.
- Feature engineering is an art

STEPS FOR ML

- Gathering Data
- Cleaning data
- ***Feature engineering***
- Selecting a model and optimization algorithm
- Training, testing model and predicting the output

Feature Engineering

One-Hot Encoding

- Integer encoding vs. one-hot encoding
- E.g. colors are not sequential
  - RED = [1, 0, 0]
  - BLUE = [0, 1, 0]
  - GREEN = [0, 0, 1]
- If the order of the feature's values is not important then using integer encoding may confuse the learning algorithm.

Feature Engineering

Binning (or Bucketing)

- Converting a numerical feature into a categorical feature
- E.g. representation of age (instead of age use age bins)
- Binning can help the learning algorithm to learn using fewer examples
- Similar to giving “hints” to the learning algorithm

Feature Engineering

Normalization and Standardization

- Normalization converts the raw range of numerical feature into a standard range of values (usually, [-1, 1] or [0, 1]).
- Standardization rescales the numerical values of a feature so that it has a standard normal distribution (mean = 0; standard deviation = 1)
- Both normalization and standardization may improve the learning rate and also help avoid numerical overflow problems

Which one to use?

- No clear winner!
- Rule of Thumb: use normalization except in the following cases:
  - Unsupervised learning
  - If the values of a feature are close to a bell curve
  - If the feature has extremely high or low values (outliers)

Feature Engineering

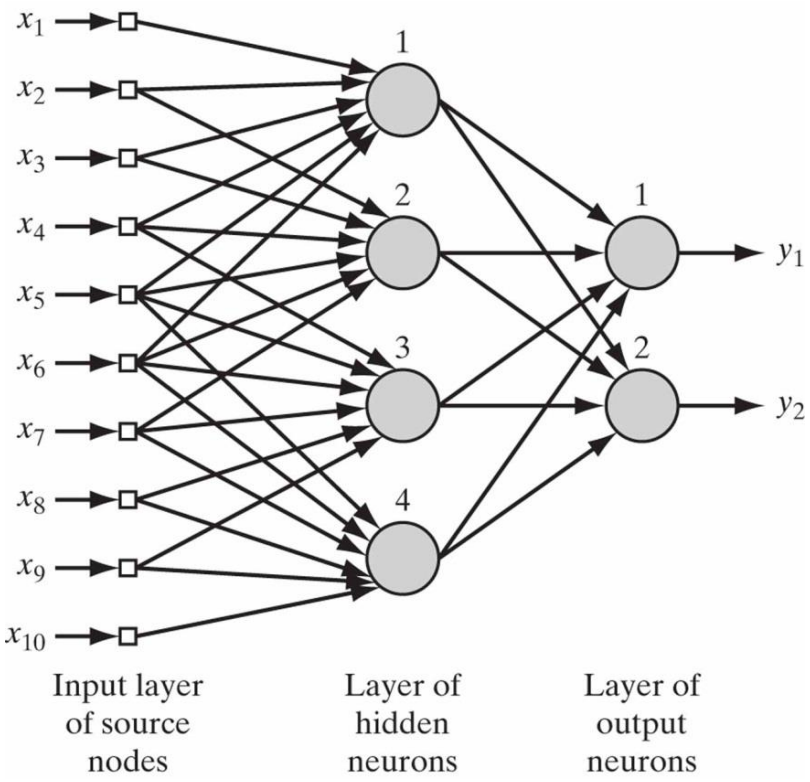
Dealing with Missing Features

- Remove the examples with missing features
- Use ML algorithms that can handle missing features
- Use a *data imputation* method:
  - Replace the missing value of a feature by an average value of this feature in the dataset, or the middle of the range values
  - Replace the missing value with a value outside the normal range of values for that feature
  - Use regression to estimate the missing feature value

Supervised Learning in Multilayer Neural Networks

Notation

- $E(n) = \frac{1}{2}\sum_{i_L} e_{i_L}^2$: value of the cost function associated with the n-th training pattern, *n=1,...,N.*
- $E_{av} = \frac{1}{N}\sum_{n} E(n)$
- $e_{i_L}(n) = d_{i_L}(n) - y_{i_L}(n)$: error for the particular example
- m_l: denotes the size of layer *l=0,...,L* where m_0, m_L is the size of input and output layer, respectively.
- $w_{i_l i_{l-1}}(n)$: value of the synaptic weight connecting the output of neuron *i* to the input of neuron *j*
- i_l: index related to the i-th neuron of the *l*-th layer, $i_l = 0,1,...,m_l$



Supervised Learning in Multilayer Neural Networks

- Given a number of training examples *[x(n),d(n)]*, *n=1,...,N*, train the network (find appropriate values of the synaptic weights that minimize our objective function E_{av}).

$$E_{av}(W) = \frac{1}{N}\sum_{n=1}^{N} E(W,n) = \frac{1}{2N}\sum_{n=1}^{N}\sum_{i_L=1}^{m_L}(d_{i_L}(n) - y_{i_L}(n))^2$$

- Objective: $\min_{W} E(\mathrm{W})$
- What is the expression that gives the output value of an output neuron?

$$y_{i_L}(n) = f(v_{i_L}) = f\Big(\sum_{i_{L-1}=1}^{m_{L-1}} w_{i_L i_{L-1}}(n)\, y_{i_{L-1}}\Big), \qquad i_L = 1,\ldots,m_L$$

- $\underline{y}_{i_{L-1}}$: is a function of the weights of the previous layer. Hence an output neuron is a function of all weights of the network. Additionally, the weights of the output layer only appear in the expressions of the output neurons.

Computing the Gradient of E_{av}

- What is the gradient in this case?

$$\Delta w_{i_l,i_{l-1}} = [\nabla E_{av}(W)]_{i_l,i_{l-1}} = \frac{\partial E_{av}(W)}{\partial w_{i_l,i_{l-1}}}, \qquad l = 1,\ldots,N, \quad \forall i_l, i_{l-1}$$

- In order to create an iterative gradient descent optimization algorithm we need to compute $\Delta w_{i_l,i_{l-1}}$ in each iteration of the algorithm for all layers.
- How are weights not directly connected to the output layer associated with the objective function?
- Weights connecting neurons from layer *l-1* to layer *l* appear in the output of neurons in layers *l,l+1,…,L.*
- How can we deal with this problem? We need to apply the Chain Rule.

Computing the gradient of E_{av}

- **Chain Rule**: If $z = f(g(\underline{x}))$ and $f(.)$, $g(.)$ are continuous differentiable functions then:

$$\frac{dz}{dx_i} = \frac{\partial f}{\partial g}\frac{\partial g(\underline{x})}{\partial x_i}$$

- Consider in our case a weight connecting a neuron from layer *L-1* to layer *L*. We have that:

$$\frac{\partial E_{av}(W)}{\partial w_{i_L i_{L-1}}} = \frac{1}{N}\sum_{n=1}^{N}\frac{\partial E(W,n)}{\partial w_{i_L i_{L-1}}}$$

This method can be generalized in order to compute the gradient of all weights of the network exploiting the computations of the previous steps via the local gradient δ_{iL}

$$\frac{\partial E(W,n)}{\partial w_{i_L i_{L-1}}} = \frac{\partial E(W,n)}{\partial e_{i_L}}\frac{\partial e_{i_L}(n)}{\partial y_{i_L}}\frac{\partial y_{i_L}(n)}{\partial v_{i_L}}\frac{\partial v_{i_L}(n)}{\partial w_{i_L i_{L-1}}}$$

$$= (e_{i_L}(n))(-1)(f'(v_{i_L}(n)))(y_{i_L}(n))$$

$$= -\underbrace{e_{i_L}(n) f'(v_{i_L}(n))}_{\delta_{i_L}(n) = -\frac{\partial E(W,n)}{\partial v_{i_L}}} y_{i_L}(n)$$

Backpropagation Gradient Descent Algorithm

1. Initialize the weights and the biases of the neurons
2. Present the network with an epoch of training examples applying to each example steps 3-4.
3. ***Forward Computation*:** Compute the outputs of the neurons y_{jl} starting from the neurons of hidden layer one. The neurons of the next layer are only computed after the output of all neurons of the particular layer have been computed.
4. ***Backward Computation:*** Compute the local gradients of the network starting from the output layer and going to the input layer using:

$$\delta_{i_l}(n) = \begin{cases} e_{i_L}(n)\varphi'_{i_L}(v_{i_L}(n)), & l = L, \forall i_L \\ \varphi'_{i_l}(v_{i_l}(n))\sum_{i_{l+1}} \delta_{i_{l+1}}(n) w_{i_{l+1}i_l}(n), & l = 1,\ldots,L-1, \forall i_l \end{cases}$$

and update the weights: $w_{i_{l+1}i_l}(n+1) = w_{i_{l+1}i_l}(n) + a w_{i_{l+1}i_l}(n-1) + \eta \delta_{i_{l+1}}(n) y_{i_l}(n)$

5. ***Convergence:*** Iterate with new epochs for all learning examples until a stopping criterion is met.

Backpropagation Algorithm – Learning rate

- **Constant learning rate $0 \leq \eta < 1$**
  - **Small:** slow convergence.
  - **Large:** Oscillatory behaviour near a local minimum.
- **Momentum constant $0 \leq \alpha < 1$**
  - Provides an increased learning rate and smooths out possible oscillations without requiring any extra information.

$$\Delta w_{i_l i_{l-1}}(n) = -\eta \sum_t \frac{\partial E^t(n)}{\partial w_{i_l i_{l-1}}}$$

  - When the partial derivatives $\partial E^t(n) / \partial w_{i_l i_{l-1}}$ for two consecutive iterations have the same sign the momentum term tends to increase the descent
  - When they have opposing sign the momentum tends to reduce the effect of the weight correction
- We can choose a different learning rate η for each weight parameter, when they are updated separately.

Backpropagation Algorithm – Activation Functions

- Sigmoid function

$$\varphi(v_j(n)) = \frac{1}{1+\exp(-av_j(n))}$$

$$\varphi'(v_j(n)) = \frac{a\exp(-av_j(n))}{[1+\exp(-av_j(n))]^2} = ay_j(n)(1-y_j(n))$$

- Hyperbolic tangent function

$$\varphi(v_j(n)) = a\tanh(bv_j(n)), \quad a,b>0$$

$$\varphi'(v_j(n)) = ab(1-\tanh(bv_j(n)))$$

$$= \frac{b}{a}[a-y_j(n)][a+y_j(n)]$$

Backpropagation Algorithm - Training Modes

Sequential mode

- Weight updating is performed after the presentation of each training example
- Good practice to randomize the order of presentation of training examples from one epoch to the next to avoid getting trapped into a local minimum
- We aim at minimizing *E(W,n)* for each *n*; as a result of this minimization, we also minimize $E_{av}(W)$.
- Difficult to establish theoretical conditions for convergence

Backpropagation Algorithm - Training Modes

Batch mode

- Weight updating after the presentation of all training examples that constitute an epoch
- Weight corrections are obtained by:

$$\Delta w_{i_l i_{l-1}} = -\eta \frac{\partial E_{av}(W)}{\partial w_{i_l i_{l-1}}} = -\frac{\eta}{N} \sum_{n} \sum_{i_L} \frac{\partial e_{i_L}(W, n)}{\partial w_{i_l i_{l-1}}}, \quad l = 1, \ldots L$$

- Required substantial local storage to make the weight updates
- Provides an accurate estimate of gradient vector and convergence to a local minimum can be guaranteed under simple conditions

➢ *The sequential mode of training is preferred over the batch mode because it is easier to implement and provides effective solutions to large problems*

Backpropagation Algorithm – Convergence Criteria

- $\nabla W^* = 0$: Stop the learning algorithm when the Euclidean norm of the gradient vector is sufficiently small
- $E_{av}(w^*) < E_{av}(w)$: Stop the learning algorithm when the absolute rate of change of the cost function in an epoch becomes sufficiently small.
- Cross-validation: instead of having only a training and a test set, we split the training set into an estimation and a validation subset, and we stop when sufficient performance is achieved for the validation set