Data Mining

Predictive Modelling

Linear classification models

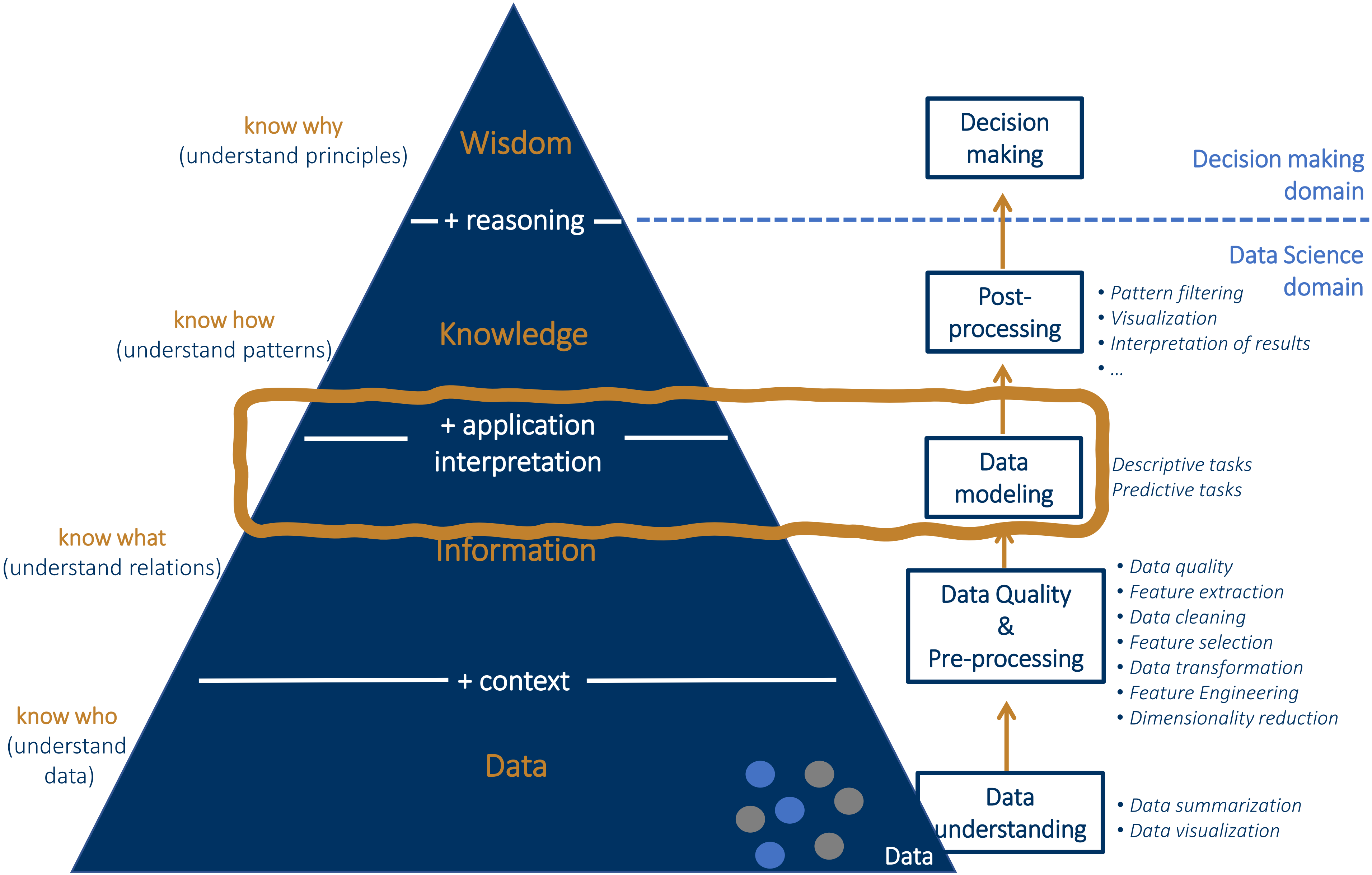
Raquel Sebastião

Departamento de Eletrónica, Telecomunicações e Informática

Universidade de Aveiro

raquel.sebastiao@ua.pt

2022/2023


know why
(understand principles)
Wisdom
+ reasoning
know how
(understand patterns)
Knowledge
+ application
interpretation
know what
(understand relations)
Information
+ context
know who
(understand
data)
Data
Data
Decision
making
Decision making
domain
Data Science
domain
Post-
processing
• Pattern filtering
• Visualization
• Interpretation of results
• ...
Data
modeling
Descriptive tasks
Predictive tasks
Data Quality
&
Pre-processing
• Data quality
• Feature extraction
• Data cleaning
• Feature selection
• Data transformation
• Feature Engineering
• Dimensionality reduction
Data
understanding
• Data summarization
• Data visualization

Contents

Predictive problems

Classification problems

Id	Sepal length	Sepal width	Petal length	Petal width	Type
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
51	7	3.2	4.7	1.4	versicolor
101	6.3	3.3	6	2.5	virginica
102	5.8	2.7	5.1	1.9	virginica

Target: class label (nominal var.)

Regression problems

Id	MSSubClass	LotArea	LotConfig	Neighborhood	Condition1	(...)	YearBuilt	Exterior1st	SalePrice
1	60	8450	Inside	CollgCr	Norm	(...)	2003	VinylSc	208500
67	70	9550	Corner	Crawfor	Norm	(...)	1915	Wd Sdng	140000
106	50	14115	Inside	Mitchel	Norm	(...)	1993	VinylSc	143000
208	60	10382	Corner	NWAmes	PosN	(...)	1973	HdBoard	200000
386	50	6120	Inside	OldTown	Artery	(...)	1931	BrkFace	129900

Target: numeric (numerical var.)

Predictive problems

Classification vs Regression

- **Classification** assigns input vector **x** to one of k discrete classes C_k , k = 1, . . . ,K
  - Common classification scenario: classes considered disjoint
    - Each input assigned to only one class
  - Input space is thereby divided into decision region

- **Regression** assigns input vector **x** to one or more continuous target variables **t**
  - Linear regression has simple analytical and computational properties

Classification: problem definition

Setting

- Given a training data set $\{(x_i, y_i)\}_{i=1}^{N}$, where each object is represented by a **D+1**–tuple: (D–dim) feature vector $\mathbf{x}_i \in \mathbb{R}^D$ and the corresponding label $y_i \in \boldsymbol{Y}$
- There is an **unknown** function: $Y = f(X)$

Goal

Learn the model that yields the best approximation of the unknown function ***fF*()**

Approach

- Assume a functional form $\boldsymbol{h_\theta}$**(x)** for the unknown function $\boldsymbol{F}$**()**, where $\boldsymbol{\theta}$ are a set of parameters
- Assume a preference criterion over the space $\boldsymbol{\theta}$ of possible parameterizations of *h*()
- Search for the "best" $\boldsymbol{h_\theta}$ (according to the criterion and the data set)

Classification: learn/build a prediction model

Given a training set $\{(\mathrm{x}_i, \mathrm{y}_i)\}_{i=1}^{N}$:

- $(\mathrm{x}_i, \omega_{c_i})$ - the label is **symbolic** or **categorical** (ex.: binary {malignant, benign})
- (x_i, t_i) - label is **numerical** (ex.: binary {−1, 1}, multiclass {0, 1, 2}))

Learning/Training phase

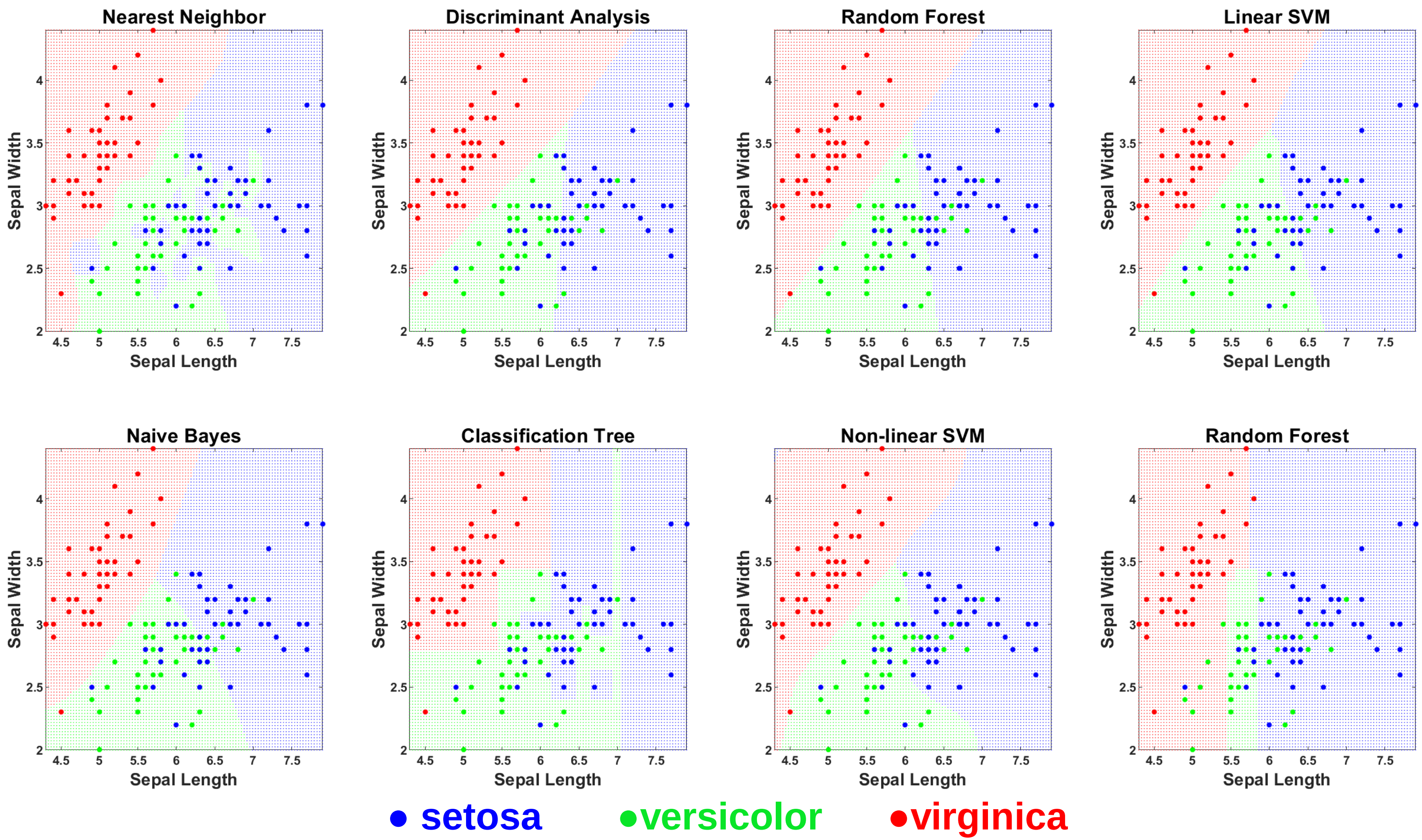
- find the "best" approximation to ***F***, $\boldsymbol{h_\theta}$

Testing phase: Given a test set (data not included in the training set)

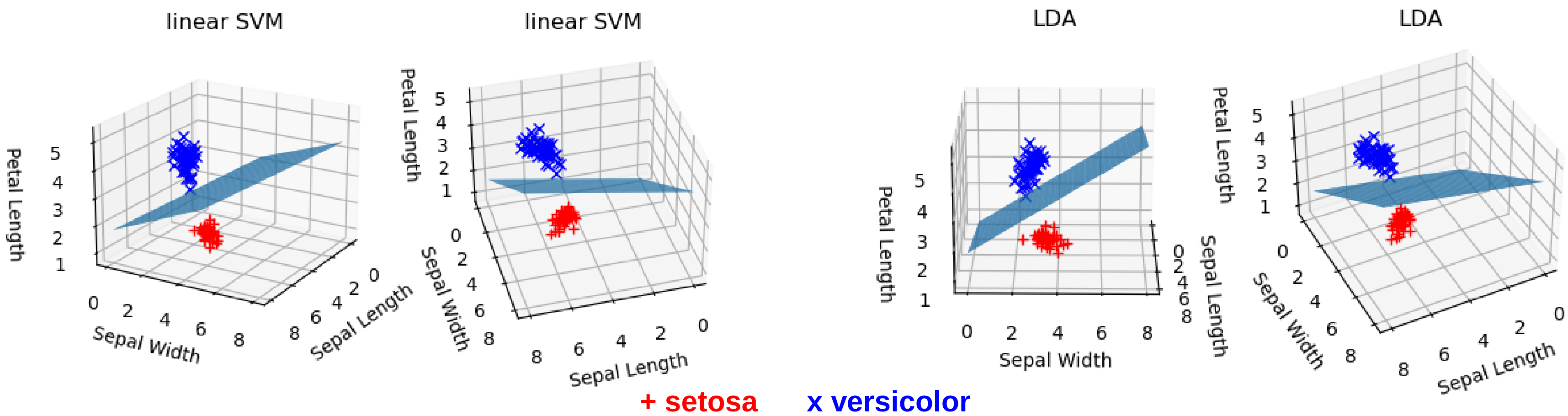
- Study the performance of the model

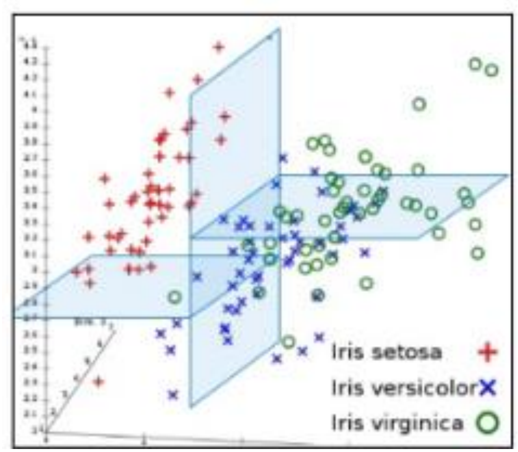
Usually, the available data set is divided into training and test sets

Classification: decision surfaces



Classification: decision surfaces





Prediction Models – approaches

Geometric approaches

- **Distance-based:** kNN
- **Linear models:** Fisher's linear discriminant, perceptron, logistic regression, SVM (w. linear kernel)

Probabilistic approaches

- naive Bayes, logistic regression

Logical approaches

- classification or regression trees, rules

Optimization approaches

- neural networks, SVM

Sets of models (ensembles)

- random forests, adaBoost

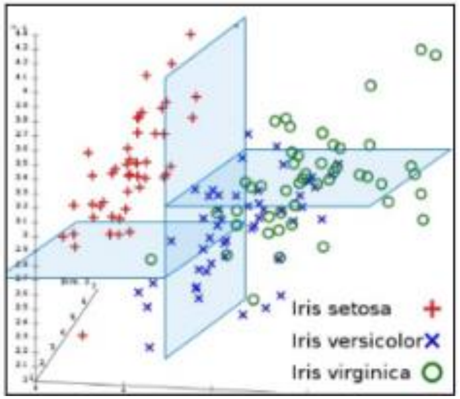
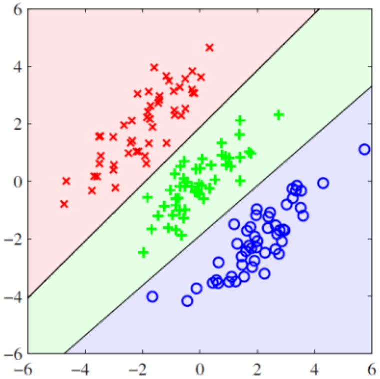
Contents

- Predictive modelling
- **Linear classification models**
  - Discriminant Functions
    - Least squares for classification
    - Fisher’s Linear Discriminant (LDA)
    - Perceptron
  - Logistic Regression
  - Comparison
- Iterative optimization
- Summary

Linear classification models

Decision surfaces are **linear functions** of input **x**

- Defined by (D −1)-dim hyperplanes within D-dim input space

3-class problem **3-D feature vector**	**3-class problem** **2-D feature vector**
	
A plane is 2-D surface in 3-D space	Straight line is 1-D decision boundary in 2-D space

- Data sets whose classes can be separated exactly by **linear decision surfaces** are said to be linearly separable

Linear classification models

Discriminant functions

- map feature vector $\mathbf{x}$ directly into decisons
- probabilities play no role
  - Least Squares for classification
  - Fisher’s linear discriminant
  - Perceptron

Probabilistic approaches

- The inference and decision are taken in separated stages
  - Logistic regression (discriminative model)

Support Vector Machines (linear SVM)

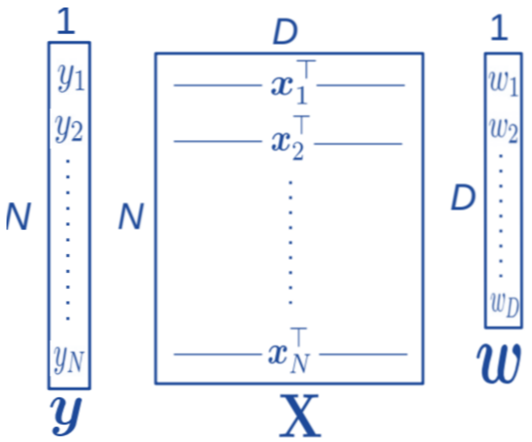
Contents

- Predictive modelling
- **Linear classification models**
  - **Discriminant Functions**
    - Least squares for classification
    - Fisher's Linear Discriminant (LDA)
    - Perceptron
  - Logistic Regression
  - Comparison
- Iterative optimization
- Summary

Linear discriminant functions

- A **discriminant function** assigns the D−dim feature vector

$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \dots \mathbf{x}_D \end{bmatrix}^T \in \mathbb{R}^D$ to one of the ***k*** classes denoted by $\mathbf{C}_k$



- **Linear discriminant functions**

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

$g(\mathbf{x})$ is linear combination of feature vector $\mathbf{x}$

  - $\mathbf{w}$ is the **weighted vector** ($\mathbf{w} = \begin{bmatrix} w_1 & w_2 & \dots w_D \end{bmatrix}^T$)
  - w_0 is the ***bias*** *(threshold = - bias = -* w_0)

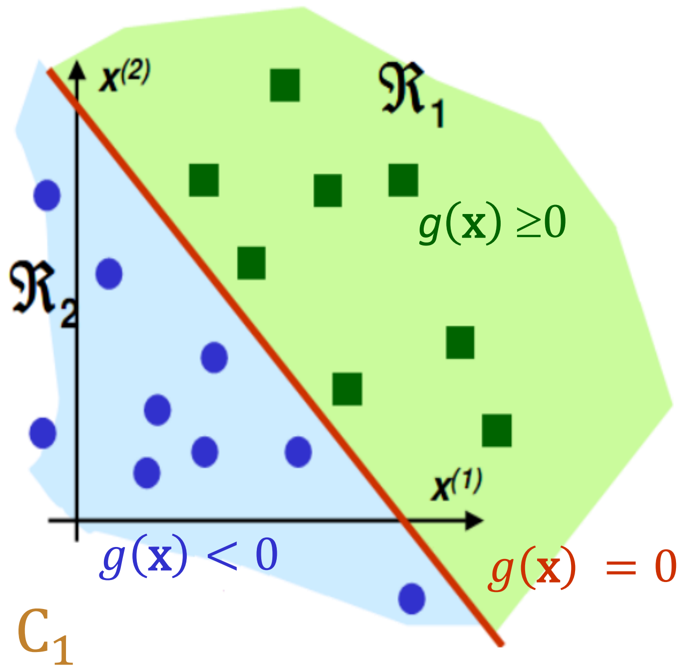
- Decision surfaces are hyperplanes

Linear discriminant functions: 2-class

- Linear discriminant functions

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

  - $\mathbf{w}$ is the **weighted vector** ($\mathbf{w} = \begin{bmatrix} w_1 & w_2 & \dots w_D \end{bmatrix}^T$)
  - w_0 is the ***bias*** *(threshold = - bias = -* w_0)



- Decision surface: $g(\mathbf{x}) = 0$
  - (D-1)-dimensional hyperplane within the D-dimensional feature vector

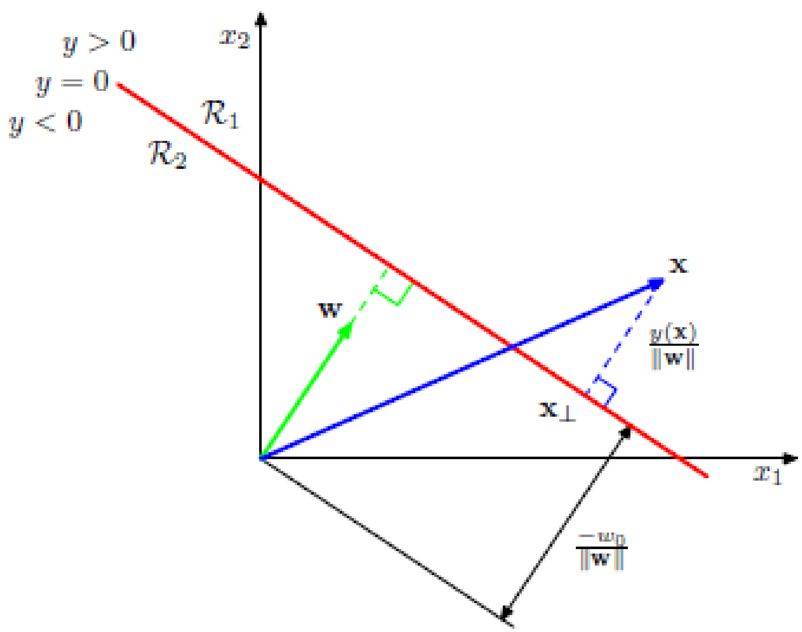
- Classification

$$g(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 \Rightarrow \begin{cases} g(\mathbf{x}) \geq 0, \ \mathbf{x} \in C_1 \\ g(\mathbf{x}) < 0, \ \mathbf{x} \in C_2 \end{cases}$$

Linear discriminant functions: 2-class

- Classification

$$g(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 \Rightarrow \begin{cases} y(\mathbf{x}) > \mathbf{0},\ \mathbf{x} \in \mathrm{C}_1 \\ y(\mathbf{x}) < \mathbf{0},\ \mathbf{x} \in \mathrm{C}_2 \end{cases}$$



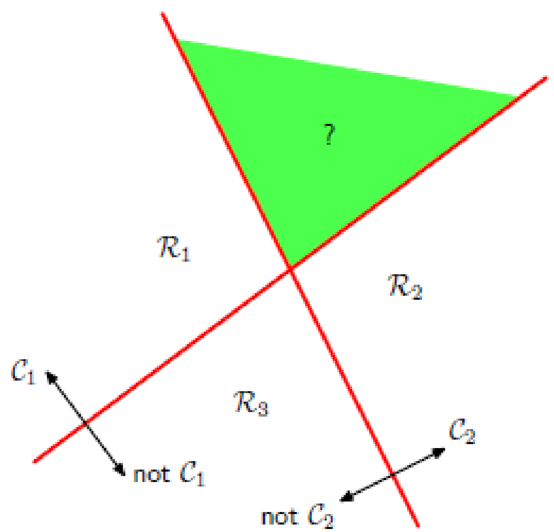
Decision surface: $g(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0$

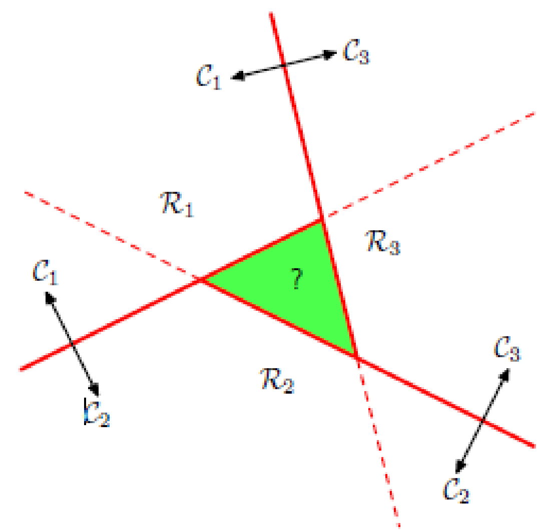
- **orientation** in space is given by its normal **w** ($\mathbf{w} \perp$ decision boundary)
- **position** in space is given by its distance $\frac{w_0}{\|\mathbf{w}\|}$ to the origin
- Distance of an object $\mathbf{x}_a$ (with label t) to the decision surface $\frac{tg(\mathbf{X})}{\|\mathbf{W}\|}$

Linear discriminant functions: multi-class

Two approaches

- Using several two-class classifiers
  - But leads to serious difficulties
- Use ***k*** linear discriminant functions





Linear discriminant functions: learning $\mathbf{w}$

Linear discriminant function

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

Given a **training data set** $(\mathbf{x}, y)$

- (D–dim) feature vector $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots x_D \end{bmatrix}^T \in \mathbb{R}^D$
- corresponding label $g(\mathbf{x}) \in \boldsymbol{Y}$

How to learn $\mathbf{w}$*???*

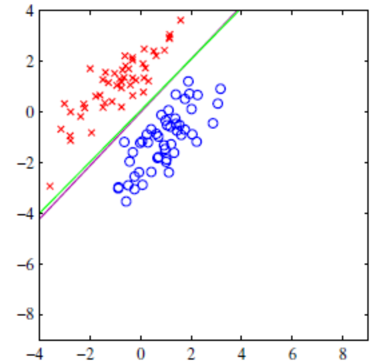
- Least Squares for classification
- Fisher's Linear Discriminant
- Perceptrons

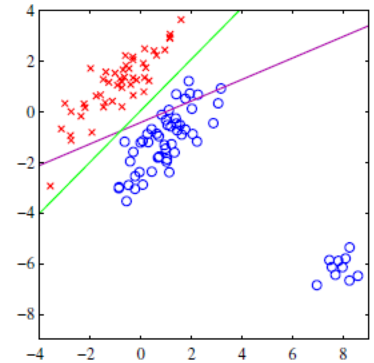
Contents

Least squares for classification

Linear discriminant function

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





decision boundary:
- least squares
- logistic regression

Pure geometric-based model

Goal:

- make the model predictions as close as possible to a set of target values
- Find parameters $\mathbf{w}$ that minimizes the Sum of Square Error (or **maximizing the likelihood function** under conditional Gaussian distribution)
- **Highly sensitive to outliers**

Contents

- Predictive modelling
- **Linear classification models**
  - Discriminant Functions
    - Least squares for classification
    - Fisher's Linear Discriminant (LDA)
    - Perceptron
  - Logistic Regression
  - Comparison
- Iterative optimization
- Summary

Fisher's linear discriminant

Linear discriminant function

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

Basic idea:

- View classification in terms of dimensionality reduction
  - Project D-dim feature vector $\mathbf{x}$ into 1-dim using $\mathbf{w}^{\mathrm{T}}\mathbf{x}$

- The goal is to find $\mathbf{w}$ (**direction**) to project the data such that:
  - It maximizes the distance between the means of each class (maximizes separability among class)
  - It minimizes the variance within each class

Linear classifier

- Place *threshold* on $g(\mathbf{x})$ to classify $y \geq w_0$ as C_1 and otherwise C_2

Fisher's linear discriminant

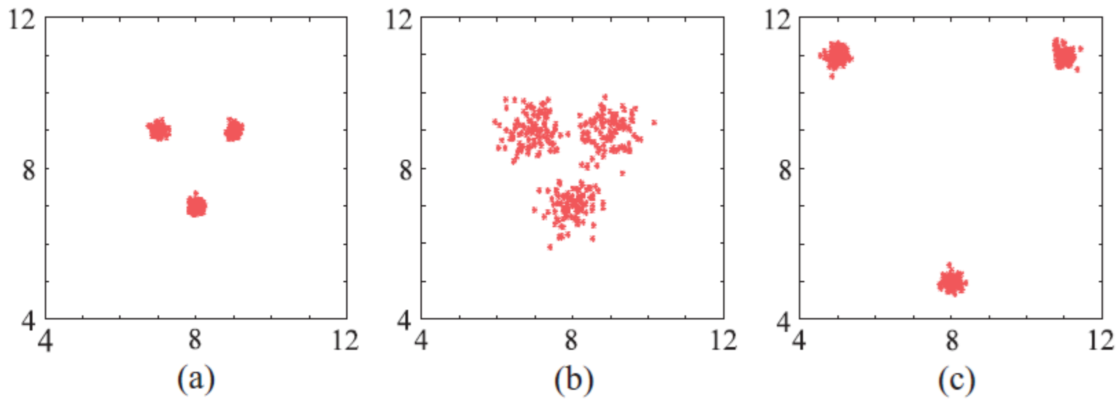
Linear discriminant function

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

The goal is to find $\mathbf{w}$ (**direction**) to project the data such that:

- It maximizes the distance between the means of each class (maximizes separability among class)
- It minimizes the variance within each class

Example: consider the data as belonging to three classes and projected into two directions



- **(c) is the best solution: large distance between classes and small variance within classes**

Fisher's linear discriminant

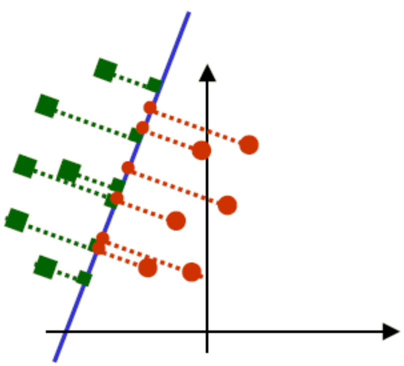
$$g(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0$$

The goal is to find directions to project the data such that:

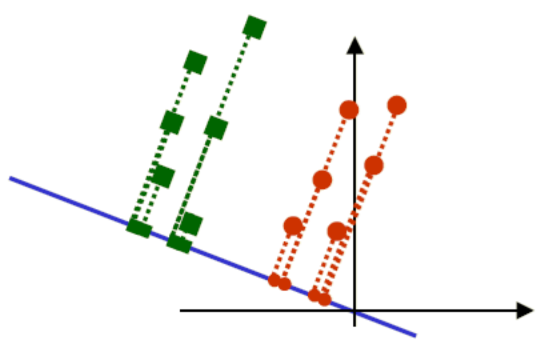
- **Large distance** between classes
- **Small variance** within class

Example: Suppose we have 2 classes and 2-dimensional samples

- find projection to a line s.t. samples from different classes are well separated



bad line to project to
(classes are mixed up)



good line to project to
(classes are well separated)

Fisher's linear discriminant: 2-class problems

Suppose we have 2 classes and 2-dimensional samples $X_1, X_2, \ldots, X_N$:

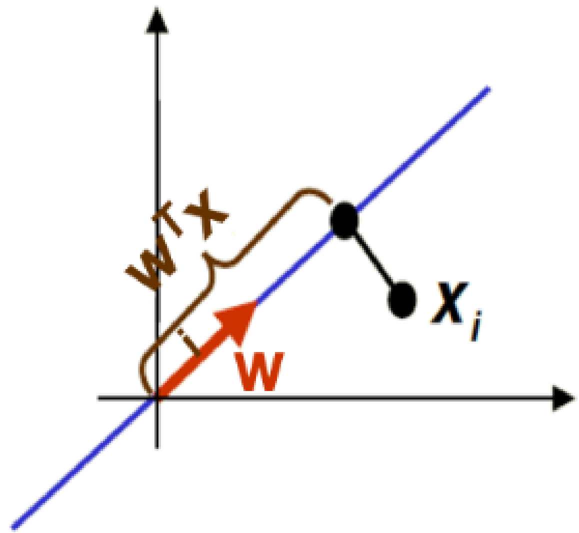
- N_1 samples come from **class 1**
- N_2 samples come from **class 2**

Basic ideia: Project all samples on a line s.t. different classes are well separated

- Let the **line direction be given by unit vector w**

- $\mathbf{w}^T\mathbf{x}_i$ is the distance of projection of $\mathbf{x}_i$ from the origin

- Thus $\mathbf{z}_i = \mathbf{w}^T\mathbf{x}_i$ is the projection of $\mathbf{x}_i$ into a one dimensional subspace



Fisher's linear discriminant: learning $\mathbf{w}$ 2-class problems

- The projection of $\mathbf{x}_i$ into a one dimensional subspace with direction $\mathbf{w}$ is given by $\mathbf{z}_i = \mathbf{w}^T\mathbf{x}_i$

How to measure separation between projections of different classes?

- Consider that μ_1 and μ_2 are the means of class 1 and 2 (in the original space)
- Let $\tilde{\mu}_1$ and $\tilde{\mu}_2$ be the means of projections of classes 1 and 2
- thus $|\tilde{\mu}_1 - \tilde{\mu}_2|$ seems like a **good measure:**

$$\tilde{\mu}_c = \frac{1}{N_c}\sum_{i\in\omega_c}\mathbf{w}^T\mathbf{x}_i = \mathbf{w}^T\left(\frac{1}{N_c}\sum_{i\in\omega_c}\mathbf{x}_i\right) = \mathbf{w}^T\mu_c$$

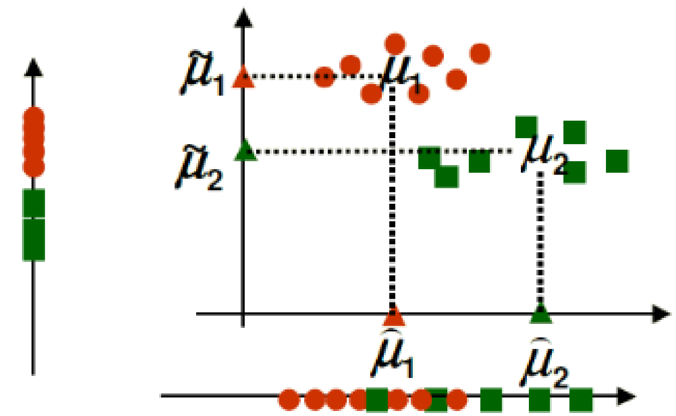
where N_c is the number of samples of class ω_c $(c = 1, 2)$

Fisher's linear discriminant: learning $\mathbf{w}$ 2-class problems

How good is $|\tilde{\mu}_1 - \tilde{\mu}_2|$ as a measure of separation?

- The larger $|\tilde{\mu}_1 - \tilde{\mu}_2|$ the better is the expected separation

- Consider the two directions for projecting the following data:
  - $\mathbf{w} = (0, 1)$
  - $\mathbf{w} = (1, 0)$



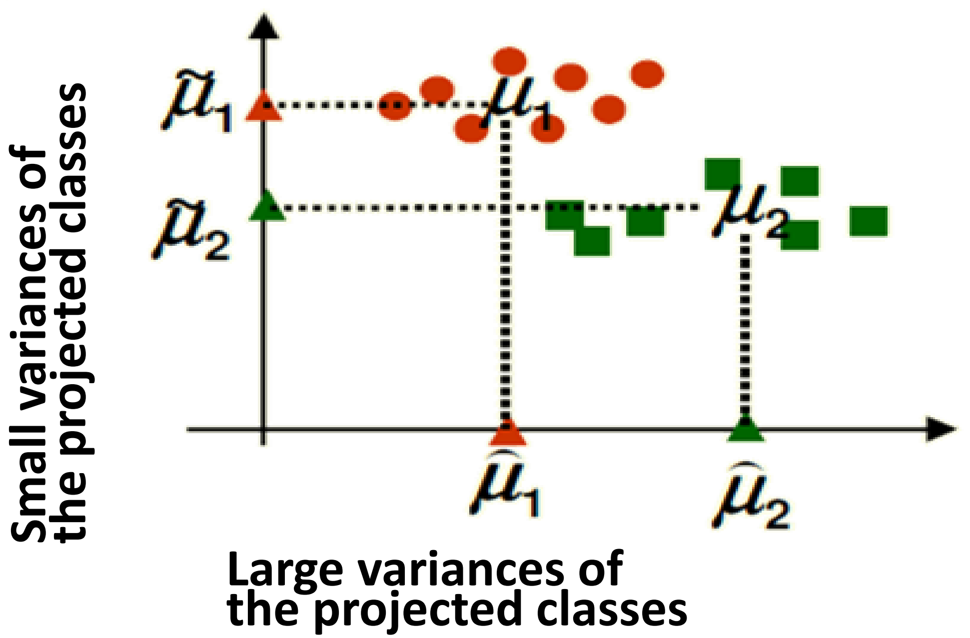
- The distance between the means of the projected data is **higher** in the horizontal axes $\Rightarrow$ **Large distance** between classes

However the vertical axes is a better line to project attending the class separability

Fisher's linear discriminant: learning $\mathbf{w}$
2-class problems

How good is $|\tilde{\mu}_1 - \tilde{\mu}_2|$ as a measure of separation?

- Problem: $|\tilde{\mu}_1 - \tilde{\mu}_2|$ does not consider the variance of the classes!!!



Ideally, the projected classes have both **faraway means** and **small variances** !!

Fisher's linear discriminant: learning w 2-class problems

Problem: $|\tilde{\mu}_1 - \tilde{\mu}_2|$ does not consider the variance of the classes!!!

Thus, $|\tilde{\mu}_1 - \tilde{\mu}_2|$ needs to be **normalized** by a factor proportional to **variance**[1]

Fisher Solution: normalize $|\tilde{\mu}_1$ - $\tilde{\mu}_2|$ by scatters

- Consider the projected samples: $\mathbf{z}_i = \mathbf{w}^T \mathbf{x}_i$
- Scatters (sample variance multiplied by *N*) for projected samples:
  - $\tilde{\mathbf{s}}_1 = \sum_{\mathbf{z}_i \in C_1} (\mathbf{z}_i - \tilde{\mu}_1)^2$
  - $\tilde{\mathbf{s}}_2 = \sum_{\mathbf{z}_i \in C_2} (\mathbf{z}_i - \tilde{\mu}_2)^2$
- The optimal solution can be achieved by the maximization, w.r.t **w**, of:

$$\boxed{J(\mathbf{w}) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{\mathbf{s}}_1 + \tilde{\mathbf{s}}_2}}$$

Closed-form optimal solution

The optimal **w** should be such that

- $(\tilde{\mu}_1 - \tilde{\mu}_2)^2$: large
- $\tilde{\mathbf{s}}_1$; $\tilde{\mathbf{s}}_2$: both small

[1]scatter measures the same thing as variance, the spread of data around the mean

Fisher's linear discriminant: learning $\mathbf{w}$ 2-class problems

The optimal solution can be achieved by the maximization, w.r.t **w**, of:

$$\boxed{J(\mathbf{w}) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1 + \tilde{s}_2}}$$

- express J explicitly as a function of **w** and maximize it
  - **straightforward but need linear algebra and calculus**
  - (...)

$$\boxed{J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_b \mathbf{w}}{\mathbf{w}^T \mathbf{S}_w \mathbf{w}}}$$

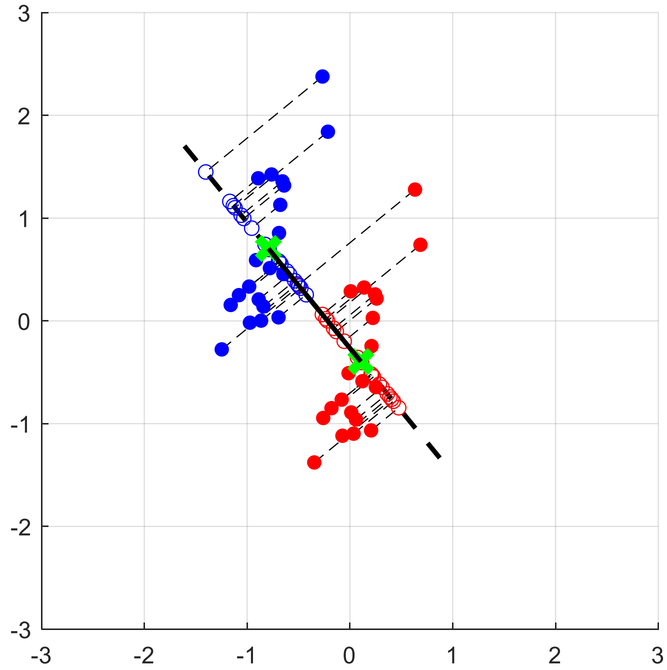
    - $\mathbf{S}_b$ is the **between class scatter matrix**
    - $\mathbf{S}_w$ is the **within class scatter matrix**
  - Thus, supposing $\mathbf{S}_w$ non-singular, the maximum is equivalent to the largest eigenvector $\mathbf{w}_1$ of $\mathbf{S}_w^{-1}\mathbf{S}_b$:

$$\boxed{\mathbf{S}_w^{-1}\mathbf{S}_b\mathbf{w}_1 = \lambda_1 \mathbf{w}_1}$$

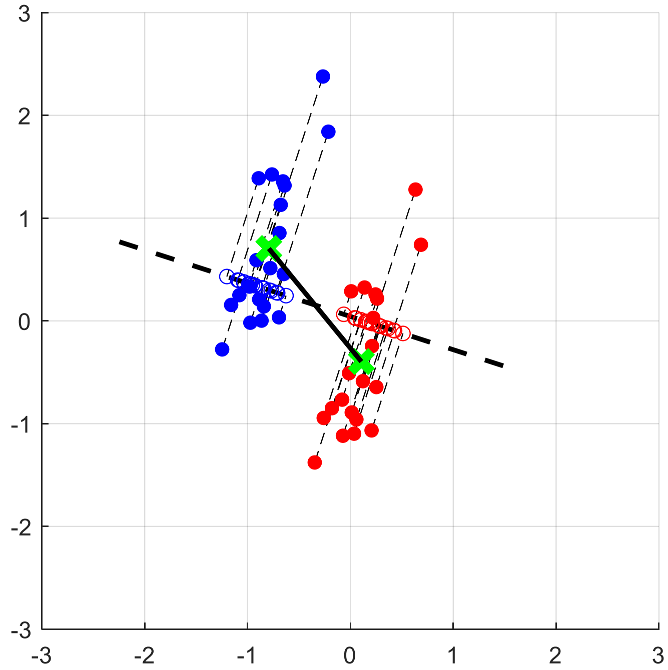
  - Moreover, noting that $\mathbf{S}_b$ is in the same direction as $(\tilde{\mu}_1 - \tilde{\mu}_2)$, and multiplying by $\mathbf{S}_w^{-1}$:

$$\boxed{\mathbf{w} \propto \mathbf{S}_w^{-1}(\tilde{\mu}_1 - \tilde{\mu}_2)}$$

Fisher's linear discriminant: learning $\mathbf{w}$
2-class problems



(a)



(b)

(a) direction to project is parallel to the vector of the difference between means. **Classes overlap**

(b) optimal direction (Fisher's solution): accounts for the difference between means and variance minimization. **Classes do not overlap**

Fisher's linear discriminant: learning $\mathbf{w}$ 2-class problems

- A variant for $C = 2$ classes
  - with both classes equiprobable (the same number of elements of each class in the data set)
  - avoids **eigendecompositions**

The **main steps** are

1. Compute **class means**: given N_c elements in each class

$$\boldsymbol{\mu}_c = \frac{1}{N_c} \sum_{i \in \omega_c} \mathbf{x}_i, \qquad c = 1, 2$$

2. Compute *within-class scatter matrix*: $\mathbf{S}_W = \mathbf{S}_1 + \mathbf{S}_2$
   - The matrices $\mathbf{S}_1$ and $\mathbf{S}_2$ are computed with data of class ω_1 and ω_2, respectively.
3. The **direction to project the data** is $\mathbf{w} = \mathbf{S}_W^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$
4. Project data: $y_k = \mathbf{w}^T \mathbf{x}_k, \qquad k = 1, 2 \ldots N$

Fisher's linear discriminant: learning $\mathbf{w}$
2-class problems

Decision rule: projecting the mean point of the means of two classes

$$p = 0.5\mathbf{w}^T(\boldsymbol{\mu}_2 + \boldsymbol{\mu}_1)$$

p is **the threshold** used to identify the projections y_k of each class.

- $y_k > p$ belongs to class ω_1 otherwise belongs ω_2

Re-writing by incorporating the projection calculation and $b = -p$

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \Rightarrow \begin{cases} g(\mathbf{x}) > 0 & \mathbf{x} \in \omega_1 \\ g(\mathbf{x}) < 0 & \mathbf{x} \in \omega_2 \end{cases}$$

The **linear discriminant function**: decision is taken with an **weighted sum of the feature values**

Fisher's linear discriminant: learning w algebraic approach

Given a training set: $(\mathbf{x}_k, y_k), k = 1 \ldots N$, the following system of equations

$$\begin{bmatrix} 1 & x_{11} & x_{12} & \ldots & x_{1D} \\ 1 & x_{21} & x_{22} & \ldots & x_{2D} \\ 1 & x_{31} & x_{32} & \ldots & x_{3D} \\ 1 & & & \ldots & \\ 1 & x_{N1} & x_{N2} & \ldots & x_{ND} \end{bmatrix} \begin{bmatrix} w_0 \equiv b \\ w_1 \\ \ldots \\ w_D \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \ldots \\ y_K \end{bmatrix} \Leftrightarrow \mathbf{X}\mathbf{w} = \mathbf{y}$$

where

- The *kth* row of $\mathbf{X}$ has $[1\ \mathbf{x}_k^T]$
- The *kth* row of vector $\mathbf{y}$ has y_k the value that the model should predict with $\mathbf{x}_k$
- The unknown $\mathbf{w}$: the vector of parameters of the model

System of equations: more equations than variables? $\Rightarrow$ Minimizing least squares

Fisher's linear discriminant: learning $\mathbf{w}$ algebraic approach

$$\mathbf{X}\mathbf{w} = \mathbf{y}$$

Multiplying both members by $\mathbf{X}^T$[2]

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y} \Rightarrow \mathbf{R}_{xx}\mathbf{w} = \mathbf{r}_{xy}$$

The solution is

$$\mathbf{w} = \mathbf{R}_{xx}^{-1}\mathbf{r}_{xy}$$

- $\mathbf{R}_{xx}$ is the correlation matrix (of augmented data vector $[1 \quad \mathbf{x}_k]$)
- The pseudo-inverse of $\mathbf{R}_{xx}$ can substitute the inverse.

[2] The first column of the data matrix has only ones

Fisher's linear discriminant: learning $\mathbf{w}$

Toy example 2D

Given the data set

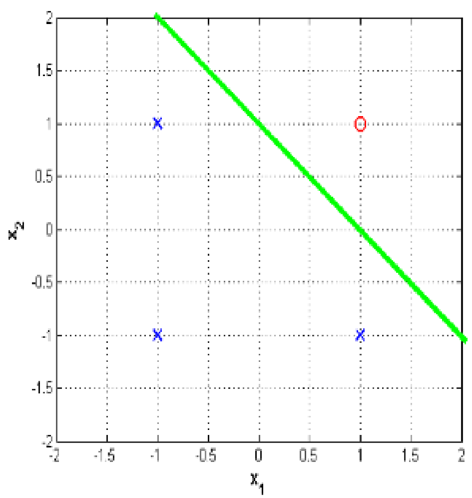
	A_1	A_2	label $\equiv d$
$\mathbf{x}_1^T$	-1	1	1
$\mathbf{x}_2^T$	1	-1	1
$\mathbf{x}_3^T$	-1	-1	1
$\mathbf{x}_4^T$	1	1	-1

The solution of the system of equations is

$$\mathbf{w} = \begin{pmatrix} w_0 \equiv b \\ w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0.5 \\ -0.5 \\ -0.5 \end{pmatrix}$$

Fisher's linear discriminant: learning **w**
Toy example 2D

The decision surface (**green line**) divides the feature space into two regions



The **parameters of the classifier** are

- $\mathbf{w} = [-0.5 \quad -0.5]^T$ and $b = 0.5$

The equation of green line is
$0.5 - 0.5x_1 - 0.5x_2 = 0$

$$\Rightarrow x_2 = 1 - x_1$$

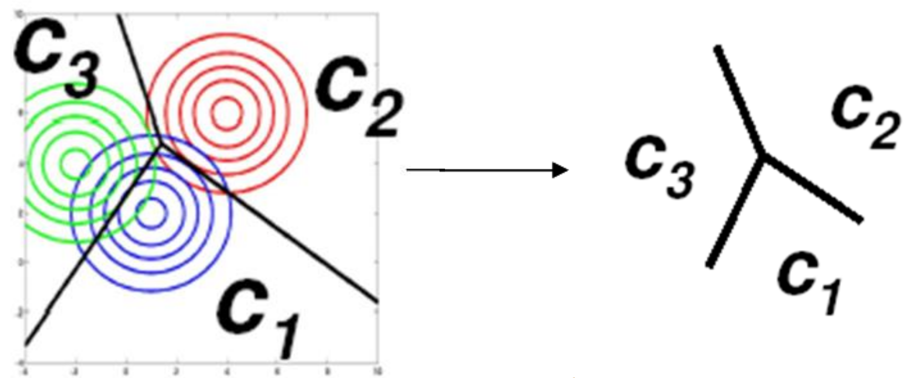
The **decision rule is**

$$y = [-0.5 \quad -0.5] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + b = -0.5x_1 - 0.5x_2 + 0.5 \Rightarrow \begin{cases} y > 0 & \textit{Region } \Re_1 \\ y < 0 & \textit{Region } \Re_2 \end{cases}$$

Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA)

- linear model for multiclass classification (generalization of Fisher's linear disc.)
- dimensionality reduction



Binary classification

- compute w of $\mathrm{g(x)}$

Decision:

- Given a new object $\mathbf{z}$, compute $g(\mathbf{z}) = \mathbf{w}^{\mathrm{T}}\mathbf{z} + w_0 \Rightarrow \begin{cases} g(\mathbf{x}) \geq \mathbf{0}, & \mathbf{x} \in \mathrm{C}_1 \\ g(\mathbf{x}) < \mathbf{0}, & \mathbf{x} \in \mathrm{C}_2 \end{cases}$

Multi-class classification

- compute the parameters of C functions $g_c(\mathbf{x}) = \mathbf{w}_{\mathrm{c}}^{\mathrm{T}}\mathbf{x} + b_c, \mathrm{c} = 1, .., \mathrm{C}$

Decision:

- Given a new object $\mathbf{z} : \mathbf{z} \in \mathrm{C_c}\text{: } \mathrm{g_c}(\mathbf{z}) > \mathrm{g_j}(\mathbf{z}), \forall\ \mathrm{j} \neq c$

Linear discriminant analysis: learning $\mathbf{w}$ multiclass

The solution is based on the definition

- **Within** class scatter matrices

$$\mathbf{S}_W = \sum_{c=1}^{C} \mathbf{S}_c$$

where $\mathbf{S}_c$ is the **scatter matrix** computed with the **data belonging to** c**- class**

- **Between** class scatter matrices

$$\mathbf{S}_B = \sum_{c=1}^{C} N_c \mathbf{S}_{m_c}$$

where N_c is the number of objects in class c and

$$\mathbf{S}_{m_c} = (\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^T$$

with $\boldsymbol{\mu}$ is the **global** mean vector and $\boldsymbol{\mu}_c$ is the **mean** vector of the **data belonging to class** c

Linear discriminant analysis: learning $\mathbf{w}$ multiclass

- The $C-1$ optimal directions are computed with the eigendecomposition of

$$\mathbf{S}_T = \mathbf{S}_W^{-1}\mathbf{S}_B$$

  - The matrix $\mathbf{S}_W$ should be invertible. For high-dimensional and sparse data it might be a drawback.
  - The $\mathbf{S}_T$ is not symmetric it might have non-real eigenvalues.
- The generalized eigendecomposition of $(\mathbf{S}_W, \mathbf{S}_B)$ is an alternative solution.

The chosen $(C-1)$ **eigenvectors** form the columns of the **projection model w**

- The new representation (reduced dimension)

$$\mathbf{P} = \mathbf{X}\mathbf{w}$$

  is formed by $(C-1)$ **features**

Fisher's and LDA

FLF and LDA assumptions on data:

- normal or Gaussian distribution
- classes with identical covariance matrices
  - However, it performs quite well when assumptions are violated → optimized

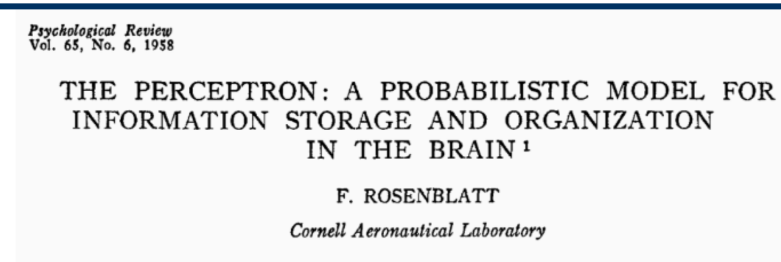
FLF and LDA

- fails when the discriminatory information is not in the mean, but rather in the variance of the data

Contents

- Predictive modelling
- **Linear classification models**
  - **Discriminant Functions**
    - **Least squares for classification**
    - **Fisher's Linear Discriminant (LDA)**
    - **Perceptron**
  - Logistic Regression
  - Comparison
- Iterative optimization
- Summary

Perceptron



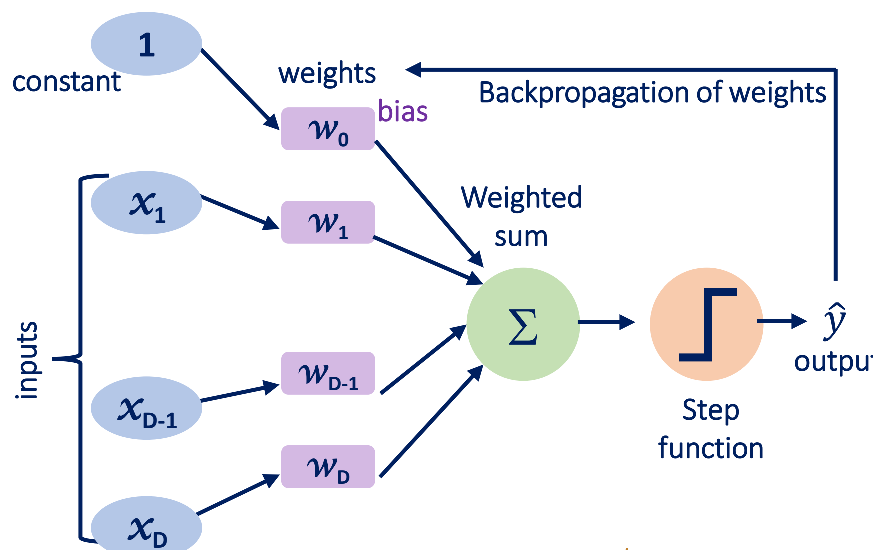
Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN [1]

F. ROSENBLATT

Cornell Aeronautical Laboratory

- Proposed by Rosenblatt (1958)
  - introduced the notion of **perceptron networks**
  - correspond to a 2-class model
- Perceptrons are the simplest ANN:
  - Only **one input** layer
  - Only **one output** layer
- Learn **linear** decision boundaries
- **Binary** problems



Perceptron

The processing steps:

$$g(\mathbf{x}) = f\left(\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0\right)$$

- **Weighted** sum of the inputs:

$$a = \sum_d w_d x_d + w_0 = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0$$

- $f(\cdot)$ **activation function**: step function

$$g(\mathbf{x}) = f\left(\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0\right) = f(a) = \begin{cases} +1, & a > \mathbf{0} \\ -1, & \text{otherwise} \end{cases}$$

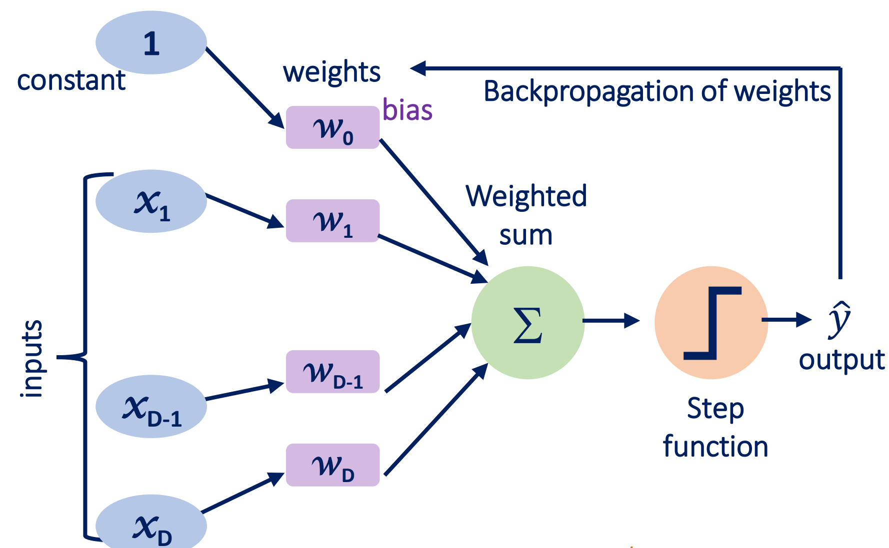
- Output: $f(a)$ - allows to assign a $class \in \{-1,1\}$ to feature vector $\mathbf{x}$

Perceptron algorithm

- During the **learning phase:**
  - For each example n (n =1,2,...N):
    - **if** $y_n \mathbf{x}_n^T \mathbf{w}^{(i-1)}$:
      - **update the weights**:

$$\mathbf{w}^{(i)} = \mathbf{w}^{(i-1)} + \lambda y_n \mathbf{x}_n$$

(λ is the learning rate)



Perceptron: learning w

- Algorithm

Algorithm 18.1 (The online perceptron algorithm).

- Initialization
  - $\boldsymbol{\theta}^{(0)} = \mathbf{0}$.
  - Select μ; usually it is set equal to one.
  - $i = 0$.
- **Repeat**; Each iteration corresponds to an epoch.
  - counter = 0; Counts the number of updates per epoch.
  - **For** $n = 1, 2, \ldots, N$, **Do**; For each epoch, all samples are presented.
    - **If** $(y_n \boldsymbol{x}_n^T \boldsymbol{\theta}^{(i-1)} \leq 0)$ **Then**
      - $i = i + 1$
      - $\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu y_n \boldsymbol{x}_n$
      - counter=counter+1
  - **End For**
- **Until** counter=0

- It only updates weights[1] when misclassification occurs

[1] $\boldsymbol{W} \equiv \theta$, *training set* $(\mathbf{x}_i, y_i)$, i *iteration (epoch)*, μ *learning rate*

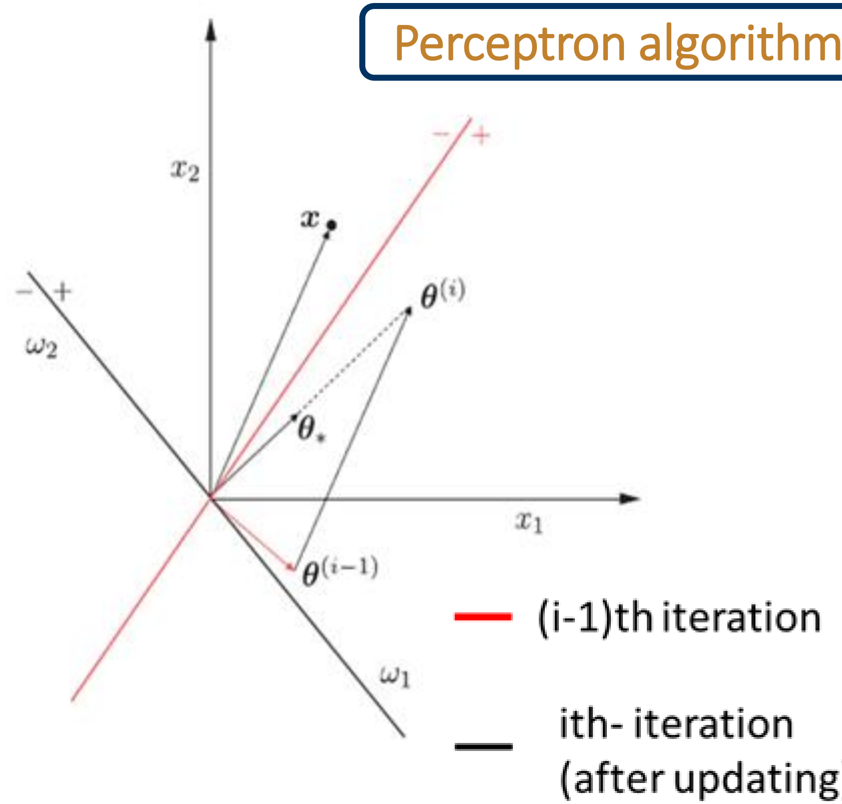
Perceptron: learning w

- The decision surface is orthogonal[1] to $\theta = [\theta_1 \;\; \theta_2]^T$,

Perceptron algorithm

At each iteration, only updates weights when misclassification occurs

- Assuming that **x** is misclassified
  - The **weight** vector moves towards **x**



[1] $\theta_0 \equiv b = 0$, the decision lines crosses the origin of the feature space

Perceptron: learning **w**

Perceptron convergence theorem

- if there exists an exact solution (i.e., if the training data set is linearly separable)
  - perceptron learning algorithm is guaranteed to find an exact solution in a finite number of steps
    - # steps to convergence could still be substantial

- Limitations of the Perceptron learning algorithm
  - **It only stops when all exemples are learned**
    - Not applicable in real applications
  - **Limitation of the weight updating rule**
    - It does not guarantee to reduce the total error function at each stage

If data is not linearly separable, the perceptron learning algorithm will never converge

→ Alternative: optimization of the error

Perceptron: learning $\mathbf{w}$

Perceptron criterion associates zero error with any input **correctly classified**

- $\mathbf{x}_n$ in C_1 (y_n=+1): add to $\mathbf{w}$
- $\mathbf{x}_n$ in C_2 (y_n=-1): subtract from $\mathbf{w}$
- For each **misclassified** sample, it tries to **minimize**

$$E_p(\mathbf{w}) = -\sum_{m \in M} \mathbf{w}^T \mathbf{x}_m y_n = \sum_{m \in M} E_n(\mathbf{w})$$

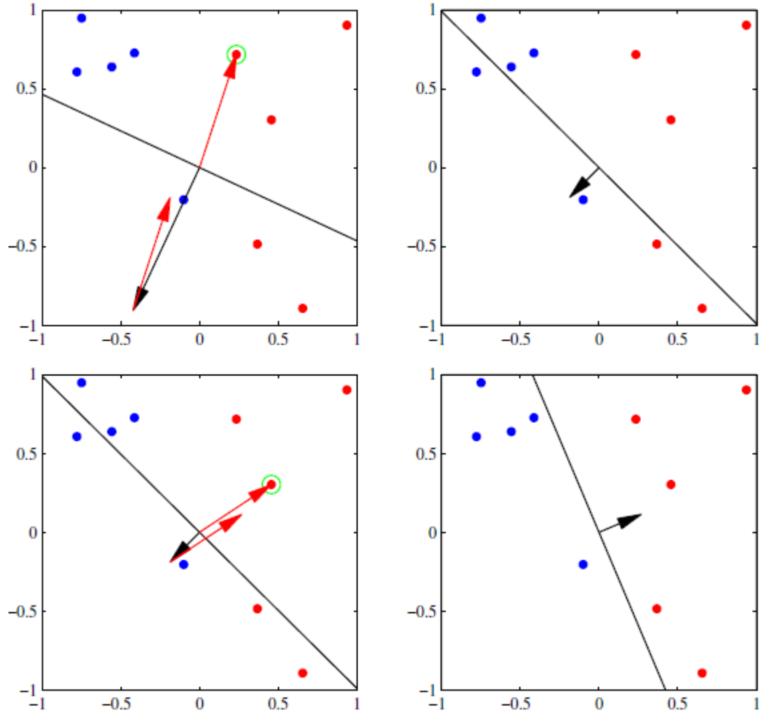
  - M: set of all misclassified samples

- **Gradient Descent (GD)**
  - **Weight update formula:** for each misclassified $\mathbf{x}_m$

$$\mathbf{w}^{(i)} = \mathbf{w}^{(i-1)} - \lambda E_n(\mathbf{w}) = \mathbf{w}^{(i-1)} + \lambda(y_n \mathbf{x}_n)$$

GD for Perceptron criterion

$C_1 : y_n$=+1, $C_2 : y_n$=-1

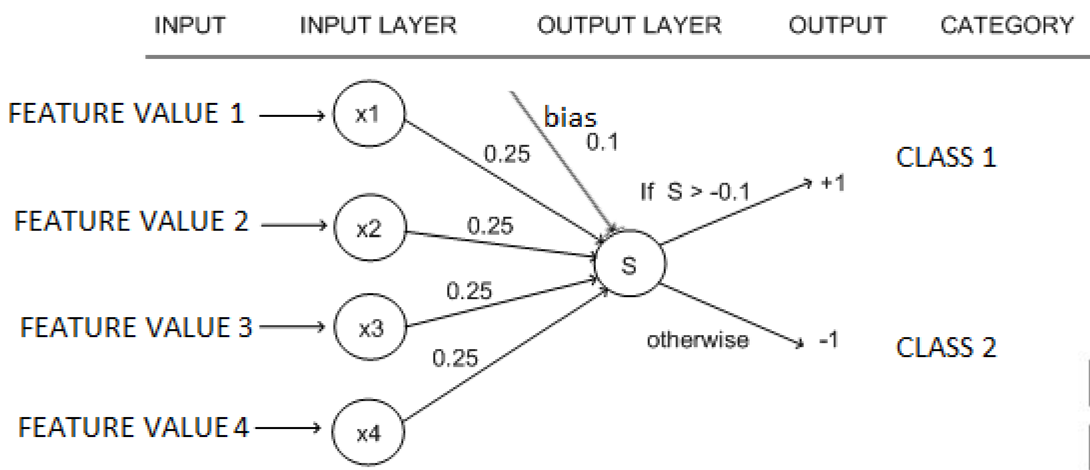


Pattern Recognition and Machine Learning, C. Bishop, *Springer, 2007*

Perceptron: limitations

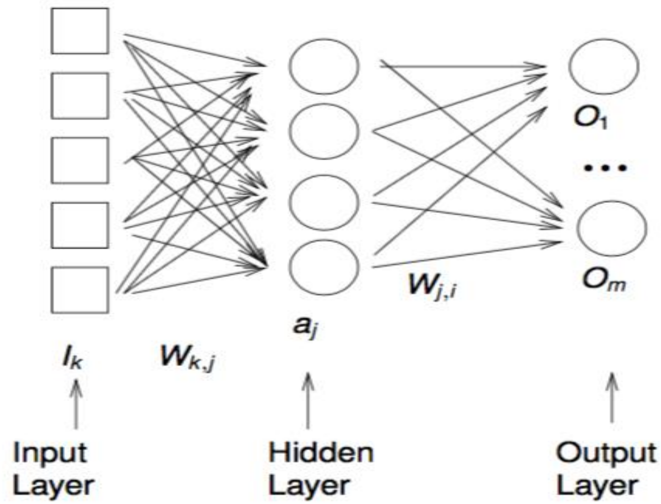
- Perceptron only works for 2 classes
- If dataset is not linearly separable, the **perceptron algorithm will not converge**
- Based on linear combination of fixed basis functions
- **Different solutions** depending on the initialization of $\mathbf{w}$
- Online algorithm: **decision boundary** depends on the order of the learning examples
  - Also, the order in which data is presented in GD affects the decision boundaries

Extending the Perceptron …



Perception: precursor for Neural Networks!

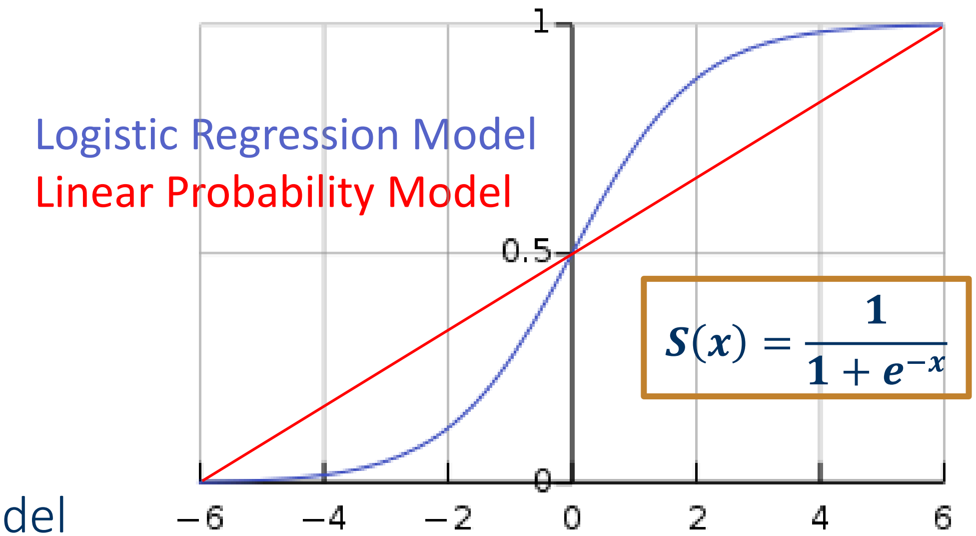
Extending Perceptron: connecting units together into multilayer Neural Networks!



Contents

Logistic Regression: a generalized linear model

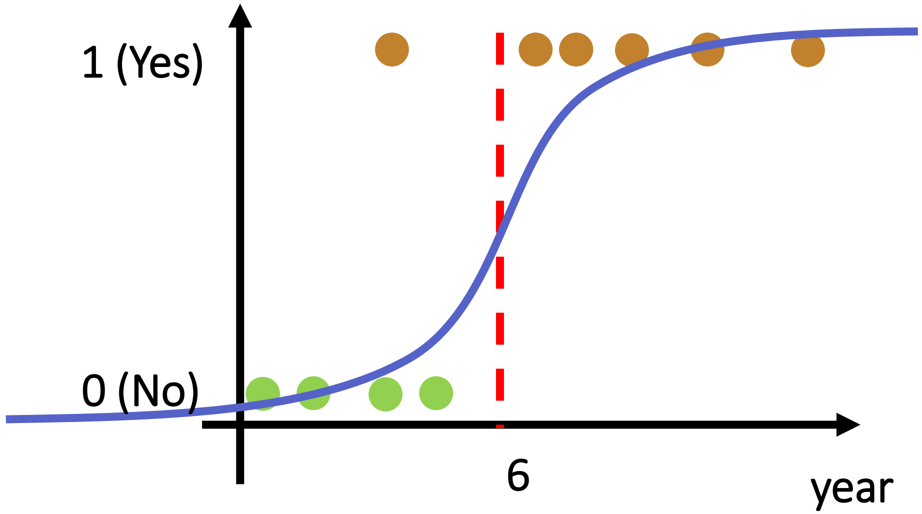
- **Probabilistic discriminative linear** model for **binary** classification
- Key Idea: Turn linear predictions into probabilities
  - The goal is to calculate a score
- **Sigmoid function** to define the conditional probability of $y \in \omega_1$
  - $P(y \in \omega_1 | \mathbf{x}) = \frac{1}{1+\exp(-y)}, \quad y = \mathbf{w}^\top \mathbf{x} + b$
    - Projects $(-\infty, +\infty)$ to $[0, 1]$
- Decision rule
  - $y \in \omega_1$ if $P(y \in \omega_1 | \mathbf{x}) > 0.5$
  - $y \in \omega_2$ if $P(y \in \omega_2 | \mathbf{x}) > 0.5$
- Smoother than linear probability model



Logistic Regression

- **Key Idea**: Turn linear predictions into probabilities
  - The goal is to calculate a score

$$S(x) = \frac{1}{1 + e^{-x}}$$

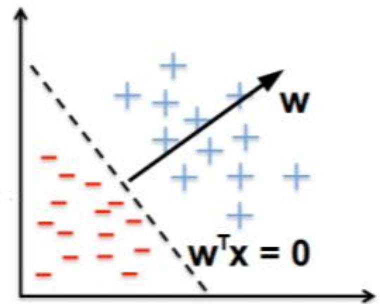


- Decision rule
  - $y \in \omega_1$ if $P(y \in \omega_1 | \mathbf{x}) > 0.5$
  - $y \in \omega_2$ if $P(y \in \omega_2 | \mathbf{x}) > 0.5$

$$y = \mathbf{w}^\top \mathbf{x} + b$$

- Probabilistic discriminative linear model for **binary** classification
  - Learns the PMF of the output label given the input, i.e., $p(y \in \omega_k | \mathbf{x}), \quad k = 1,2$
  - Does not model inputs $\mathbf{x}$ (only relationship between $\mathbf{x}$ and y)
  - $P(y \in \omega_1 | \mathbf{x})$ are nonlinear functions with a linear function of $\mathbf{x}$ as input
    - **Linear** relation with the parameters ($\mathbf{w}$, b) of the classifier $g(\mathbf{x})$ despite the non-linear scoring function

Logistic Regression



- Very large positive $\boldsymbol{w}^\top\mathbf{x}$ means $p(y = 1|\mathbf{x})$ close to 1
- Very large negative $\boldsymbol{w}^\top\mathbf{x}$ means $p(y = 0|\boldsymbol{w})$ close to 1
- At decision boundary, $\boldsymbol{w}^\top\mathbf{x}=0$ implies $p(y = 1|\mathbf{x}) = p(y = 0$

The logit function gives the logarithm of the probability of one class divided by the probability of the other class

$$logit\left(P(y \in \omega_1|\mathbf{x})\right) = ln\left(\frac{P(y \in \omega_1|\mathbf{x})}{1 - P(y \in \omega_1|\mathbf{x})}\right) = \mathbf{w}^\top\,\mathbf{x} + b$$

  - It is the log of the odds ratio
  - It links the probability to the predictor variables

- Extension **Multiclass Logistic Regression** (aka Softmax)

Logistic Regression: learning w

Given the **training data set** $\chi = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$, where $y_n \in \{0,1\}$ are the values to assigned to the class labels

Likelihood function*:

$$L(\mathbf{w}, b|\chi) = \prod_{n=1}^{N} P(y_n|\mathbf{x}) = \prod_{n=1}^{N} p_n^{y_n}(1-p_n)^{1-y_n}$$

where the probability of $\mathbf{x}_n$ belonging to the class with label:

$$y_n = 1 \; is \; p_n = \frac{\exp(\boldsymbol{w}^\top \mathbf{x}_n)}{1+\exp(\boldsymbol{w}^\top \mathbf{x}_n)} \qquad y_n = 0 \; is \; 1 - p_n = 1 - \frac{\exp(\boldsymbol{w}^\top \mathbf{x}_n)}{1+\exp(\boldsymbol{w}^\top \mathbf{x}_n)}$$

- The likelihood measures the goodness of fit of a statistical model to a sample of data for given values of the parameters ($\mathbf{w}$, b)

*Bernoulli distribution

Logistic Regression: learning w

Cost function

- The log-likelihood $J(\,) = -\log\big(L(\,)\big)$

$$E = J(\mathbf{W}) = -\sum_{n=1}^{N}\big(y_n log(p_n) + (1-y_n)log(1-p_n)\big)$$

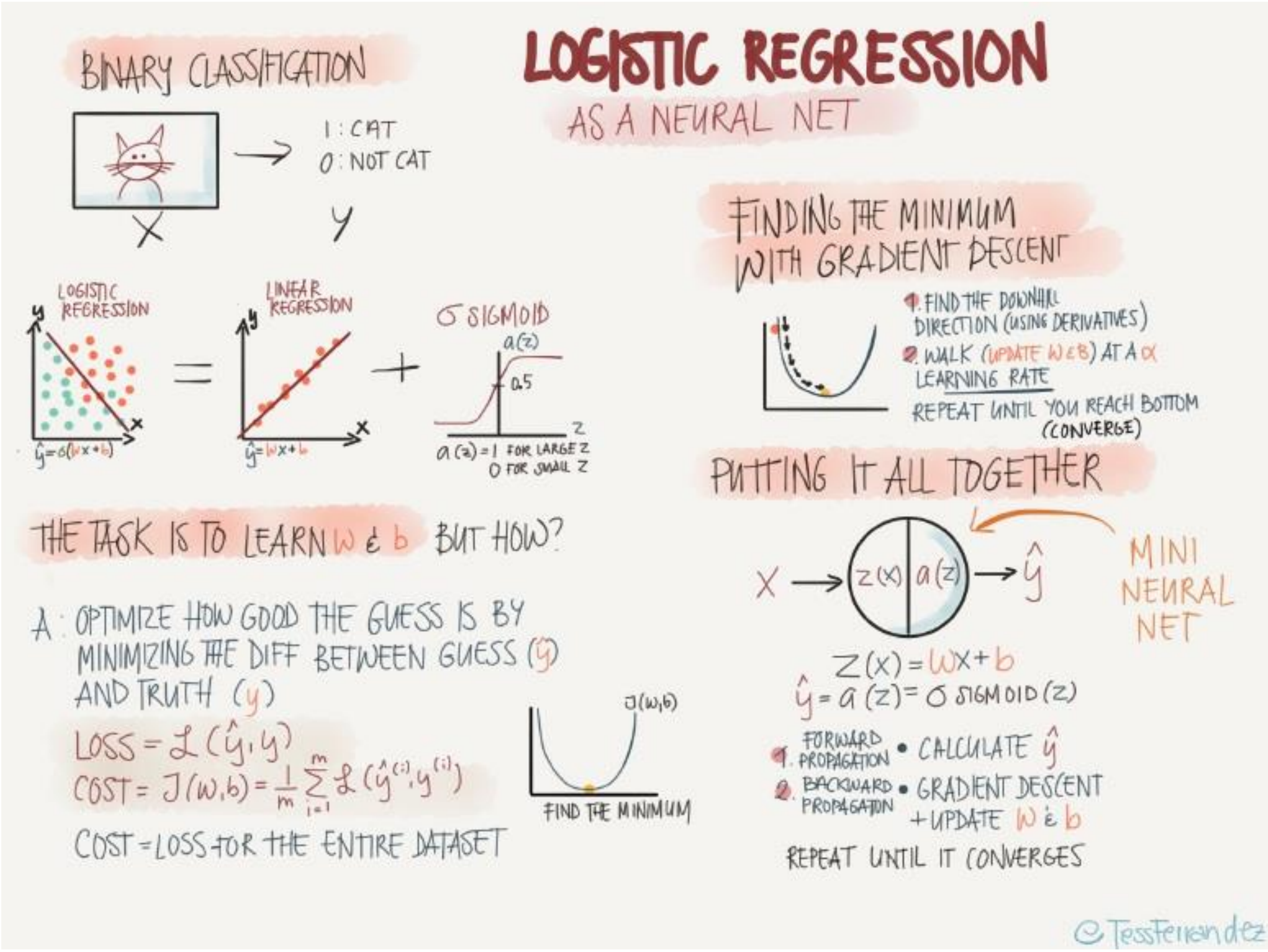
- No closed form solution
  - Iterative optimization methods are need to **minimized** the **cost function**
    - **gradient descendente**

The parameter **updating values*** are

$$\Delta w^d = -\lambda \frac{\partial E}{\partial w_i} = \lambda \sum_{n=1}^{N}\left(\frac{y_n}{p_n} - \frac{1-y_n}{1-p_n}\right) p_n(1-p_n)x_n^d$$

* Note that $x_n^d \equiv \langle x_d \rangle_n$, $w^d \equiv w_d$, bias $b \equiv w_0, \mathrm{x}_0 = 1$

Logistic Regression



https://storage.ning.com/topology/rest/1.0/file/get/2408482975?profile=original

Contents

Comparison: Training/Learning in linear approaches

Different algorithms to the binary classification problem

- FDF and LDA : Algebraic approach or **optimized criterium**
- Perceptron: perceptron learning algorithm or **optimized criterium**
- Logistic Regression: defining a cost function (based on likelihood) and **optimization** with iterative optimization methods

Optimized criterium for FDF, LDA and Perceptron

- **Optimization** of the J(·) / error with iterative optimization methods
  - gradient descendent

Comparison: Training/Learning in linear approaches

FDF/LDA:

- fails when the discriminatory information is not in the mean, but rather in the variance of the data

Logistic regression:

- the non-linearity of $f(\cdot)$ gives more flexibility → it can limit the effects of outliers
- **w** can become unstable:
  - If classes are well-separated
  - Data size (#objects) is small

Perceptron:

- **decision boundary** depends on the order of the learning examples

Comparison: Training/Learning in linear approaches

Logistic Regression & Perceptron: only for binary classification

- Multiclass logistic regression

FDF/LDA vs Logistic regression:

When the normality assumption of data (approximately) holds, its expected that FDF/LDA outperforms Logistic regression

Perceptron vs Logistic regression:

Perceptron only updates weights when misclassification

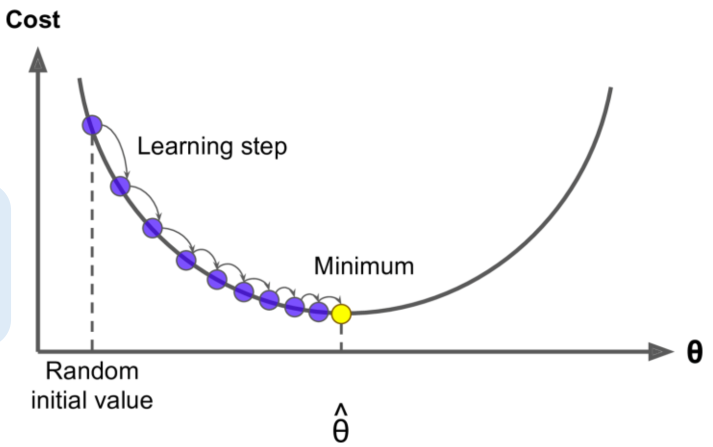
Contents

Iterative optimization methods: Gradient-based learning

- Iterative optimization algorithm for finding the minimum of a function, typically a convex **cost/loss** function
  - Mostly used when there is no closed form solution (the parameters can not be calculated analytically, e.g., using linear algebra)
  - Used to fit linear classifiers and regressors under convex loss functions
- For a function $F(x)$ at a point **a**, $F(x)$ decreases fastest if we go in the direction of the negative gradient of **a**

$$a_n = a_{n-1} - \lambda \, \nabla \mathrm{F}(a_{n-1})$$

When the **gradient** is zero, we arrive at the **local minimum**



Iterative optimization methods: Gradient-based learning

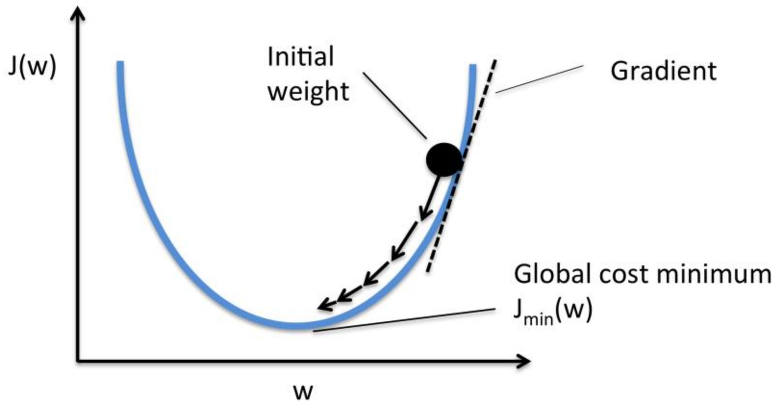
- Cost/loss function $E = J(\mathbf{w})$
- **Vector gradient**

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_D}\right)$$

- each **weight** is updated according to

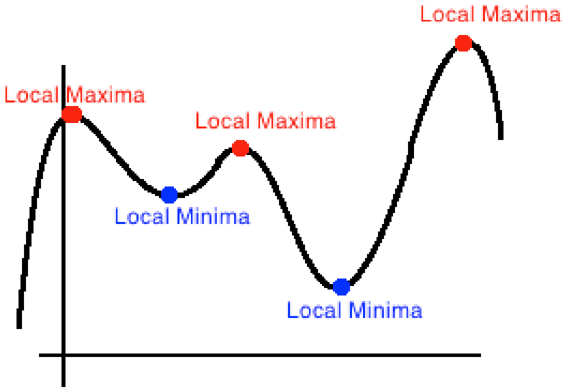
$$\Delta w_i = -\lambda \frac{\partial E}{\partial w_i}$$

Linear activation function



The learning rate λ is the step length in negative gradient direction

Non-linear error function



drawback (to live with): the learning can converge to any local minimum

Iterative optimization methods: Batch vs Stochastic Gradient Descent

Global Cost/loss function $E = \sum_n E^{(n)}$

The gradient can be estimated with

- **Batch Gradient Descent:** with the total (or average) error in training set E

$$\boldsymbol{w}^{(i)} = \boldsymbol{w}^{(i-1)} - \lambda \nabla E$$

  - the batch size is the number of samples

- **Online/Stochastic Gradient Descent**: with the error of a single training example $E^{(n)}$

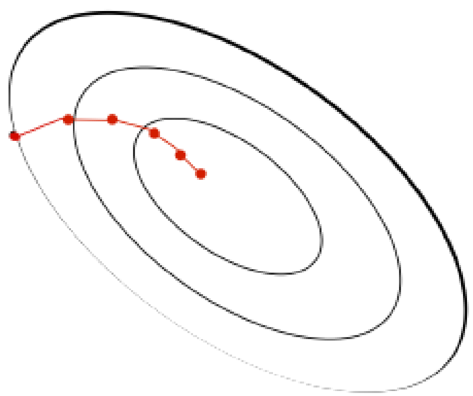
$$\boldsymbol{w}^{(i)} = \boldsymbol{w}^{(i-1)} - \lambda \nabla E^{(n)}$$

  - instead of calculating the gradient of the full error function (which involves using the full training set), we update the weights one example at a time

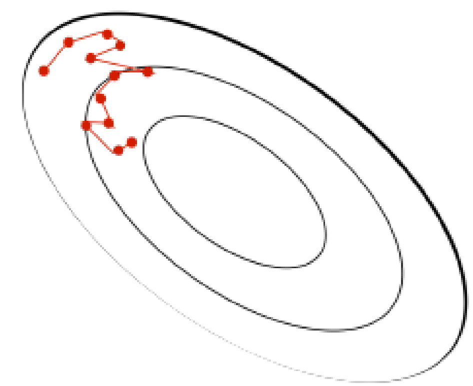
Both are more effective to escape from local minima

Iterative optimization methods: Batch vs Stochastic Gradient Descent

- Batch gradient descent moves directly downhill
- Batch is impractical for large data sets
- SGD takes steps in a noisy direction, but moves downhill on average
- Randomly sampling the training set → SGD converges to Batch solution
- **Mixed Strategy: Mini-Batch**



batch gradient descent



stochastic gradient descent

Contents

Summary

- **Predictive modelling**
  - Predictive problems
  - Models approaches for classification
- **Linear classification models**
  - Discriminant Functions
    - Least squares for classification
    - Fisher's Linear Discriminant (LDA)
    - Perceptron
  - Logistic Regression
  - Comparison
- **Iterative optimization**
  - Batch Gradient Descent
  - Stochastic Gradient Descent

Bibliography

Introduction to Data Mining, Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, Vipin Kumar, *Pearson*, 2019 (chap 3.1, 3.2, 6.1, 6.6, 6.7)

Data Mining, the Textbook, Charu C. Aggarwal, *Springer*, 2015 (chap 10.2.1.4, 10.5.2, 10.7.1)

Machine Learning: The Art and Science of Algorithms That Make Sense of Data*, P. Flach, Cambridge University Press*, 2012 (ch 1, 2, 7.1, 7.4)

Pattern Recognition and Machine Learning, C. Bishop, *Springer*, 2007 (ch 4)