AARHUS
UNIVERSITET

Computer Vision & Machine Learning

Alexandros Iosifidis
@
Department of Electrical and Computer Engineering
Aarhus University

This week

Regression models

Linear Classification models

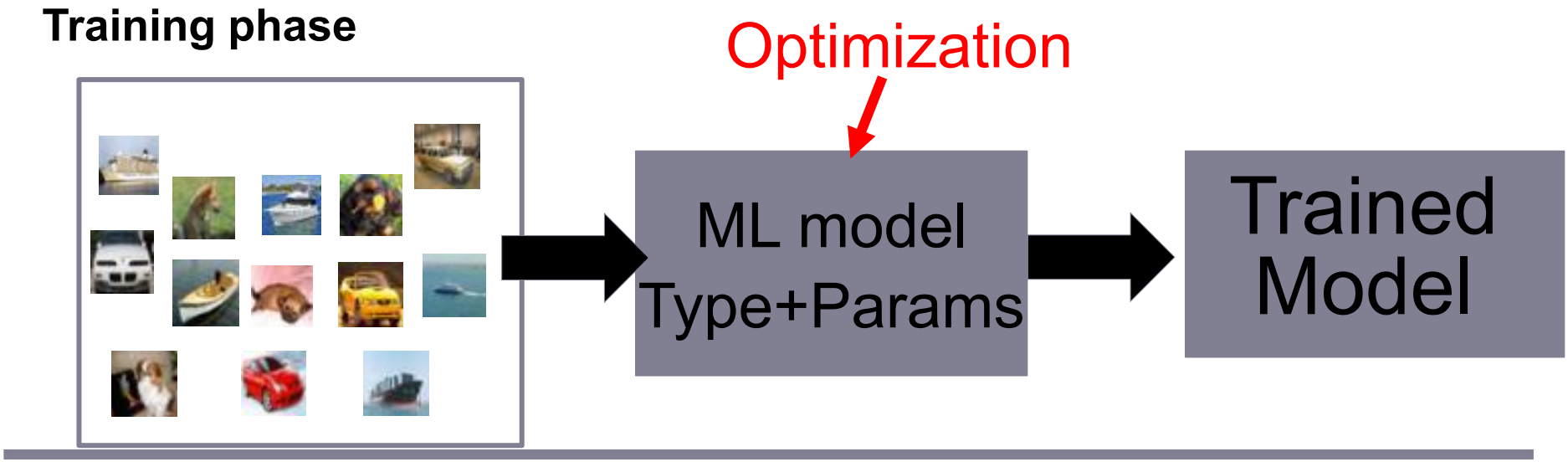
Non-linear Classification/Regression models

Non-linear Clustering

Acceleration/Approximation approaches for non-linear models

ML for out-of-sample analysis

Training phase



Test phase/Evaluation/Online process

Linear Regression

In regression problems, we want to find a model that maps a vector $X \in \mathbb{R}^p$ to a real-valued output Y.

A linear model for this regression has the form:

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

Linear Regression

In regression problems, we want to find a model that maps a vector $X \in \mathbb{R}^p$ to a real-valued output Y.

A linear model for this regression has the form:

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

In order to define the parameters of the model β, we use a set of training data (x_1,y_1) …(x_N,y_N). Each $x_i \in \mathbb{R}^p$ and y_i is a real (target) value.

By minimizing the residual sum of squares we have:

$$\mathrm{RSS}(\beta) = \sum_{i=1}^{N} (y_i - f(x_i))^2 = \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij} \beta_j \right)^2$$

Linear Regression

By minimizing the residual sum of squares we have:

$$\mathrm{RSS}(\beta) = \sum_{i=1}^{N}(y_i - f(x_i))^2 = \sum_{i=1}^{N}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2$$

Using an augmented version of the vectors $x_i \leftarrow [x_i^T\ 1]^T$, we have:

$$\mathrm{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)$$

Where $X = [x_1 \ldots x_N]$ and $y = [y_1 \ldots y_N]^T$

Linear Regression

By minimizing the residual sum of squares we have:

$$\mathrm{RSS}(\beta) = \sum_{i=1}^{N}(y_i - f(x_i))^2 = \sum_{i=1}^{N}\Big(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\Big)^2$$

Using an augmented version of the vectors $x_i \leftarrow [x_i^T\ 1]^T$, we have:

$$\mathrm{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)$$

Where $X = [x_1 \ldots x_N]$ and $y = [y_1 \ldots y_N]^T$

The derivatives of RSS(β) are:

$$\frac{\partial \mathrm{RSS}}{\partial \beta} = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta)$$

$$\frac{\partial^2 \mathrm{RSS}}{\partial \beta \partial \beta^T} = 2\mathbf{X}^T\mathbf{X}.$$

Linear Regression

By minimizing the residual sum of squares we have:

$$\mathrm{RSS}(\beta) = \sum_{i=1}^{N}(y_i - f(x_i))^2 = \sum_{i=1}^{N}\Big(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\Big)^2$$

Using an augmented version of the vectors $x_i \leftarrow [x_i^T\ 1]^T$, we have:

$$\mathrm{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)$$

Where $X = [x_1 \ldots x_N]$ and $y = [y_1 \ldots y_N]^T$

Setting the derivative of RSS(β) to zero: $\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0$

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

Linear Regression

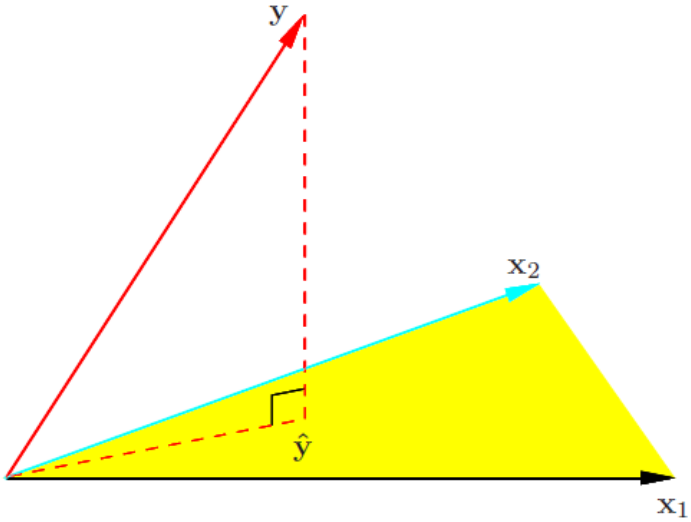
The predicted values at for the input vectors are given by:

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

If X^TX is singular, an inexact solution will be obtained (the values of β are not uniquely defined).

If the target values y_i are uncorrelated and have constant variance σ^2, then:

$$\hat{\beta} \sim N(\beta, (\mathbf{X}^T\mathbf{X})^{-1}\sigma^2)$$



Linear Regression

The predicted values at for the input vectors are given by:

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

If X^TX is singular, an inexact solution will be obtained (the values of β are not uniquely defined).

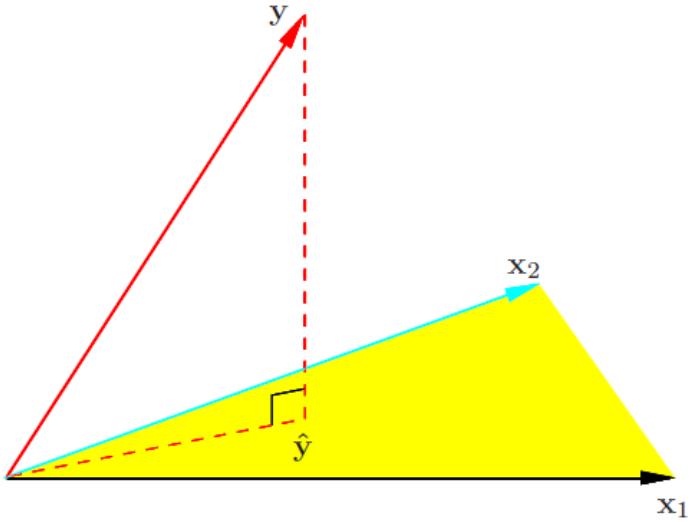
The RSS estimate is also unbiased:

- Consider the parameters $\theta = a^T\beta$, then:

$$\hat{\theta} = a^T\hat{\beta} = a^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

- The expected value of θ is:

$$\begin{aligned} \mathrm{E}(a^T\hat{\beta}) &= \mathrm{E}(a^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}) \\ &= a^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\beta = a^T\beta \end{aligned}$$



Linear Regression

In the case where the regression target is a vector $Y \in \mathbb{R}^K$, then the linear regression for the k-th elements is:

$$\begin{aligned} Y_k &= \beta_{0k} + \sum_{j=1}^{p} X_j \beta_{jk} + \varepsilon_k \\ &= f_k(X) + \varepsilon_k. \end{aligned}$$

Linear Regression

In the case where the regression target is a vector Y ϵ $\mathbb{R}^K$, then the linear regression for the k-th elements is:

$$\begin{aligned} Y_k &= \beta_{0k} + \sum_{j=1}^{p} X_j \beta_{jk} + \varepsilon_k \\ &= f_k(X) + \varepsilon_k. \end{aligned}$$

With N training vectors, we get $\mathbf{Y} = \mathbf{XB} + \mathbf{E}$, where Y ϵ $\mathbb{R}^{NxK}$ is the target matrix, X ϵ $\mathbb{R}^{Nx(p+1)}$ is the augmented data matrix and B ϵ $\mathbb{R}^{(p+1)xK}$ is the matrix of parameters.

The parameters matrix B is calculated by minimizing:

$$\begin{aligned} \mathrm{RSS}(\mathbf{B}) &= \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2 \\ &= \mathrm{tr}[(\mathbf{Y} - \mathbf{XB})^T (\mathbf{Y} - \mathbf{XB})] \end{aligned}$$

Linear Regression

In the case where the regression target is a vector Y ϵ $\mathbb{R}^K$, then the linear regression for the k-th elements is:

$$\begin{aligned} Y_k &= \beta_{0k} + \sum_{j=1}^{p} X_j \beta_{jk} + \varepsilon_k \\ &= f_k(X) + \varepsilon_k. \end{aligned}$$

With N training vectors, we get $\mathbf{Y} = \mathbf{XB} + \mathbf{E}$, where Y ϵ $\mathbb{R}^{NxK}$ is the target matrix, X ϵ $\mathbb{R}^{Nx(p+1)}$ is the augmented data matrix and B ϵ $\mathbb{R}^{(p+1)xK}$ is the matrix of parameters.

The parameters matrix B is calculated by minimizing:

$$\begin{aligned} \mathrm{RSS}(\mathbf{B}) &= \sum_{k=1}^{K}\sum_{i=1}^{N}(y_{ik} - f_k(x_i))^2 \\ &= \mathrm{tr}[(\mathbf{Y} - \mathbf{XB})^T(\mathbf{Y} - \mathbf{XB})] \end{aligned} \quad \Rightarrow \quad \hat{\mathbf{B}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

Ridge Regression

The Ridge Regression model is a linear regression model where we set additional constraints on the size of the parameter values:

$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

where $\lambda \geq 0$ is a parameter that controls the amount of shrinkage for β.

The above problem is written in a matrix form as:

$$\text{RSS}(\lambda) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) + \lambda\beta^T\beta$$

and the solution is

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

Ridge Regression

Ridge Regression optimization problem can be written also in the form:

$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 \leq t,$$

which makes explicit the constraint on the size of the values of β.

Using Lagrange optimization, the same problem as before is obtained:

$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

LASSO

The Least Absolute Shrinkage and Selection Operator (LASSO) uses a different constraint on the values of β:

$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2$$

$$\text{subject to} \sum_{j=1}^{p} |\beta_j| \leq t.$$

which can be transformed to the Lagrangian form:

$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \left\{ \frac{1}{2} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

LASSO

The replace of the l_2 norm penalty $\sum_1^p \beta_j^2$ to the l_1 norm penalty $\sum_1^p |\beta_j|$ makes the solution of this problem more complicated (there is no closed-form solution). Existing solutions use iterative processes.

The effect of using the l_1 norm penalty is that many values of β are pushed close to zero, i.e. we obtain a sparse solution.

$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \left\{ \frac{1}{2} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

RR and LASSO generalization

The optimization problems of LASSO and Ridge Regression can be generalized to the following (for values q=1 and q=2, respectively):

$$\tilde{\beta} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|^q \right\}$$

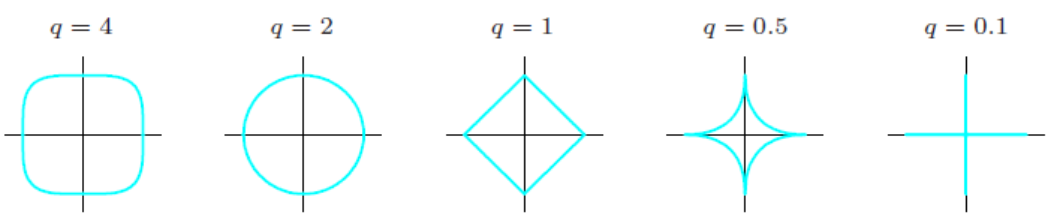


FIGURE 3.12. *Contours of constant value of $\sum_j |\beta_j|^q$ for given values of q*

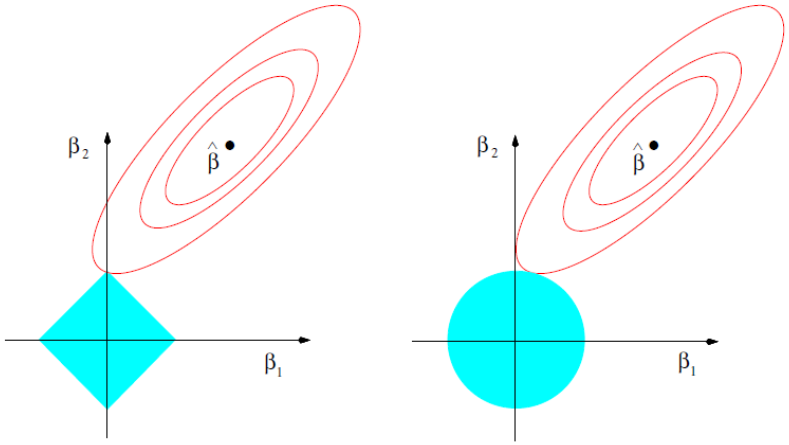


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \le t$ and $\beta_1^2 + \beta_2^2 \le t^2$, respectively, while the red ellipses are the contours of the least squares error function.*

Principal Components Regression

PCR determines the linear regression from the principal components of the input data X to the target values y:

$$\hat{\mathbf{y}}^{\mathrm{pcr}}_{(M)} = \bar{y}\mathbf{1} + \sum_{m=1}^{M} \hat{\theta}_m \mathbf{z}_m$$

where $\hat{\theta}_m = \langle \mathbf{z}_m, \mathbf{y} \rangle / \langle \mathbf{z}_m, \mathbf{z}_m \rangle$ and z_m are the principal components of X:

- X is decomposed using SVD → $X = VSU^T$
- $z_m = XU_m$ is the m-th principal component of X

In order to discard the dependency to the mean vector of y and X, both are centered at the beginning.

The regression coefficients in terms of X are given by: $\hat{\beta}^{\mathrm{pcr}}(M) = \sum_{m=1}^{M} \hat{\theta}_m v_m$

Linear Regression of an Indicator Matrix

Let us assume that we have N samples X_i, i=1,…,N and that each sample is followed by a class label $l_i \in \{1,..,K\}$.

An Indicator matrix $Y \in \mathbb{R}^{NxK}$ is a matrix formed by K vectors $Y_k \in \mathbb{R}^N$ having elements equal to $Y_{ki} = 1$ if $l_i = k$ and $Y_{ki} = 0$, otherwise.

Then, we can create the following regression-based classifier:

- Calculate the regression parameters

$$\hat{\mathbf{B}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

- A new vector x is classified by calculating the regression vector $\hat{f}(x)^T = (1, x^T)\hat{\mathbf{B}}$ and assigning to it the label corresponding to the maximum value:

$$\hat{G}(x) = \text{argmax}_{k \in \mathcal{G}} \hat{f}_k(x)$$

Logistic Regression

In LR we consider the indicator matrix as a probability matrix:

- Each row of Y contains the conditional probability of the corresponding class, given the observation x, e.g. the first element encodes: $\Pr(G = 1|X = x)$
- Since we consider probabilities, they should sum to one and remain in the interval [0,1]
- The same holds also for the predicted values (note that all above regression models do not satisfy these conditions!)

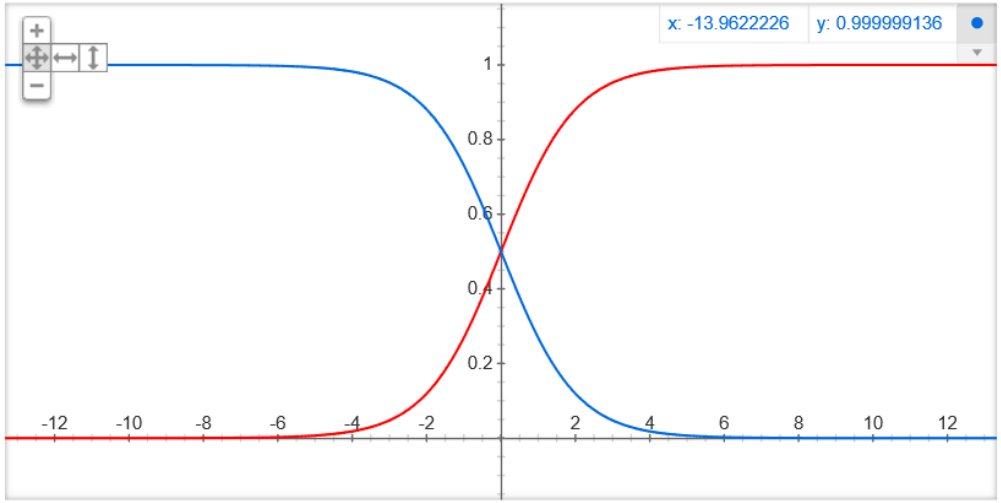
Logistic Regression

For a two-class problem we define the conditional probabilities using the logit transformation log[p/(1-p)]:

$$\Pr(G=1|X=x) = \frac{\exp(\beta_0+\beta^T x)}{1+\exp(\beta_0+\beta^T x)}$$

$$\Pr(G=2|X=x) = \frac{1}{1+\exp(\beta_0+\beta^T x)}$$

1/(1+exp(x)), exp(x)/(1+exp(x))



Logistic Regression

For a two-class problem we define the conditional probabilities using the logit transformation log[p/(1-p)]:

$$\Pr(G=1|X=x) = \frac{\exp(\beta_0 + \beta^T x)}{1+\exp(\beta_0 + \beta^T x)}$$

$$\Pr(G=2|X=x) = \frac{1}{1+\exp(\beta_0 + \beta^T x)}$$

This leads to log-odds equal to: $\log \frac{\Pr(G=1|X=x)}{\Pr(G=2|X=x)} = \beta_0 + \beta^T x$

The decision boundary is the set of points for which log-odds are equal to zero, which is the hyperplane $\{x|\beta_0 + \beta^T x = 0\}$

Logistic Regression

The generalization of the model for K classes has the form:

$$\begin{aligned}
\Pr(G=k|X=x) &= \frac{\exp(\beta_{k0}+\beta_k^T x)}{1+\sum_{\ell=1}^{K-1}\exp(\beta_{\ell 0}+\beta_\ell^T x)}, \quad k=1,\ldots,K-1, \\
\Pr(G=K|X=x) &= \frac{1}{1+\sum_{\ell=1}^{K-1}\exp(\beta_{\ell 0}+\beta_\ell^T x)},
\end{aligned}$$

leading to log-odds (w.r.t. the last class):

$$\begin{aligned}
\log\frac{\Pr(G=1|X=x)}{\Pr(G=K|X=x)} &= \beta_{10}+\beta_1^T x \\
\log\frac{\Pr(G=2|X=x)}{\Pr(G=K|X=x)} &= \beta_{20}+\beta_2^T x \\
&\vdots \\
\log\frac{\Pr(G=K-1|X=x)}{\Pr(G=K|X=x)} &= \beta_{(K-1)0}+\beta_{K-1}^T x
\end{aligned}$$

Logistic Regression

The generalization of the model for K classes has the form:

$$\Pr(G=k|X=x) = \frac{\exp(\beta_{k0}+\beta_k^T x)}{1+\sum_{\ell=1}^{K-1}\exp(\beta_{\ell 0}+\beta_\ell^T x)}, \quad k=1,\ldots,K-1,$$

$$\Pr(G=K|X=x) = \frac{1}{1+\sum_{\ell=1}^{K-1}\exp(\beta_{\ell 0}+\beta_\ell^T x)},$$

leading to log-odds (w.r.t. the last class):

$$\log\frac{\Pr(G=1|X=x)}{\Pr(G=K|X=x)} = \beta_{10}+\beta_1^T x$$

$$\log\frac{\Pr(G=2|X=x)}{\Pr(G=K|X=x)} = \beta_{20}+\beta_2^T x$$

$$\vdots$$

$$\log\frac{\Pr(G=K-1|X=x)}{\Pr(G=K|X=x)} = \beta_{(K-1)0}+\beta_{K-1}^T x$$

Note that here we have (K-1) β's. Why?

Logistic Regression

To emphasize the dependence of Pr(·) to the parameter set $\theta = \{\beta_{10}, \beta_1^T, \ldots, \beta_{(K-1)0}, \beta_{K-1}^T\}$ we use the term $\Pr(G = k|X = x) = p_k(x;\theta)$

The parameter values θ of the model is fit by using maximum likelihood. For N samples, the log-likelihood is given by:

$$\ell(\theta) = \sum_{i=1}^{N} \log p_{g_i}(x_i;\theta)$$

where $p_k(x_i;\theta) = \Pr(G = k|X = x_i;\theta)$

Label for x_i

Logistic Regression

Two-class case:

class 1 → $y_i = 1$ and $p_1(x;\theta) = p(x;\theta)$

class 2 → $y_i = 0$ and $p_2(x;\theta) = 1 - p(x;\theta)$

Using $\beta = [\beta_{10}, \beta_1^T]^T$ and augmented versions for the x_i's, the log-likelihood takes the form:

$$\begin{aligned}\ell(\beta) &= \sum_{i=1}^{N}\Big\{y_i \log p(x_i;\beta) + (1-y_i)\log(1-p(x_i;\beta))\Big\} \\ &= \sum_{i=1}^{N}\Big\{y_i\beta^T x_i - \log(1+e^{\beta^T x_i})\Big\}\end{aligned}$$

We set the derivative of $l(\beta)$ to zero, leading to (p+1) equations which are nonlinear to β:

$$\frac{\partial \ell(\beta)}{\partial \beta} = \sum_{i=1}^{N} x_i(y_i - p(x_i;\beta)) = 0$$

Logistic Regression

Two-class case:

class 1 → $y_i = 1$ and $p_1(x;\theta) = p(x;\theta)$

class 2 → $y_i = 0$ and $p_2(x;\theta) = 1 - p(x;\theta)$

Using $\beta = [\beta_{10}, \beta_1^T]^T$ and augmented versions for the x_i's, the log-likelihood takes the form:

$$\begin{aligned}\ell(\beta) &= \sum_{i=1}^{N}\left\{y_i \log p(x_i;\beta) + (1-y_i)\log(1-p(x_i;\beta))\right\} \\ &= \sum_{i=1}^{N}\left\{y_i \beta^T x_i - \log(1+e^{\beta^T x_i})\right\}\end{aligned}$$

We set the derivative of $l(\beta)$ to zero, leading to (p+1) equations which are nonlinear to β:

$$\frac{\partial \ell(\beta)}{\partial \beta} = \sum_{i=1}^{N} x_i(y_i - p(x_i;\beta)) = 0$$

The first value of x_i is equal to 1 → the first equation shows the expected number of class 1

$$\sum_{i=1}^{N} y_i = \sum_{i=1}^{N} p(x_i;\beta)$$

Logistic Regression

Two-class case:

class 1 → $y_i = 1$ and $p_1(x;\theta) = p(x;\theta)$

class 2 → $y_i = 0$ and $p_2(x;\theta) = 1 - p(x;\theta)$

Using $\beta = [\beta_{10}, \beta_1{}^T]^T$ and augmented versions for the x_i's, the log-likelihood takes the form:

$$\begin{aligned} \ell(\beta) &= \sum_{i=1}^{N} \Big\{ y_i \log p(x_i;\beta) + (1-y_i)\log(1-p(x_i;\beta)) \Big\} \\ &= \sum_{i=1}^{N} \Big\{ y_i \beta^T x_i - \log(1+e^{\beta^T x_i}) \Big\} \end{aligned}$$

To maximize $l(\beta)$, we use the Newton-Raphson algorithm:

$$\beta^{\text{new}} = \beta^{\text{old}} - \left(\frac{\partial^2 \ell(\beta)}{\partial\beta\partial\beta^T} \right)^{-1} \frac{\partial \ell(\beta)}{\partial \beta}$$

where $\frac{\partial^2 \ell(\beta)}{\partial\beta\partial\beta^T} = -\sum_{i=1}^{N} x_i x_i{}^T p(x_i;\beta)(1-p(x_i;\beta))$ and the derivatives are evaluated at β^{old}

Logistic Regression

Two-class case:

class 1 → $y_i = 1$ and $p_1(x;\theta) = p(x;\theta)$

class 2 → $y_i = 0$ and $p_2(x;\theta) = 1 - p(x;\theta)$

To maximize l(β), we use the Newton-Raphson algorithm:

$$\beta^{\text{new}} = \beta^{\text{old}} - \left(\frac{\partial^2 \ell(\beta)}{\partial\beta\partial\beta^T}\right)^{-1} \frac{\partial \ell(\beta)}{\partial\beta}$$

Then we have:

$$\begin{aligned}\beta^{\text{new}} &= \beta^{\text{old}} + (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{y}-\mathbf{p}) \\ &= (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}\left(\mathbf{X}\beta^{\text{old}} + \mathbf{W}^{-1}(\mathbf{y}-\mathbf{p})\right) \\ &= (\mathbf{X}^T\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{W}\mathbf{z}\end{aligned}$$

where $\mathbf{z} = \mathbf{X}\beta^{\text{old}} + \mathbf{W}^{-1}(\mathbf{y}-\mathbf{p})$

Logistic Regression

Two-class case:

class 1 → $y_i = 1$ and $p_1(x;\theta) = p(x;\theta)$

class 2 → $y_i = 0$ and $p_2(x;\theta) = 1 - p(x;\theta)$

The updates are done in an iterative manner, since at each iteration p, W and z change.

This algorithm is called Iterative Reweighted Least Squares (IRLS).

A good starting point is $\beta = 0$ (convergence is not guaranteed!, but it is usually the case).

For $K \geq 3$, Newton algorithm can also be expressed as an iteratively reweighted least squares algorithm, but with a vector of $K-1$ responses.

Extension to non-linear models

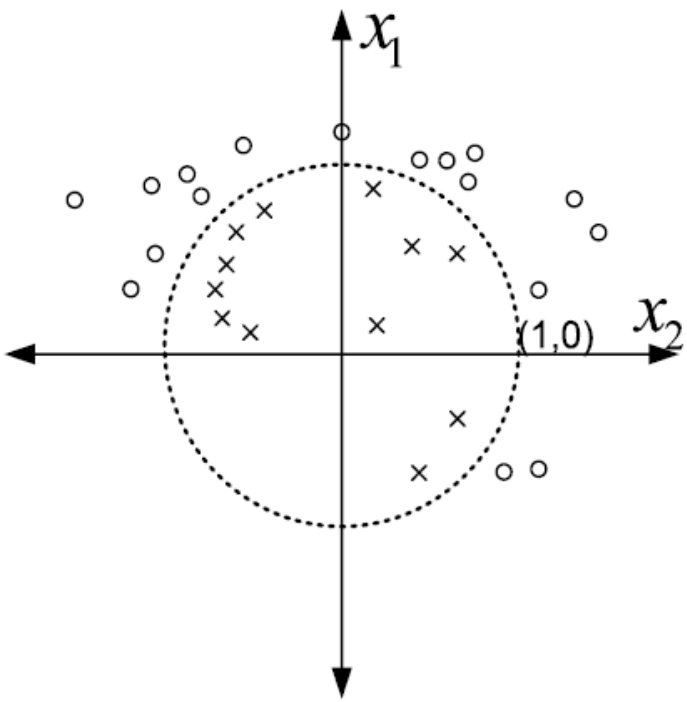
One way to extend all methods presented above for non-linear regression/classification/ clustering/subspace analysis is to:

- apply a non-linear mapping from x_i's to z_i's using a function $f(\cdot)$
- apply the linear method using z_i's

This process leads to non-linear method on the x_i's (Generalized Linear Functions).

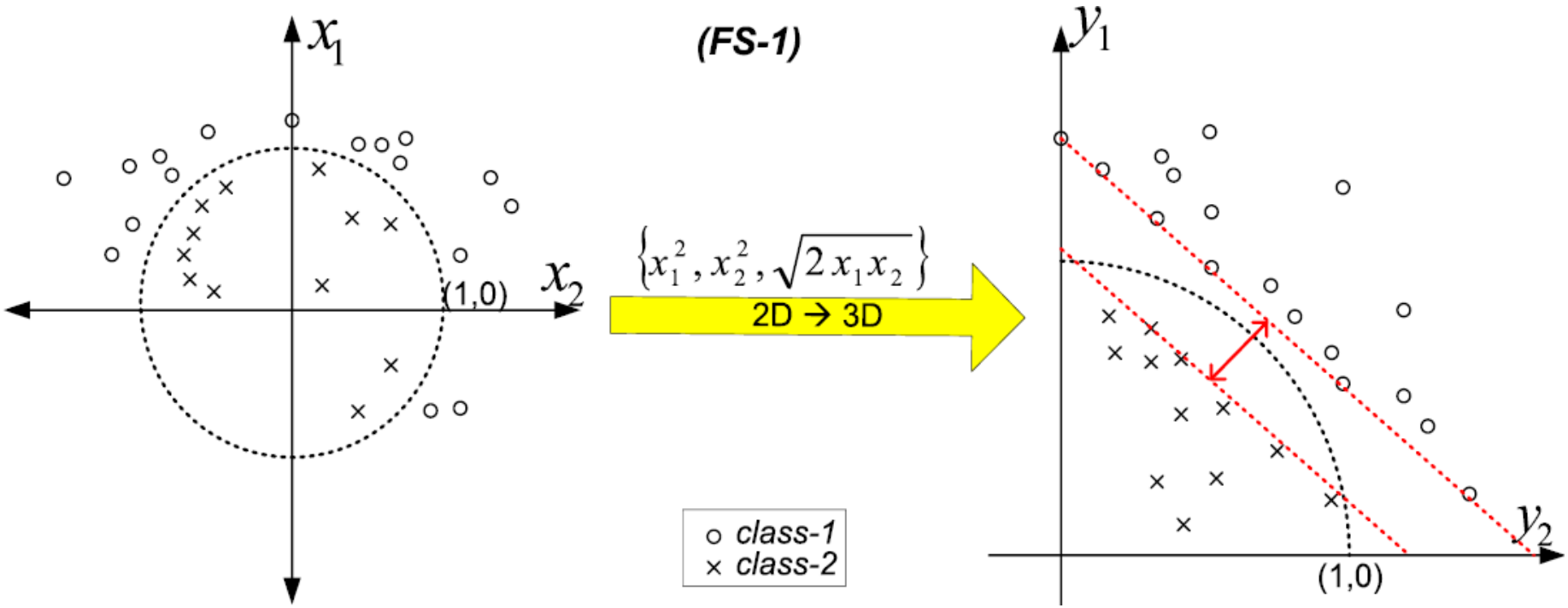
Extension to non-linear models

Example 1



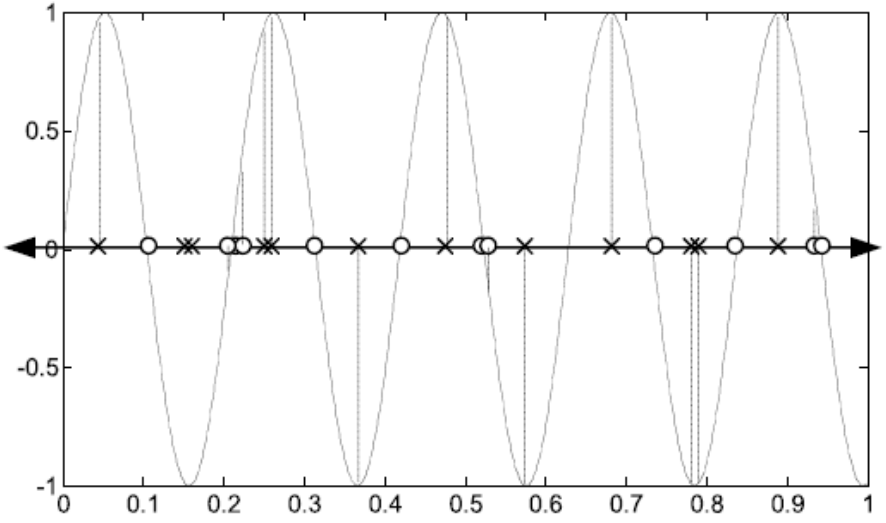
Extension to non-linear models

Example 1



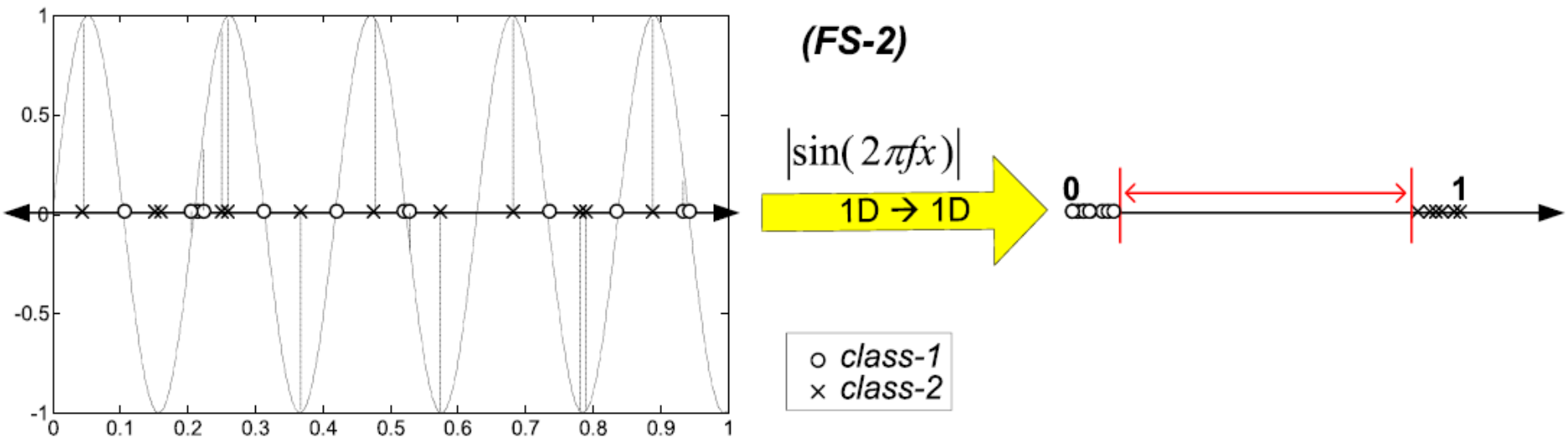
Extension to non-linear models

Example 2



Extension to non-linear models

Example 2



Non-linear regression

Radial Basis Function network:

- Apply a nonlinear mapping $x_i \rightarrow \varphi_i = \varphi(x_i - c_k)$ where c_k, $k=1,\ldots,J_2$ is a set of prototypes (calculated by applying K-Means on x_i)
- Then, apply linear regression using φ_i's: $y_i(\vec{x}) = \sum_{k=1}^{J_2} w_{ki}\phi(\|\vec{x} - \vec{c}_k\|), \quad i = 1, \ldots, J_3,$

Radial basis functions:

$$\mathbf{Y} = \mathbf{W}^T\mathbf{\Phi} \Longrightarrow \mathbf{W} = \left(\mathbf{\Phi}^T\right)^{\dagger}\mathbf{Y}^T = \left(\mathbf{\Phi}\mathbf{\Phi}^T\right)^{-1}\mathbf{\Phi}\mathbf{Y}^T$$

$$\phi(r) = e^{-r^2/2\sigma^2}, \quad \text{Gaussian,}$$

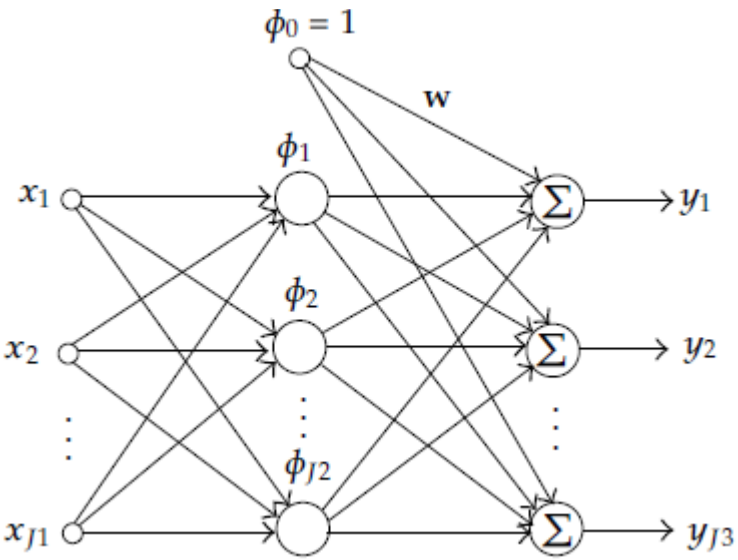
$$\phi(r) = \frac{1}{(\sigma^2 + r^2)^{\alpha}}, \quad \alpha > 0,$$

$$\phi(r) = \left(\sigma^2 + r^2\right)^{\beta}, \quad 0 < \beta < 1,$$

$$\phi(r) = r, \quad \text{linear,}$$

$$\phi(r) = r^2 \ln(r), \quad \text{thin-plate spline,}$$

$$\phi(r) = \frac{1}{1 + e^{(r/\sigma^2) - \theta}}, \quad \text{logistic function,}$$

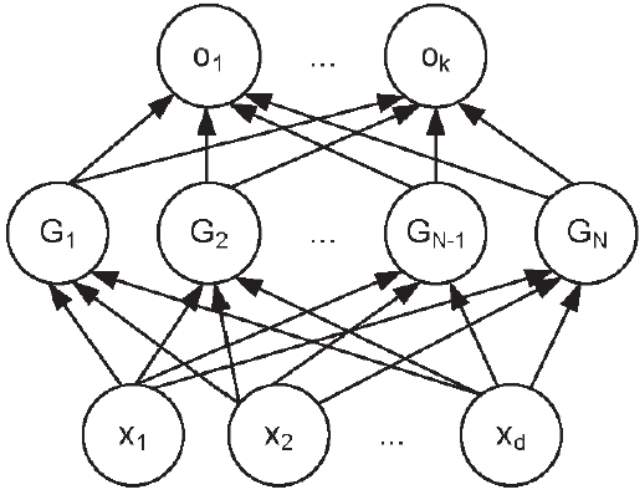


Non-linear regression

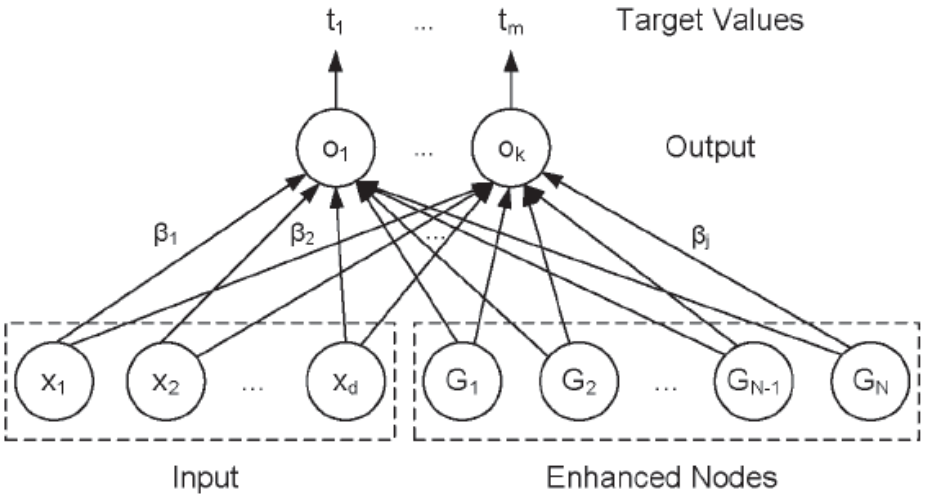
<u>Random Vector Functional Link and Extreme Learning Machine networks:</u>

- Apply a nonlinear mapping $x_i \rightarrow \varphi_i$ using random vectors
- For RVFL: concatenate input vector xi with φ_i: $\varphi_i \leftarrow [x_1^T\ \varphi_i^T]^T$
- Then, apply linear regression using φ_i's:

ELM



RVFL



Extension to non-linear models

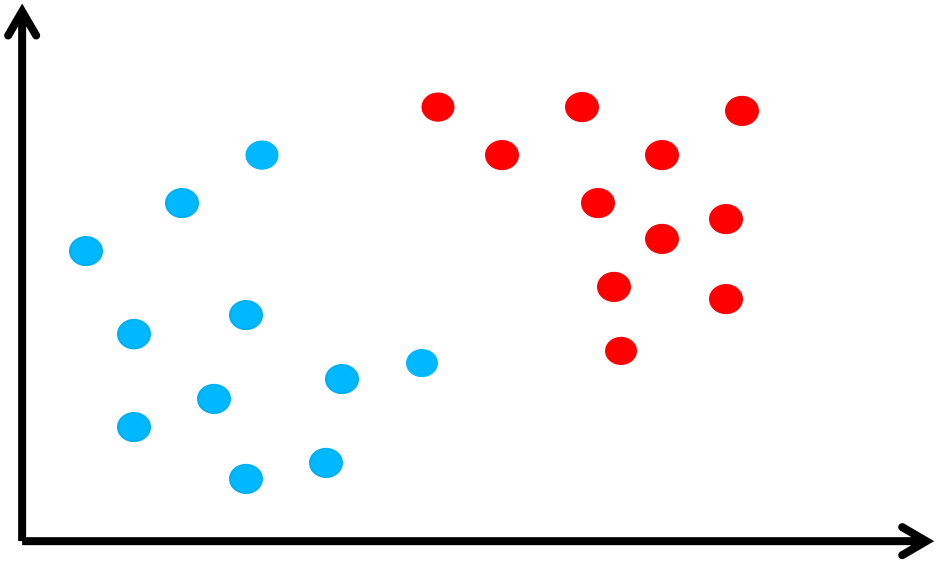
The second way to extend a method for non-linear regression/classification/clustering/ subspace analysis is to:

- apply an **inherent** non-linear mapping from x_i's to φ_i's using a function $\varphi(\cdot)$
- apply the linear method by expressing all equations as dot-products between φ_i's

This process leads to non-linear method on the x_i's (kernel-based methods).

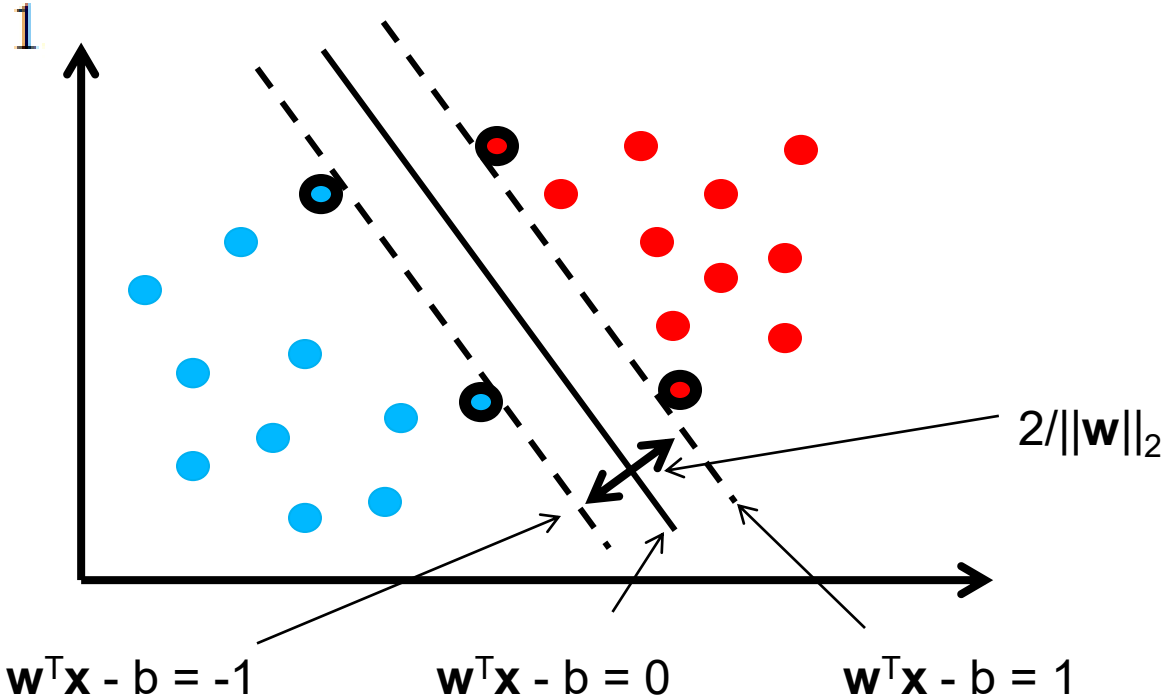
Let's remember how kernel methods were derived from a linear classification method, the Support Vector Machine

Support Vector Machine



Support Vector Machine

We use: $q\|\mathbf{w}\|_2 = 1$



Support Vector Machine

That is, given a set of N samples, each represented by a vector $\mathbf{x}_i \in \mathbb{R}^D$, and the corresponding labels $l_i = \{-1,1\}$ we want to optimize the parameters of $g(\cdot)$ in order to define a discriminant hyperplane discriminating the two classes.

Support Vector Machine (SVM) assumes the data $\mathbf{x}$ is mapped to $\boldsymbol{\varphi}$ using a function $\varphi(\cdot)$

$$\mathbf{x}_i \in \mathbb{R}^D \xrightarrow[\phi(\cdot)]{} \phi_i \in \mathcal{F}$$

Note that the above is a generic mapping. For example a linear function $\varphi(\mathbf{x}) = \mathbf{x}$ can also be used.

Then, we define the decision function $$g(\phi_i) = \mathbf{w}^T \phi_i - b$$

Support Vector Machine

If the parameters of the decision function are optimized, then

$$l_i g(\phi_i) \geq 0 \Rightarrow l_i(\mathbf{w}^T \phi_i - b) \geq 0$$

or

$$\begin{aligned} \mathbf{w}^T \phi_i - b &\geq q, \text{for } l_i = 1 \text{ and} \\ \mathbf{w}^T \phi_i - b &\leq -q, \text{for } l_i = -1. \end{aligned}$$

q expresses the minimal distance between the decision hyperplane and the closest to it training samples. That is, q is the margin appearing between the two classes.

Remember that the decision function expresses distance of a sample from the hyper-plane.

Thus for $\boldsymbol{\varphi}_m$ (the closest point) $\dfrac{l_m g(\phi_m)}{\|\mathbf{w}\|_2} = q$

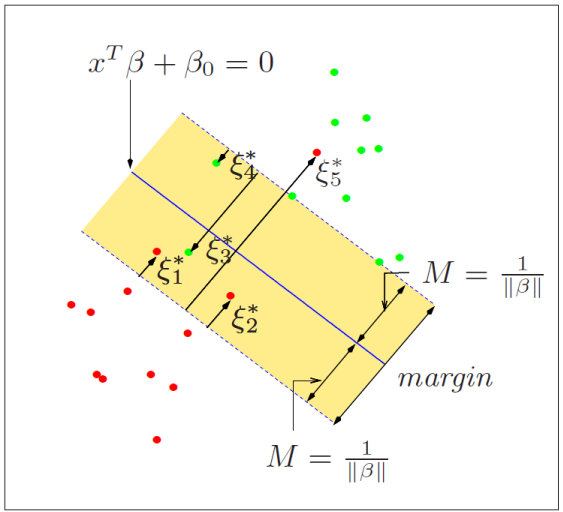
Support Vector Machine

In order to define the weights w and the margin b, SVM optimizes for

$$\mathcal{J}_{SVM} = \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i,$$

subject to the constraints:

$$\begin{aligned} l_i(\mathbf{w}^T\phi_i - b) &\geq 1 - \xi_i,\ i = 1, \ldots, N \\ \xi_i &\geq 0. \end{aligned}$$



Support Vector Machine

To optimize J_{SVM} s.t. the constraints, we define the Lagrangian

$$\mathcal{L} = \frac{1}{2}\mathbf{w}^T\mathbf{w} + c\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\beta_i\xi_i - \sum_{i=1}^{N}\alpha_i[l_i(\mathbf{w}^T\phi_i - b) - 1 + \xi_i]$$

The derivatives of L w.r.t. all variables are

$$\frac{\theta\mathcal{L}}{\theta\mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{N}\alpha_i l_i \phi_i,$$

$$\frac{\theta\mathcal{L}}{\theta b} = 0 \quad \Rightarrow \quad \sum_{i=1}^{N}\alpha_i l_i = 0,$$

$$\frac{\theta\mathcal{L}}{\theta\xi_i} = 0 \quad \Rightarrow \quad c - \alpha_i - \beta_i = 0.$$

Support Vector Machine

If we substitute these equations to L, we obtain

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j l_i l_j \phi_i^T \phi_j + \sum_{i=1}^{N} \alpha_i$$

subject to the constraints

$$0 \leq \alpha_i \leq c, \; i = 1, \ldots, N$$

Support Vector Machine

If we substitute these equations to L, we obtain

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j l_i l_j \phi_i^T \phi_j + \sum_{i=1}^{N} \alpha_i$$

subject to the constraints

$$0 \leq \alpha_i \leq c,\ i = 1, \ldots, N$$

Which can also be written as

$$\max_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^T (\mathbf{l}\mathbf{l}^T \circ \mathbf{K})\boldsymbol{\alpha} + \mathbf{1}^T \boldsymbol{\alpha}$$

where $\mathbf{K} = \boldsymbol{\Phi}^T \boldsymbol{\Phi}$

Support Vector Machine

The problem

$$\max_{\boldsymbol{\alpha}} \ \boldsymbol{\alpha}^T(\mathbf{l}\mathbf{l}^T \circ \mathbf{K})\boldsymbol{\alpha} + \mathbf{1}^T\boldsymbol{\alpha}$$

is a quadratic problem having one global solution when K is positive semi-definite.

After obtaining $\boldsymbol{\alpha}$, $\mathbf{w}$ is calculated by $\mathbf{w} = \sum_{i=1}^{N} \alpha_i l_i \phi_i$

b can be calculated by selecting a training sample for which $\alpha_i > 0$ and computing

$$b = \mathbf{w}^T \phi_i - l_i$$

Kernels

We can use any function κ(⋆,⋆) defined on vector-pairs in order to calculate the elements of a matrix $\mathbf{K}_{ij}$ as long as the resulting matrix **K** is positive semi-definite.

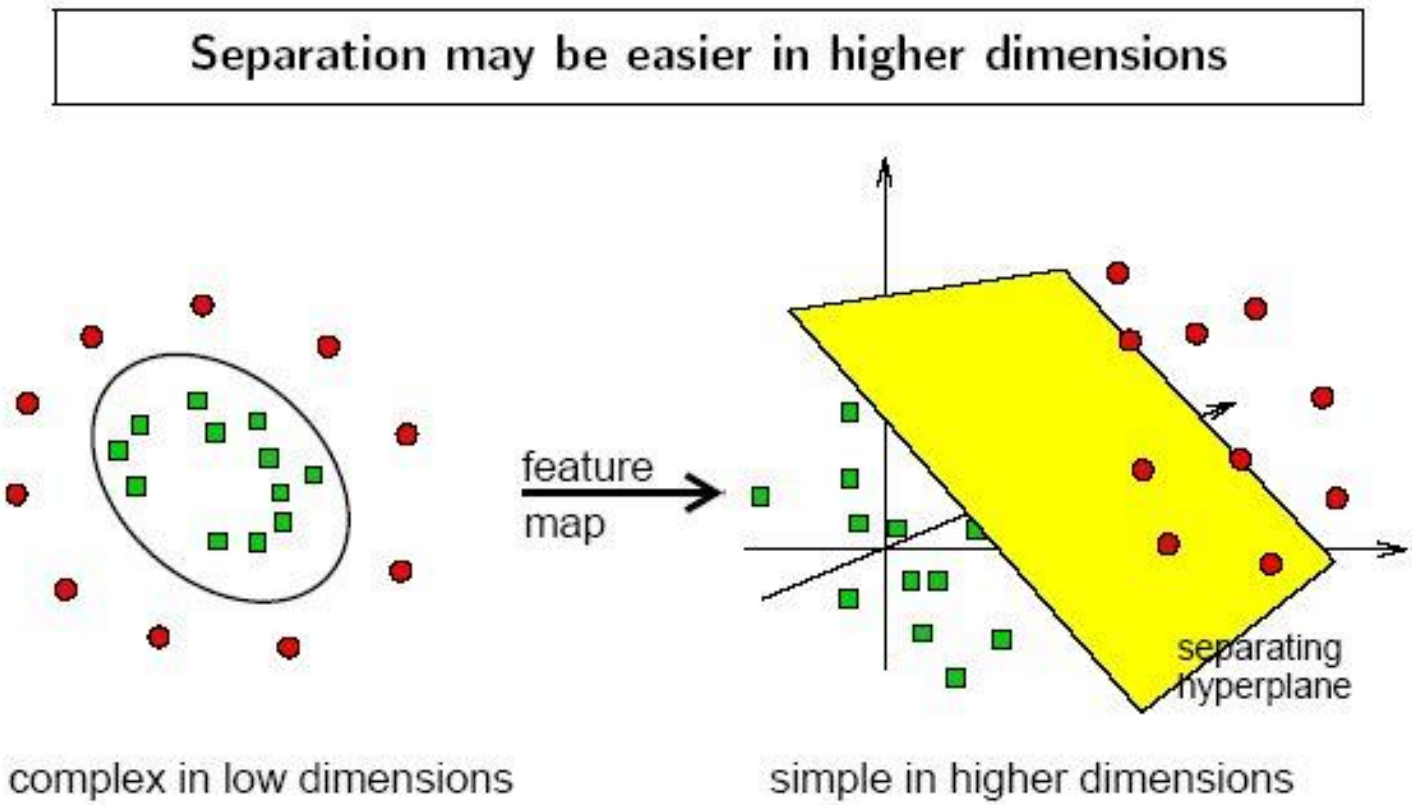
Some example functions are

$$\begin{aligned}\kappa(\mathbf{x}_i, \mathbf{x}_j) &= e^{-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|_2^2} \\ \kappa(\mathbf{x}_i, \mathbf{x}_j) &= (\mathbf{x}_i^T \mathbf{x}_j + 1)^d\end{aligned}$$

Using a generic function κ(⋆,⋆), we have

$$g(\mathbf{x}_*) = \mathbf{w}^T \phi_* - b = \sum_{i=1}^{N} l_i \alpha_i \phi_i^T \phi_* - b = \boldsymbol{\alpha}^T \mathbf{L} \mathbf{k}_* - b$$

Kernels


Separation may be easier in higher dimensions
feature
map
separating
hyperplane
complex in low dimensions
simple in higher dimensions

Kernel Discriminant Analysis

We can also define a linear projection using the data $\phi_i \in \mathcal{F},\ i = 1, \ldots, N$

In this case, the two scatter matrices are

In order to determine the matrix **W**, we optimize for

$$\mathbf{S}_w = \sum_{k=1}^{K} \sum_{i,l_i=k} (\phi_i - \boldsymbol{\mu}_k)(\phi_i - \boldsymbol{\mu}_k)^T \qquad \mathbf{S}_b = \sum_{k=1}^{K} N_k (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T$$

where

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i,l_i=k} \phi_i = \frac{1}{N_k} \boldsymbol{\Phi} \mathbf{1}_k \qquad \boldsymbol{\mu} = \frac{1}{N} \sum_{i=1}^{N} \phi_i = \frac{1}{N} \boldsymbol{\Phi} \mathbf{1}$$

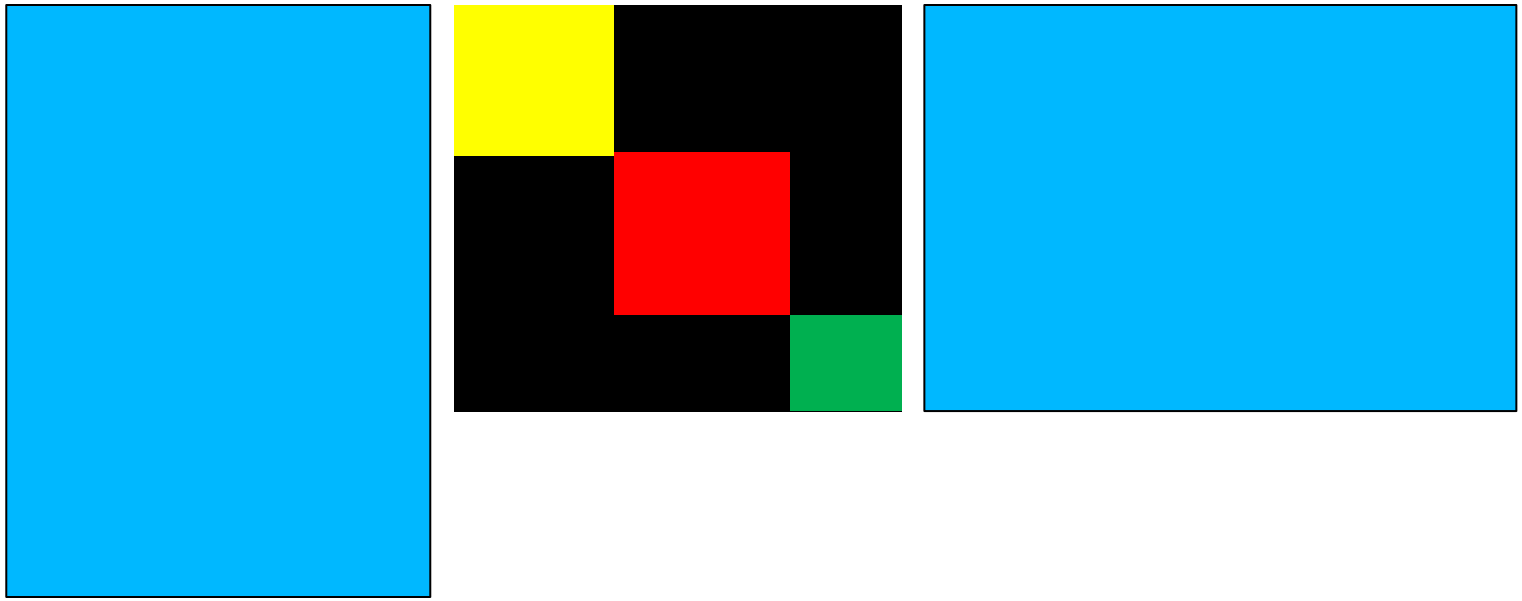
Kernel Discriminant Analysis

Substituting $\boldsymbol{\mu}_k$ and $\boldsymbol{\mu}$ in the scatter matrices, we get

$$
\begin{aligned}
\mathbf{S}_w &= \sum_{k=1}^{K} \left(\boldsymbol{\Phi}\mathbf{J}_k - \frac{1}{N_k}\boldsymbol{\Phi}\mathbf{1}_k\mathbf{1}_k^T\right)\left(\boldsymbol{\Phi}\mathbf{J}_k - \frac{1}{N_k}\boldsymbol{\Phi}\mathbf{1}_k\mathbf{1}_k^T\right)^T \\
&= \sum_{k=1}^{K} \boldsymbol{\Phi}\left(\mathbf{J}_k - \frac{1}{N_k}\mathbf{1}_k\mathbf{1}_k^T\right)\left(\mathbf{J}_k - \frac{1}{N_k}\mathbf{1}_k\mathbf{1}_k^T\right)^T \boldsymbol{\Phi}^T \\
&= \boldsymbol{\Phi}\left(\sum_{k=1}^{K}\left(\mathbf{J}_k - \frac{1}{N_k}\mathbf{1}_k\mathbf{1}_k^T\right)\left(\mathbf{J}_k - \frac{1}{N_k}\mathbf{1}_k\mathbf{1}_k^T\right)^T\right)\boldsymbol{\Phi}^T \\
&= \boldsymbol{\Phi}\mathbf{L}_w\boldsymbol{\Phi}^T,
\end{aligned}
$$

Kernel Discriminant Analysis

$S_W =$ Φ L_W Φ^T



Kernel Discriminant Analysis

Substituting $\boldsymbol{\mu}_k$ and $\boldsymbol{\mu}$ in the scatter matrices, we get

$$
\begin{aligned}
\mathbf{S}_b &= \sum_{k=1}^{K} N_k \left(\frac{1}{N_k} \boldsymbol{\Phi} \mathbf{1}_k \mathbf{1}_k^T - \frac{1}{N} \boldsymbol{\Phi} \mathbf{1} \mathbf{1}_k^T \right) \left(\frac{1}{N_k} \boldsymbol{\Phi} \mathbf{1}_k \mathbf{1}_k^T - \frac{1}{N} \boldsymbol{\Phi} \mathbf{1} \mathbf{1}_k^T \right)^T \\
&= \boldsymbol{\Phi} \left(\sum_{k=1}^{K} N_k \left(\frac{1}{N_k} \mathbf{1}_k \mathbf{1}_k^T - \frac{1}{N} \mathbf{1} \mathbf{1}_k^T \right) \left(\frac{1}{N_k} \mathbf{1}_k \mathbf{1}_k^T - \frac{1}{N} \mathbf{1} \mathbf{1}_k^T \right)^T \right) \boldsymbol{\Phi}^T \\
&= \boldsymbol{\Phi} \mathbf{L}_b \boldsymbol{\Phi}^T .
\end{aligned}
$$

Kernel Graph Embedding

$\mathbf{S}_b =$ $\mathbf{\Phi}$ $\mathbf{L}_b$ $\mathbf{\Phi}^T$

Kernel Graph Embedding

Thus, the optimization criterion becomes

$$\mathcal{J}(\mathbf{W}) = \frac{Tr(\mathbf{W}^T \left(\mathbf{\Phi L}_b \mathbf{\Phi}^T\right) \mathbf{W})}{Tr(\mathbf{W}^T \left(\mathbf{\Phi L}_w \mathbf{\Phi}^T\right) \mathbf{W})}$$

or using $\mathbf{W} = \mathbf{\Phi A}$ (this is the Representer Theorem)

$$\mathcal{J}(\mathbf{A}) = \frac{Tr(\mathbf{A}^T \left(\mathbf{K L}_b \mathbf{K}^T\right) \mathbf{A})}{Tr(\mathbf{A}^T \left(\mathbf{K L}_w \mathbf{K}^T\right) \mathbf{A})} = \frac{Tr(\mathbf{A}^T \mathbf{S}_b^{(A)} \mathbf{A})}{Tr(\mathbf{A}^T \mathbf{S}_w^{(A)} \mathbf{A})}$$

Kernel Graph Embedding

Thus, the optimization criterion becomes

$$\mathcal{J}(\mathbf{W}) = \frac{Tr(\mathbf{W}^T \left(\mathbf{\Phi L}_b \mathbf{\Phi}^T\right) \mathbf{W})}{Tr(\mathbf{W}^T \left(\mathbf{\Phi L}_w \mathbf{\Phi}^T\right) \mathbf{W})}$$

or using $\mathbf{W} = \mathbf{\Phi A}$

$$\mathcal{J}(\mathbf{A}) = \frac{Tr(\mathbf{A}^T \left(\mathbf{K L}_b \mathbf{K}^T\right) \mathbf{A})}{Tr(\mathbf{A}^T \left(\mathbf{K L}_w \mathbf{K}^T\right) \mathbf{A})} = \frac{Tr(\mathbf{A}^T \mathbf{S}_b^{(A)} \mathbf{A})}{Tr(\mathbf{A}^T \mathbf{S}_w^{(A)} \mathbf{A})}$$

which is solved by solving $\mathbf{S}_b^{(A)} \mathbf{a} = \lambda \mathbf{S}_w^{(A)} \mathbf{a}$

Kernel Least-Means Square Regression

We can also define a linear regression using the data $\phi_i \in \mathcal{F},\ i = 1, \ldots, N$

$$\mathbf{W}^T \phi_i = \mathbf{t}_i,\ i = 1, \ldots, N$$

In order to determine the matrix **W**, we optimize for

$$\begin{aligned} \mathcal{J}_{LSE} &= \|\mathbf{W}^T \mathbf{\Phi} - \mathbf{T}\|_F^2 \\ &= Tr\left(\mathbf{W}^T \mathbf{\Phi}\mathbf{\Phi}^T \mathbf{W} - 2\mathbf{W}^T \mathbf{\Phi}\mathbf{T} + \mathbf{T}\mathbf{T}^T\right) \end{aligned}$$

where $\mathbf{\Phi} = [\phi_1, \ldots, \phi_N]$, $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_N]$ and $Tr(\cdot)$ is the trace operator of a matrix.

Kernel Least-Means Square Regression

Setting the derivative w.r.t. **W** equal to zero, we have

$$\nabla \mathcal{J}_{LSE} = 0 \Rightarrow 2\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{W} = 2\mathbf{\Phi}\mathbf{T}^T$$

leading to

$$\mathbf{W} = \left(\mathbf{\Phi}\mathbf{\Phi}^T\right)^{-1}\mathbf{\Phi}\mathbf{T}^T = \mathbf{\Phi}^\dagger\mathbf{T}^T$$

when the mapping $\mathbf{x} \rightarrow \boldsymbol{\varphi}$ is defined, we can use the above equation to calculate $\mathbf{W}$.

Kernel Least-Means Square Regression

When the mapping $\mathbf{x} \rightarrow \boldsymbol{\varphi}$ is defined through the function $\kappa(\cdot,\cdot)$, we express W as a linear combination of the training samples

$$\mathbf{W} = \boldsymbol{\Phi}\mathbf{A}$$

Substituting W to J_{LSE}, we obtain

$$\begin{aligned} \mathcal{J}_{LSE} &= \|\mathbf{A}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi} - \mathbf{T}\|_F^2 = \|\mathbf{A}^T\mathbf{K} - \mathbf{T}\|_F^2 \\ &= Tr\left(\mathbf{A}^T\mathbf{K}\mathbf{K}^T\mathbf{A} - 2\mathbf{A}^T\mathbf{K}\mathbf{T} + \mathbf{T}\mathbf{T}^T\right) \end{aligned}$$

Kernel Least-Means Square Regression

When the mapping $\mathbf{x} \rightarrow \boldsymbol{\varphi}$ is defined through the function $\kappa(\cdot,\cdot)$, we express W as a linear combination of the training samples

$$\mathbf{W} = \boldsymbol{\Phi}\mathbf{A}$$

Substituting W to J_{LSE}, we obtain

$$\begin{aligned} \mathcal{J}_{LSE} &= \|\mathbf{A}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi} - \mathbf{T}\|_F^2 = \|\mathbf{A}^T\mathbf{K} - \mathbf{T}\|_F^2 \\ &= Tr\left(\mathbf{A}^T\mathbf{K}\mathbf{K}^T\mathbf{A} - 2\mathbf{A}^T\mathbf{K}\mathbf{T} + \mathbf{T}\mathbf{T}^T\right) \end{aligned}$$

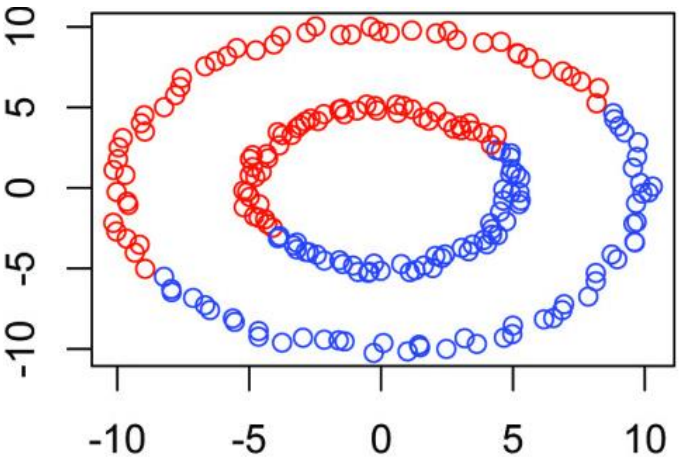
Setting the derivative to zero we have $\nabla\mathcal{J}_{LSE} = 0 \Rightarrow 2\mathbf{K}\mathbf{K}^T\mathbf{A} = 2\mathbf{K}\mathbf{T}^T$

$$\mathbf{A} = \left(\mathbf{K}\mathbf{K}^T\right)^{-1}\mathbf{K}\mathbf{T}^T = \mathbf{K}^\dagger\mathbf{T}^T$$

Kernel-based Clustering

K-Means: Define clusters by minimizing the scatter of samples from the cluster centers:

$$\min_{C,\{m_k\}_1^K} \sum_{k=1}^{K} N_k \sum_{C(i)=k} ||x_i - m_k||^2$$



Algorithm 14.1 *K-means Clustering.*

1. For a given cluster assignment C, the total cluster variance (14.33) is minimized with respect to $\{m_1, \ldots, m_K\}$ yielding the means of the currently assigned clusters (14.32).

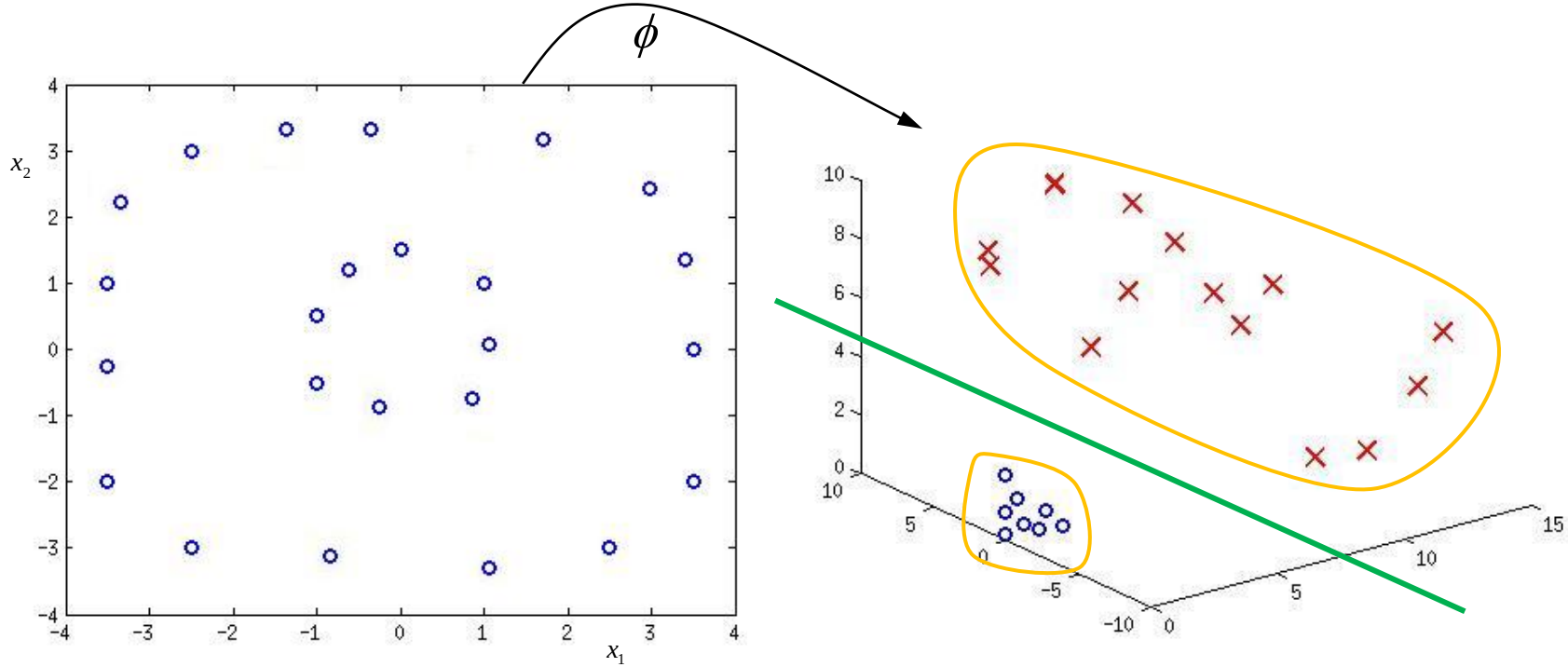
2. Given a current set of means $\{m_1, \ldots, m_K\}$, (14.33) is minimized by assigning each observation to the closest (current) cluster mean. That is,

$$C(i) = \underset{1 \le k \le K}{\operatorname{argmin}} ||x_i - m_k||^2. \tag{14.34}$$

3. Steps 1 and 2 are iterated until the assignments do not change.

Kernel-based Clustering

Kernel K-Means: map the data to a new feature space, where k-Means works well



$$\text{Polynomial kernel } K(x, y) = (x' y)^2$$

$$\phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

Kernel-based Clustering

Kernel K-Means:

- Samples are mapped to the kernel space → the cluster mean vectors are given by:

$$\mathbf{m}_c = \frac{\sum_{a_j \in \pi_c} \phi(\mathbf{x}_j)}{|\pi_c|}$$

Number of samples in cluster c

- We will use the Euclidean distance in the kernel space:

$$D(\mathbf{x}_i, \mathbf{m}_c) = ||\phi(\mathbf{x}_i) - \mathbf{m}_c||^2$$

$$D(\mathbf{x}_i, \mathbf{m}_c) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_i) - 2\phi(\mathbf{x}_i)^T \mathbf{m}_c + \mathbf{m}_c^T \mathbf{m}_c$$

Kernel-based Clustering

Kernel K-Means:

- Samples are mapped to the kernel space → the cluster mean vectors are given by:

$$\mathbf{m}_c = \frac{\sum_{a_j \in \pi_c} \phi(\mathbf{x}_j)}{|\pi_c|}$$

Number of samples in cluster c

- We will use the Euclidean distance in the kernel space:

$$D(\mathbf{x}_i, \mathbf{m}_c) = ||\phi(\mathbf{x}_i) - \mathbf{m}_c||^2 \Rightarrow$$

$$D(\mathbf{x}_i, \mathbf{m}_c) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_i) - 2\phi(\mathbf{x}_i)^T \mathbf{m}_c + \mathbf{m}_c^T \mathbf{m}_c \Rightarrow$$

$$D(\mathbf{x}_i, \mathbf{m}_c) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_i) - 2\frac{\sum_{a_j \in \pi_c} \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)}{|\pi_c|} + \frac{\sum_{a_j \in \pi_x} \sum_{a_l \in \pi_x} \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_l)}{|\pi_c|^2} \Rightarrow$$

$$D(\mathbf{x}_i, \mathbf{m}_c) = K_{ii} - 2\frac{\sum_{a_j \in \pi_c} K_{ij}}{|\pi_c|} + \frac{\sum_{a_j \in \pi_c} \sum_{a_l \in \pi_c} K_{jl}}{|\pi_c|^2}$$

Kernel-based Clustering

Kernel K-Means:

- Apply K-Means algorithm by replacing the distance function with:

$$D(\mathbf{x}_i, \mathbf{m}_c) = K_{ii} - 2\frac{\sum_{a_j \in \pi_c} K_{ij}}{|\pi_c|} + \frac{\sum_{a_j \in \pi_c} \sum_{a_l \in \pi_c} K_{jl}}{|\pi_c|^2}$$

- The above distance function is also used to assign a new vector x_i to a cluster (during testing)

Nonlinear Projection Trick

Let us assume that, given a set of vectors xi, i=1,…,N, we have calculated a kernel matrix $K \in \mathbb{R}^{NxN}$ (e.g. by using the RBF kernel function). Then:

- Since K is positive semi-definite it can be written as $K = U \Sigma U^T = Z^T Z$
- Using the columns of Z as data representations (e.g. sample 1 is represented by z_1, the 1st column of Z), the application of a linear method (e.g. linear regression) is equivalent to applying the kernel method using K.

Nonlinear Projection Trick

Let us assume that, given a set of vectors xi, i=1,…,N, we have calculated a kernel matrix $K \in \mathbb{R}^{NxN}$ (e.g. by using the RBF kernel function). Then:

- Since K is positive semi-definite it can be written as $K = U \Sigma U^T = Z^T Z$
- Using the columns of Z as data representations (e.g. sample 1 is represented by z_1, the 1st column of Z), the application of a linear method (e.g. linear regression) is equivalent to applying the kernel method using K.

Example:

The above shows that kernel K-Means is equivalent to Spectral clustering when using the same kernel/affinity function!

Computational complexity of kernel methods

In order to apply a kernel-based learning method:

- calculate and store the kernel matrix, with time and space complexities of $O(N^2)$
- Apply a matrix inversion/eigenanalysis, with time complexity of the order of $O(N^3)$, e.g. in kernel-based regression:

$$\mathbf{A} = \left(\mathbf{K}\mathbf{K}^T\right)^{-1}\mathbf{K}\mathbf{T}^T = \mathbf{K}^\dagger\mathbf{T}^T$$

The above make the application of kernel-based methods in large-scale problems intractable → we need some approximating schemes!

Nyström Approximation

Nyström method defines an approximation of the kernel matrix K:

$$\mathbf{K} \approx \tilde{\mathbf{K}} = \mathbf{C}\mathbf{C}^T$$

where $C \in \mathbb{R}^{N \times n}$ with $n << N$.

Using such an approximation, computations and memory can be highly reduced:

$$\mathbf{A} \simeq \left(\tilde{\mathbf{K}} + \delta\mathbf{I}\right)^{-1} \mathbf{T}^T = \frac{1}{\delta}\left[\mathbf{I} - \mathbf{C}\,\underline{\left(\delta\mathbf{I} + \mathbf{C}^T\mathbf{C}\right)}^{-1}\mathbf{C}^T\right]\mathbf{T}^T$$

matrix $C^TC \in \mathbb{R}^{n \times n}$

Nyström Approximation

Eigenanalysis of K gives: $$\mathbf{K} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$$

Where $U \in \mathbb{R}^{N \times N}$ is the matrix of eigenvectors and $\Lambda \in \mathbb{R}^{N \times N}$ is the (diagonal) matrix of eigenvalues.

Thus, keeping the n largest eigenvalues Λ_n and the corresponding eigenvectors U_n:

$$\mathbf{C} = \mathbf{U}_n \mathbf{\Lambda}_n^{\frac{1}{2}}$$

Nyström Approximation

Eigenanalysis of K gives: $$\mathbf{K} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$$

Where $U \in \mathbb{R}^{N \times N}$ is the matrix of eigenvectors and $\Lambda \in \mathbb{R}^{N \times N}$ is the (diagonal) matrix of eigenvalues.

Thus, keeping the n largest eigenvalues Λ_n and the corresponding eigenvectors U_n:

$$\mathbf{C} = \mathbf{U}_n \mathbf{\Lambda}_n^{\frac{1}{2}}$$

Two problems:

- We need to calculate and store K in memory
- Eigenanalysis of K has also a time complexity of the order of $O(N^3)$

Nyström Approximation

Nyström uses a sub-matrix of K formed by n columns $K_{Nn} \in \mathbb{R}^{N \times n}$ and the submatrix of K formed by the same columns and rows $K_{nn} \in \mathbb{R}^{n \times n}$ (note that we don't need to calculate K)

By applying eigenanalysis to K_{nn}, we obtain: $\mathbf{K}_{nn} = \mathbf{U}_{(n)} \mathbf{\Lambda}_{(n)} \mathbf{U}_{(n)}^T$

The matrix containing the n leading eigenvectors (those corresponding to the maximal eigenvalues) of K is given by: $\mathbf{U}_n \approx \mathbf{K}_{Nn} \mathbf{U}_{(n)} \mathbf{\Lambda}_{(n)}^{-1}$

and the rank-n approximation of K is given by: $\mathbf{K} \approx \tilde{\mathbf{K}} = \mathbf{K}_{Nn} \mathbf{K}_{nn}^{-1} \mathbf{K}_{Nn}^T = \mathbf{G}\mathbf{G}^T$
where $\mathbf{G} = \mathbf{K}_{Nn} \mathbf{K}_{nn}^{-\frac{1}{2}}$

When the actual rank of K is n, the above approximation is exact.

Nyström-based Nonlinear Projection Trick

The low-rank approximation of K obtained by Nyström method can be used for the determination of a subspace of the kernel space

$$\tilde{\mathbf{y}} = \mathbf{\Lambda}_{(n)}^{-\frac{1}{2}} \mathbf{\Lambda}_n^{-1} \mathbf{U}_n^T \mathbf{K}_{Nn}^T \mathbf{k} = \mathbf{W}^T \phi(\mathbf{x})$$

where $\mathbf{W} = \mathbf{\Phi} \mathbf{K}_{Nn} \mathbf{U}_n \mathbf{\Lambda}_n^{-1} \mathbf{\Lambda}_{(n)}^{-\frac{1}{2}}$

Non-linear regression

Random Fourier kernels: Define vectors $z(x_i)$ such that the dot product between pairs of vectors $z(x_i)^T z(x_j) \approx K_{ij}$

Process:

- Apply a nonlinear mapping $x_i \rightarrow z(x_i)$ using random weights and Fourier functions
- Then, apply linear regression using $z(x_i)$'s:

Algorithm 1 Random Fourier Features.

Require: A positive definite shift-invariant kernel $k(\mathrm{x}, \mathrm{y}) = k(\mathrm{x} - \mathrm{y})$.

Ensure: A randomized feature map $\mathrm{z}(\mathrm{x}) : \mathcal{R}^d \rightarrow \mathcal{R}^D$ so that $\mathrm{z}(\mathrm{x})'\mathrm{z}(\mathrm{y}) \approx k(\mathrm{x} - \mathrm{y})$.

Compute the Fourier transform p of the kernel k: $p(\omega) = \frac{1}{2\pi} \int e^{-j\omega'\delta} k(\delta)\, d\Delta$.

Draw D iid samples $\omega_1, \cdots, \omega_D \in \mathcal{R}^d$ from p and D iid samples $b_1, \ldots, b_D \in \mathcal{R}$ from the uniform distribution on $[0, 2\pi]$.

Let $\mathrm{z}(\mathrm{x}) \equiv \sqrt{\frac{2}{D}} \left[\cos(\omega_1'\mathbf{x}+b_1) \ \cdots \ \cos(\omega_D'\mathbf{x}+b_D) \right]'$.