

Aarhus University
Department of Engineering
Electrical & Computer Engineering

Introduction to Machine Learning

Alexandros Iosifidis

Course notes

November 21, 2017

Preface

This document provides a short description of the topics included in the second part of the course “Optimization and Data Analytics”, taught in the Department of Engineering, Electrical and Computer Engineering, at Aarhus University. The course starts by introducing Optimization theory and related methods, topics that are necessary for proceeding to the second part of the course focusing on Data Analytics and Machine Learning. The document should be used as a supplement to the readings provided by the course list. Since this is the first version of the course notes, typos might exist. Please contact me in case you notice such typos.

Contents

1	Introduction	7
1.1	Categorization of Machine Learning Models	11
1.2	Notations	12
2	Unsupervised Learning	13
2.1	Data pre-processing	13
2.2	Similarity measures	14
2.3	Principal Component Analysis	16
2.4	Clustering	18
2.5	Questions and Exercises	22
3	Probability-based Learning	23
3.1	Risk-based decision functions	26
3.2	Decision functions of the Normal Density	27
3.3	Discrete Features	31
3.4	Maximum Likelihood Estimation	31
3.5	Expectation-Maximization	33
3.6	Questions and Exercises	34
4	Linear Methods	37
4.1	Fisher Discriminant Analysis	38
4.2	Linear Discriminant Analysis	40
4.3	Linear Discriminant Functions	41
4.3.1	Linearly-separable case	43
4.3.2	Non-linearly separable case	47
4.3.3	Use of Linear Programming Algorithms	48
4.4	Generalized Linear Discriminant Functions	51
4.5	Extension to multi-class classification	52
4.6	Questions and Exercises	54
5	Kernel-based Learning	57
5.1	Support Vector Machine and Kernels	57
5.2	Least-Mean-Square Regression	60

5.3	Kernel Discriminant Analysis	61
5.4	Questions and Exercises	63
6	Multilayer Neural Networks	65
6.1	The Backpropagation Algorithm	68
6.2	Activation functions	73
6.3	Practical matters	75
6.4	Radial Basis Function network	76
6.5	Auto-Encoder networks	77
6.6	Self-Organizing Map	78
6.7	Convolutional Neural Networks	80
6.8	Questions and Exercises	83
7	Validation	85

Chapter 1

Introduction

Humans learn to interact with their environment in the early stage of their lives. This is done by recognizing faces, objects, sounds, etc., or by using a more general term *patterns*. Pattern Recognition, i.e. the process of taking as input raw data (e.g. a sequence of images depicting the face of a person) and taking an action based on the *category* or *group* in which the recognized pattern belongs to, is of vital importance in our everyday life. The recent advancement of science and technology have inevitably led to a desire in making computers (and in more general machines) capable to *understand* and *recognize* such patterns, in order to highly automate decision making in various fields. Machine Learning is the field of Computer Science and Engineering dealing with processes and methodologies that can be expressed in a form that can be used by computers and make them learn how to make decisions related to patterns.

For example, let us consider the problem of recognizing a person through his/her facial image. Such a process requires a number of steps: capturing the image, detection of the image location depicting the face and segmentation of the facial region in order to create a smaller image depicting only the person's face. Additional steps might also be required in order to make the problem easier to be solved by a computer, like light normalization, frontalization of the human face. These pre-processing steps are related to the specific problem at hand, and can be different if the problem to be solved comes from another field. Example (frontal) facial images are illustrated in Figure 1.1.

After obtaining the pre-processed samples, we need to represent them in a way that can make the problem easier for the computer to solve it. For example, in the problem of distinguishing the two persons depicted in Figure 1.1, one would argue that these two people can be easily distinguished by considering their age as a characteristic. Indeed, the reader can easily realize that the person depicted in the second row of Figure 1.1 is, most probably, older than the person depicted in the first row. However, even though such a characteristic can lead to a good recog-



Figure 1.1: *Facial images depicting two persons.*

nition result, it is difficult to calculate by a computer. In order to automatically perform the recognition task, a computer needs to calculate image characteristics that can be easily and reliably computed using the raw image data. Example such characteristics include the image intensities and colors, edges and local patterns, etc. We call the set of such characteristics *features*, which are used to represent the images/samples. The process of extracting the features forming the sample's representation is called *feature extraction*.

In the example above, we will use as features the intensities of the (gray-scale) image, which is vectorized in order to form a facial vector $\mathbf{x}_i \in \mathbb{R}^D$. For a 40×30 pixel facial image, the corresponding facial vector dimensionality is equal to $D = 1200$. In order to visualize the facial image representations in the two-dimensional space, the facial image vectors \mathbf{x}_i need to be mapped to vectors $\mathbf{z}_i \in \mathbb{R}^2$. When this mapping is done using a linear transformation, we have:

$$\mathbf{z}_i = \mathbf{W}^T \mathbf{x}_i, \quad (1.1)$$

where $\mathbf{W} \in \mathbb{R}^{D \times 2}$ is the transformation matrix and $(\cdot)^T$ denotes the transpose operator for a vector/matrix. For now, we assume that the matrix \mathbf{W} is defined in the feature selection process or feature extraction process we use for representing the facial images of the two persons in Figure 1.1. We will see how to calculate such transformation matrices using unsupervised and supervised criteria in the following (Sections 2 and 4).

The 2-dimensional facial image vectors \mathbf{z}_i corresponding to the facial images of the two persons in Figure 1.1 are plotted in Figure 1.2. In this figure, the 2D points illustrated with asterisks correspond to the facial image vectors of the first person, while the 2D points illustrated with triangles correspond to the facial image vectors of the second person.

In order to distinguish the facial images depicting the two persons, we need to define a *decision function*. Intuitively, a good decision function would be the one that can correctly classify all facial images according to their class, which is already known. Such a decision function is illustrated in Figure 1.3. In order to

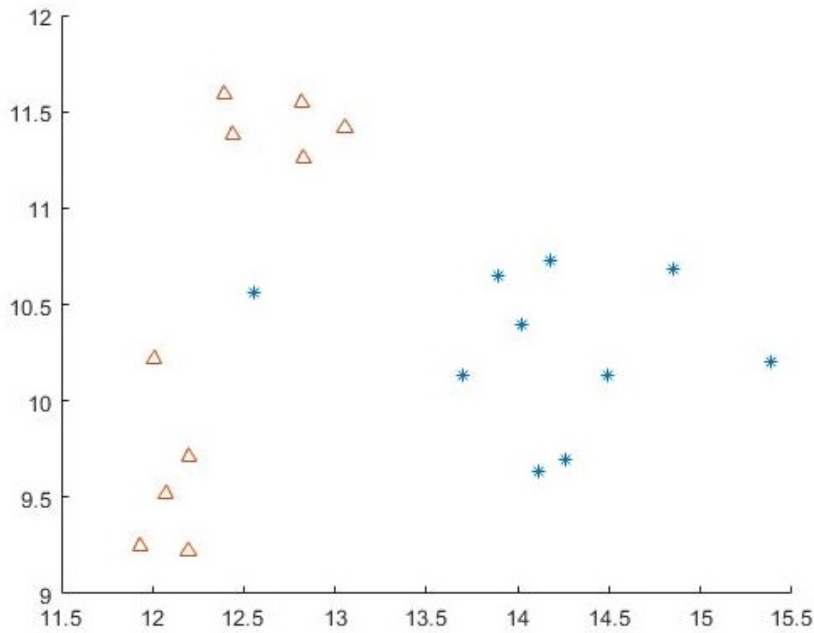


Figure 1.2: 2-dimensional facial image vectors belonging to two classes

obtain a decision function, one needs to firstly define its type and, then, optimize its parameters for satisfying an objective (e.g. the highest correct classification rate) or minimizing some total expected cost (e.g. the mean error rate over a set of samples). Parameters' optimization is performed using a set of samples for which the labels are known. Such samples are called training samples and the process of parameters' optimization is called training. The type of decision function and its parameters form the so-called (classification) *model*. Thus, the process of defining the best decision function is called *model selection*.

Depending on the selected model's type and the process used to optimize its parameters, different decision functions will be obtained. For example, another decision function for the above described problem is shown in Figure 1.4. Comparing the two decision functions, we can observe that the decision function in Figure 1.4 is simpler, while the decision function in Figure 1.3 is better in classifying the training samples. However, one can argue that (based on the arrangement of the training samples of the two classes) the simpler decision function is more probable to perform better on new samples, since the sample of the blue class leading to complicating the classification task is an outlier. Forcing the decision function to correctly classify such outliers leads to *overfitting*, which results in complex decision functions (usually requiring a higher number of parameters) that are more probable to perform poorly on unseen samples. The ability of a model to perform well on unseen samples is called *generalization of the model*.

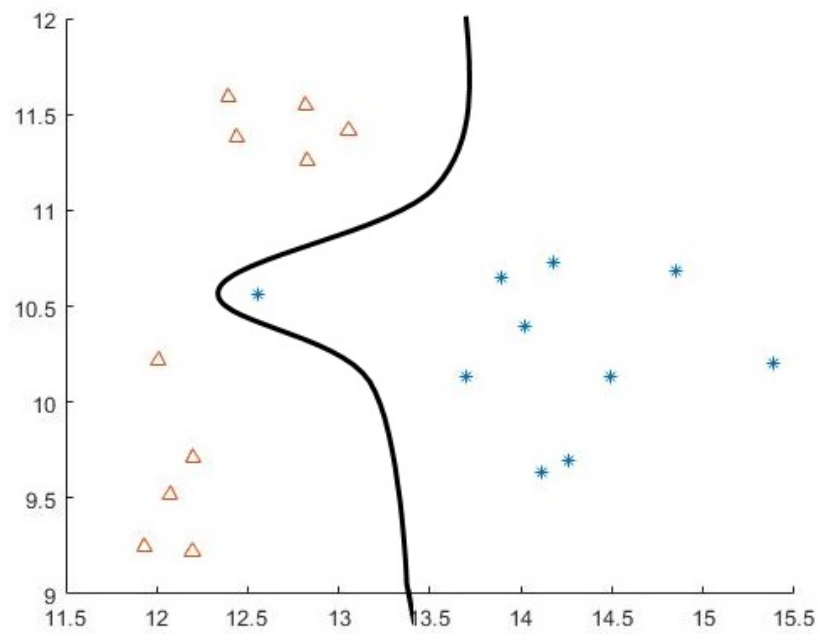


Figure 1.3: *Decision function correctly classifying all training samples.*

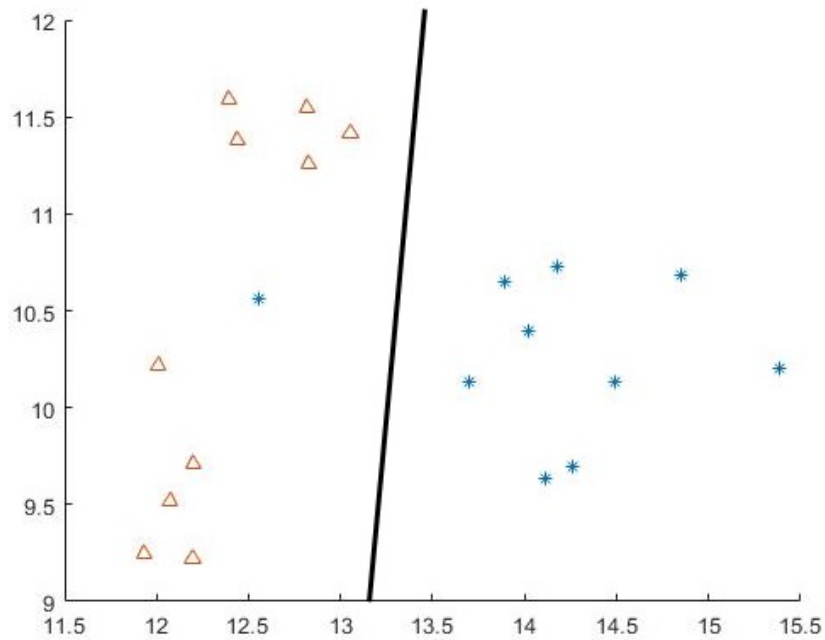


Figure 1.4: *A simpler decision function classifying correctly all but one training sample.*

Except the performance of a model, another important factor is its *computational complexity*, which is defined as the number of computations needed in order to reach a decision. While in some problems we might be able to obtain very accurate models, their computational complexity might be prohibitive. As an example let us assume that in the above classification problem we are able to capture facial images depicting all possible variations (including different facial poses, expressions and lighting conditions), leading to millions of samples for each class. In this case, a new facial image can be correctly classified simply by comparing it with all known facial images. While such a decision model would lead to perfect performance, its computational complexity is extremely high compared to the simple classification model in Figure 1.4.

1.1 Categorization of Machine Learning Models

As has been discussed above, Machine Learning is the field focusing on devising processes and methodologies that can be used to make computers recognize patterns and make decisions. Depending on the type of information being available during this process, it can be roughly split in three categories.

In *Supervised Learning*, the information available for model selection is formed by a set of training samples and their labels. This means that the training set has been labeled by a human expert and these labels can be used by the model selection process in order to define a decision function which is in line with the decisions made by the expert. While in this way the model selection process can exploit valuable information related to the problem at hand, labeling is a tedious and expensive process and this is why labeled sets are rare and small in size.

In *Unsupervised Learning*, there is no expert who can provide domain knowledge for the problem at hand. Instead, there is the assumption that samples naturally form groups based on a similarity criterion and the model selection process will be able to reveal this similarity function and identify the underlying groups of patterns.

In *Reinforcement Learning*, model selection is guided by an expert who can provide only binary feedback. This means that the training process follows an iterative process where the model is presented with a sample and makes a guess. The expert observes this guess and indicates if it is correct or not. Given this feedback, the training process can either update the model (in case of an error) or continue by using another training sample.

1.2 Notations

Throughout this document we will refer to sets with capital calligraphic letters and samples within a set will be denoted by capital italic letters. For example, a set of images will be denoted as \mathcal{I} and the i -th image in this set as $I_i \in \mathcal{I}$. Vectors and matrices will be denoted by lower-case and upper-case bold letters, i.e. \mathbf{x} and \mathbf{X} , respectively. We say that \mathbf{x} belongs to a D -dimensional space, i.e. $\mathbf{x} \in \mathbb{R}^D$, when \mathbf{x} is formed by D elements (each denoting the value of \mathbf{x} in the corresponding dimension of \mathbb{R}^D). A set of N vectors \mathbf{x}_i , $i = 1, \dots, N$ can be denoted by a matrix $\mathbf{X} \in \mathbb{R}^{D \times N}$, where each column i corresponds to the \mathbf{x}_i . The dot product between two vectors $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{y} \in \mathbb{R}^D$ is denoted by $\mathbf{x}^T \mathbf{y}$, where \cdot^T is the transpose operator.

Chapter 2

Unsupervised Learning

In the case where the only available information is a set of D -dimensional vectors \mathbf{x}_i , $i = 1, \dots, N$, one can try to define patterns based on their natural grouping in clusters. The outcome of clustering depends on various factors, like the relative scale of the various dimensions of the data, the selected similarity (or distance) metric, the criterion function used in order to define the clustering objective and the optimization process followed in order to optimize this objective.

2.1 Data pre-processing

Given a set of vectors \mathbf{x}_i , in which each dimension corresponds to a different feature type (possibly taking values in different ranges and having different scales), it is usually beneficial to apply pre-processing before continuing to the application of pattern recognition techniques.

Data centering is the process of translating the mean of each dimension of the data matrix $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ to zero. This is achieved by applying:

$$\bar{\mathbf{x}}_i = \mathbf{x}_i - \boldsymbol{\mu}, \quad i = 1, \dots, N, \quad (2.1)$$

where $\boldsymbol{\mu} \in \mathbb{R}^D$ is the mean vector of the dataset given by:

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i. \quad (2.2)$$

Data centering makes the various dimensions contribute equally to the optimization of the criterion used in subsequent steps (in the case of equal dimension scales). When scaling of the various dimensions is not equal, an additional scaling process can be applied.

Data standardization applies both centering and scaling of each dimension of the data. Let us denote by $\mathbf{s} \in \mathbb{R}^D$ a vector formed by the standard deviation values corresponding to each dimension of the data, i.e.:

$$s_d = \frac{1}{N-1} \sqrt{\sum_{i=1}^N (x_{id} - \mu_d)^2}. \quad (2.3)$$

Then, the standardized sample $\hat{\mathbf{x}}_i$ has dimensions equal to:

$$\hat{x}_{id} = \frac{x_{id} - \mu_d}{s_d}. \quad (2.4)$$

Data normalization scales each sample independently, so that it satisfies a criterion. For example, Euclidean norm (or l_2) normalization scales the values of a vector \mathbf{x}_i as follows:

$$\tilde{\mathbf{x}}_i = \frac{\mathbf{x}_i}{\|\mathbf{x}_i\|_2} = \frac{\mathbf{x}_i}{\sqrt{\mathbf{x}_i^T \mathbf{x}_i}}. \quad (2.5)$$

In the case of vectors formed by positive values, l_1 normalization scales the values of a vector \mathbf{x}_i as follows:

$$\dot{\mathbf{x}}_i = \frac{\mathbf{x}_i}{\|\mathbf{x}_i\|_1} = \frac{\mathbf{x}_i}{\sum_{d=1}^D x_{id}}. \quad (2.6)$$

l_2 and l_1 normalization can be seen as mapping a vector to the unit circle (hypersphere when $D > 2$) and the unit chord, as illustrated in Figure 2.1.

Rotation such that the data dimensions correspond to the principal axes (i.e. the directions corresponding to the maximal spread of the data values) is another type of pre-processing. It is applied by:

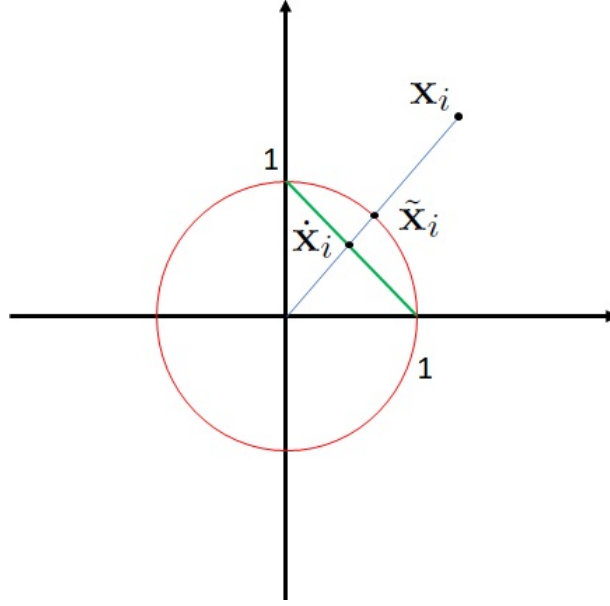
$$\ddot{\mathbf{x}}_i = \mathbf{R}^T \mathbf{x}_i. \quad (2.7)$$

$\mathbf{R} \in \mathbb{R}^{D \times D}$ is a rotation matrix formed by the eigenvectors of the scatter matrix of the data (as we will see in the Section 2.3).

2.2 Similarity measures

In order to define groups of samples, we need to use a definition of similarity and/or distance. Given two vectors \mathbf{x}_i and \mathbf{x}_j , their Euclidean distance is given by:

$$\begin{aligned} d_E(\mathbf{x}_i, \mathbf{x}_j) &= \|\mathbf{x}_i - \mathbf{x}_j\|_2 = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T (\mathbf{x}_i - \mathbf{x}_j)} \\ &= \sqrt{\mathbf{x}_i^T \mathbf{x}_i + \mathbf{x}_j^T \mathbf{x}_j - 2\mathbf{x}_i^T \mathbf{x}_j} = \sqrt{\sum_{d=1}^D (x_{id} - x_{jd})^2} \end{aligned} \quad (2.8)$$

Figure 2.1: l_2 and l_1 normalization of \mathbf{x}_i .

The Manhattan distance between \mathbf{x}_i and \mathbf{x}_j is given by:

$$d_M(\mathbf{x}_i, \mathbf{x}_j) = \sum_{d=1}^D |x_{id} - x_{jd}|, \quad (2.9)$$

where $|\cdot|$ denotes the absolute value of a scalar.

The above two distance functions are special cases of the Minkowski metric, defined as:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{d=1}^D |x_{id} - x_{jd}|^q \right)^{\frac{1}{q}}, \quad (2.10)$$

where a choice of $q = 2$ corresponds to the Euclidean and $q = 1$ corresponds to the Manhattan distance.

More generally, one can define similarity functions to compare two vectors \mathbf{x}_i and \mathbf{x}_j . Similarity functions can be defined based on a distance function, or not. For example, distance-based similarity functions are:

$$s(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{d(\mathbf{x}_i, \mathbf{x}_j)}{\sigma}}, \quad (2.11)$$

$$s(\mathbf{x}_i, \mathbf{x}_j) = \frac{\sigma}{d(\mathbf{x}_i, \mathbf{x}_j)}, \quad (2.12)$$

where $d(\cdot, \cdot)$ can be replaced with any of the above distance functions and σ is a scaling parameter. A similarity function that is not defined based on a distance function is:

$$s(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i^T \mathbf{x}_j}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_j\|_2} \quad (2.13)$$

and defines the similarity between the two vectors based on the angle they form in the D -dimensional Euclidean space.

2.3 Principal Component Analysis

Given a set of vectors $\mathbf{x}_i \in \mathbb{R}^D$, stored in the columns of a data matrix $\mathbf{X} \in \mathbb{R}^{D \times N}$ one can define a linear data transformation such that the dimensions of the transformed data will be uncorrelated. This transformation is defined in such a way that the first dimension of the transformed data has the largest possible variance (i.e. corresponds to the direction of the data having the maximal variability or dispersion) and each succeeding dimension has the highest possible variance, given the previously defined dimensions.

Let us express the linearly transformed data by $\mathbf{y}_i = \mathbf{W}^T \mathbf{x}_i$, $i = 1, \dots, N$ (or by using a matrix notation $\mathbf{Y} = \mathbf{W}^T \mathbf{X}$). The dispersion of the transformed data is expressed by:

$$\mathbf{S}_T = \sum_{i=1}^N (\mathbf{y}_i - \mathbf{m})(\mathbf{y}_i - \mathbf{m})^T = \sum_{i=1}^N \bar{\mathbf{y}}_i \bar{\mathbf{y}}_i^T = \bar{\mathbf{Y}} \bar{\mathbf{Y}}^T, \quad (2.14)$$

where $\mathbf{m} = \frac{1}{N} \sum_{i=1}^N \mathbf{y}_i$ is the mean vector of the transformed data. Since the transformed data is unknown, it is convenient to express \mathbf{S}_T as a function of \mathbf{X} , i.e.:

$$\begin{aligned} \mathbf{S}_T &= \sum_{i=1}^N [\mathbf{W}^T (\mathbf{x}_i - \boldsymbol{\mu})] [\mathbf{W}^T (\mathbf{x}_i - \boldsymbol{\mu})]^T = \sum_{i=1}^N \mathbf{W}^T \bar{\mathbf{x}}_i \bar{\mathbf{x}}_i^T \mathbf{W} \\ &= \mathbf{W}^T \bar{\mathbf{X}} \bar{\mathbf{X}}^T \mathbf{W} = \mathbf{W}^T \tilde{\mathbf{S}}_T \mathbf{W}, \end{aligned} \quad (2.15)$$

where $\tilde{\mathbf{S}}_T = \bar{\mathbf{X}} \bar{\mathbf{X}}^T$ is the scatter matrix defined on the data \mathbf{x}_i .

Thus, \mathbf{W} can be obtained by optimizing the following criterion:

$$\begin{aligned} \mathbf{W}^* &= \arg \max \text{Tr}(\mathbf{W}^T \tilde{\mathbf{S}}_T \mathbf{W}), \\ \text{subject to} & : \mathbf{W}^T \mathbf{W} = \mathbf{I}, \end{aligned} \quad (2.16)$$

where the asterisk is used to denote the optimal solution for \mathbf{W} and $\mathbf{I} \in \mathbb{R}^{D \times D}$ is the identity matrix. $\text{Tr}(\mathbf{A})$ is the trace operator, i.e. the summation of the

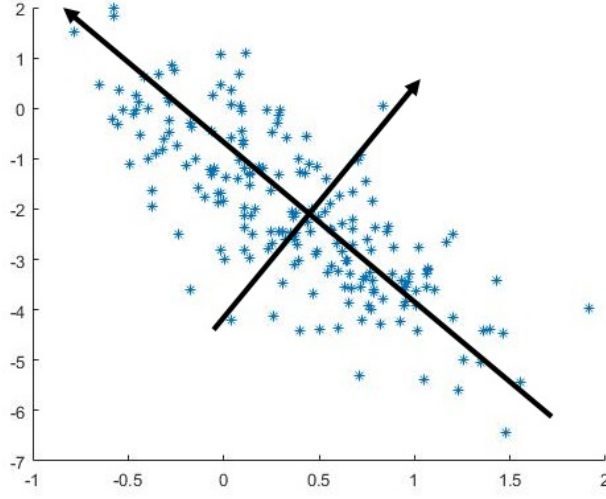


Figure 2.2: 2D data and the corresponding principal axes.

elements in the diagonal of the matrix \mathbf{A} . The constraint is imposed in order to define an orthogonal transformation. The solution of 2.16 is obtained by applying eigen-analysis to the matrix $\tilde{\mathbf{S}}_T$. \mathbf{W}^* is formed by the eigen-vectors of $\tilde{\mathbf{S}}_T$ sorted according to the descending order of the corresponding eigen-values. It can be shown that the above linear orthogonal transformation also corresponds to the transformation providing the minimal reconstruction error (in the case of centered data), i.e. solving the following optimization problem:

$$\begin{aligned} \mathbf{W}^* &= \arg \min \|\mathbf{W}\bar{\mathbf{Y}} - \bar{\mathbf{X}}\|_F^2, \\ \text{subject to } &: \mathbf{W}^T \mathbf{W} = \mathbf{I}, \end{aligned} \quad (2.17)$$

where $\|\cdot\|_F^2$ is the (squared) Frobenius norm of a matrix.

In the case where the matrix \mathbf{W} is square, i.e. when it is formed by all the eigen-vectors of $\tilde{\mathbf{S}}_T$, then the transformation $\mathbf{Y} = \mathbf{W}^T \mathbf{X}$ corresponds to a rotation of the data such that the data dimensions correspond to the principal axes (i.e. it corresponds to the matrix \mathbf{R} of Section 2.1). However, one can choose to keep fewer ($d < D$) eigenvectors, leading to the mapping of the data in a d -dimensional sub-space of \mathbb{R}^D . Given the solution of (2.17), the obtained sub-space corresponds to the one preserving most of the information of the original data \mathbf{X} .

An example of 2-dimensional data and the corresponding principal axes are illustrated in Figure 2.2.

2.4 Clustering

Given a set of unlabeled vectors \mathbf{x}_i , $i = 1, \dots, N$, we would like to determine a set of K groups, each formed by vectors which are similar to each other. We say that each group (or cluster) of vectors forms a pattern on the available data. The simplest and most widely used criterion for clustering is the sum-of-squared-error (SSE) criterion. Let us assume that cluster k is formed by N_k vectors, which are denoted by $\mathbf{x}_i^{(k)}$, $i = 1, \dots, N_k$. Here we use the superscript to denote the cluster in which a vector belongs to. The mean vector of the cluster is given by:

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i=1}^{N_k} \mathbf{x}_i^{(k)}. \quad (2.18)$$

Now, we are able to express the SSE criterion as follows:

$$\mathcal{J}_{SSE} = \sum_{k=1}^K \sum_{i=1}^{N_k} \|\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_k\|_2^2 = \sum_{k=1}^K \mathcal{J}_k. \quad (2.19)$$

The criterion \mathcal{J}_{SSE} exploits the mean cluster vector in order to *represent* a cluster. Such a choice seems natural, since in Euclidean spaces the mean vector of a group minimizes the sum of squared lengths of the ‘error’ vectors $\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_k$. This means that each of the mean cluster vectors $\boldsymbol{\mu}_k$, $k = 1, \dots, K$ is a good choice for *representing* the N_k vectors belonging to the k -th cluster, since it minimizes the variance among the various cluster vectors. This is why clusterings optimizing the criterion in (2.19) are called *minimum variance* partitions.

After defining the criterion to be optimized for defining the K clusters and the corresponding cluster mean vectors, the question remains: ‘how to optimize the criterion and find the optimal groups?’. Let us assume that we have used a process to define the cluster mean vectors $\boldsymbol{\mu}_k$, $k = 1, \dots, K$. Let us also assume that we have decided that the vector $\mathbf{x}_i^{(k)}$ needs to be removed from cluster k and be included in cluster m . This means that (after the change) the mean vector of cluster m will change as follows:

$$\boldsymbol{\mu}_m^* = \boldsymbol{\mu}_m + \frac{N_m}{N_m + 1} (\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_m), \quad (2.20)$$

while the mean vector of cluster k will change as follows:

$$\boldsymbol{\mu}_k^* = \boldsymbol{\mu}_k - \frac{N_k}{N_k - 1} (\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_k). \quad (2.21)$$

Of course, in the above we assume that cluster k is not formed by one vector before the change (clusters formed by one vector are called singletons). Also, we

should keep in mind that, after the change, the superscript of $\mathbf{x}_i^{(k)}$ will change from k to m .

Given the above change, \mathcal{J}_m and \mathcal{J}_k in 2.19 change as follows:

$$\begin{aligned}\mathcal{J}_m^* &= \sum_{j=1}^{N_m} \|\mathbf{x}_j^{(m)} - \boldsymbol{\mu}_m^*\|_2^2 + \|\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_m^*\|_2^2 \\ &= \left(\sum_{j=1}^{N_m} \|\mathbf{x}_j^{(m)} - \boldsymbol{\mu}_m - \frac{1}{N_m+1}(\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_m)\|_2^2 \right) + \left\| \frac{N_m}{N_m+1}(\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_m) \right\|_2^2 \\ &= \mathcal{J}_m + \frac{N_m}{N_m+1} \|\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_m\|_2^2\end{aligned}\tag{2.22}$$

and

$$\mathcal{J}_k^* = \mathcal{J}_k - \frac{N_k}{N_k-1} \|\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_k\|_2^2.\tag{2.23}$$

Thus, if:

$$\frac{N_k}{N_k-1} \|\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_k\|_2^2 > \frac{N_m}{N_m+1} \|\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_m\|_2^2,\tag{2.24}$$

which (for the case where cluster are balanced, i.e. they are formed by the same/similar number of vectors) is the case when $\mathbf{x}_i^{(k)}$ is closer to $\boldsymbol{\mu}_m$ than to $\boldsymbol{\mu}_k$, i.e. when:

$$\|\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_m\|_2^2 < \|\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_k\|_2^2,\tag{2.25}$$

making the change will lead to a lower value for \mathcal{J}_{SSE} . Since our objective is to minimize \mathcal{J}_{SSE} , we choose to make the change.

The above process can be used in order to optimize \mathcal{J}_{SSE} in an iterative manner as follows:

What remains in order to complete Algorithm 1 is to define a way for initializing the cluster mean vectors $\boldsymbol{\mu}_k$, $k = 1, \dots, K$. While there are some (most of which heuristic) initialization approaches, $\boldsymbol{\mu}_k$, $k = 1, \dots, K$ are usually initialized by randomly selected K training vectors. Since the above iterative optimization process is not guaranteed to produce the global optimal (i.e. overall minimal) criterion value, different random initializations are expected to lead to different clustering results. This is why usually multiple (e.g. five) random initializations are used and the clustering result corresponding to the minimum value of \mathcal{J}_{SSE} is selected.

Another way to optimize \mathcal{J}_{SSE} is to follow the batch update process described in Algorithm 2. This algorithm is faster and usually leads to better clustering results, since it is not affected by the sequence in which the vectors are used to update the clusters.

In the above, each vector is assigned to a cluster and contributes to the calculation of only one cluster mean vector (i.e. the one corresponding to the cluster

Algorithm 1: Iterative minimization of SSE for clustering

-
- 1: Initialize $\boldsymbol{\mu}_k$, $k = 1, \dots, K$
 - 2: Assign each vector \mathbf{x}_i , $i = 1, \dots, N$ to a cluster by:
 - 3: $l^* = \arg \min_l \|\mathbf{x}_i - \boldsymbol{\mu}_l\|_2^2$
 - 4:
 - 5: % Start iterative optimization
 - 6: **do** randomly select a samples $\mathbf{x}_i^{(k)}$
 - 7: **if** $N_k \neq 1$ **then** calculate
 - 8: $r_k = \frac{N_k}{N_k-1} \|(\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_k)\|_2^2$
 - 9: $r_m = \frac{N_m}{N_m+1} \|(\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_m)\|_2^2$, $m \neq k$
 - 10: Calculate the minimum value $r_q = \min r_m$, $m \neq k$
 - 11: **if** $r_q < r_k$ **then** transfer $\mathbf{x}_i^{(k)}$ to cluster q
 - 12: recompute $\boldsymbol{\mu}_k$, $\boldsymbol{\mu}_q$ and \mathcal{J}_{SSE}
 - 13: **until** no change in \mathcal{J}_{SSE} in is obtained after checking all N samples
-

Algorithm 2: K -Means clustering

-
- 1: Initialize $\boldsymbol{\mu}_k$, $k = 1, \dots, K$
 - 2: **Do**
 - 3: Assign all vectors \mathbf{x}_i , $i = 1, \dots, N$ to a cluster by:
 - 4: $l^* = \arg \min_l \|\mathbf{x}_i - \boldsymbol{\mu}_l\|_2^2$
 - 5: Update the cluster mean vectors by:
 - 6: $\boldsymbol{\mu}_k^* = \frac{1}{N_k} \sum_{\mathbf{x}_i \in D_k} \mathbf{x}_i$, $k = 1, \dots, K$
 - 7: **until** no change in $\boldsymbol{\mu}_k^*$, $k = 1, \dots, K$
-

it has been assigned to). We say that the vectors forming a cluster are the *members* of that cluster. However, there might be cases where one vector is (almost) equally far from the mean vectors of two clusters. In this case, one needs to make an arbitrary choice and assign the vector to one of these two clusters. A more plausible way to deal with such cases is to relax the membership of each vector to the various clusters. That is, for each vector \mathbf{x}_i we define a membership vector $\mathbf{a}_i \in \mathbb{R}^K$ having values that satisfy:

$$\sum_{k=1}^K a_{ik} = 1. \quad (2.26)$$

A natural way to calculate the values of the membership vector \mathbf{a}_i is the following:

$$a_{ik} = \frac{\|\mathbf{x}_i - \boldsymbol{\mu}_k\|_2^{-\gamma}}{\sum_{l=1}^K \|\mathbf{x}_i - \boldsymbol{\mu}_l\|_2^{-\gamma}}, \quad (2.27)$$

where γ is called *fuzzification parameter* and scales the Euclidean distance for the calculation of the membership values. That is, the membership of a vector is higher for the clusters it is closest to (in relation to its distance from the remaining clusters). Using the membership values a_{ik} , the cluster mean vectors are calculated by:

$$\boldsymbol{\mu}_k = \sum_{i=1}^N a_{ik} \mathbf{x}_i, \quad k = 1, \dots, K. \quad (2.28)$$

Using the membership values a_{ik} and the corresponding cluster mean vectors $\boldsymbol{\mu}_k$, the clustering criterion now becomes:

$$\mathcal{J}_{FKMeans} = \sum_{k=1}^K \sum_{i=1}^{N_k} a_{ik} \|\mathbf{x}_i^{(k)} - \boldsymbol{\mu}_k\|_2^2. \quad (2.29)$$

The criterion $\mathcal{J}_{FKMeans}$ corresponds to a fuzzy version of the K -Means algorithms and can be optimized by applying the process described in Algorithm 3.

Algorithm 3: Fuzzy K -Means clustering

- 1: Initialize $\boldsymbol{\mu}_k$, $k = 1, \dots, K$
 - 2: **Do**
 - 3: Calculate the membership vectors \mathbf{a}_i , $i = 1, \dots, N$
 - 4: Update the new cluster mean vectors $\boldsymbol{\mu}_k^*$, $k = 1, \dots, K$
 - 5: **until** no change in $\boldsymbol{\mu}_k^*$, $k = 1, \dots, K$
-

2.5 Questions and Exercises

2.1 Show that, when the training data are centered, the projection matrix obtained by applying PCA (Eq. (2.16)) solves the Regression problem of Eq. (2.17).

2.2 Cluster the following 2-dimensional data in two clusters:

$$\mathbf{X} = \begin{bmatrix} -1 & 0 & -0.5 & -1.5 & -2 & 0 & -1 & 1 & 1.3 & 0.7 & 2.5 & 0 \\ 0 & -1 & -0.5 & -1.5 & 0 & -2 & -1.3 & 1 & 0.7 & 1.3 & 1 & 1 \end{bmatrix}. \quad (2.30)$$

Initialize the cluster mean vectors by using:

$$\mathbf{M} = \begin{bmatrix} -1 & -0.9 \\ -1 & 0 \end{bmatrix} \quad (2.31)$$

and apply three iterations of the batch K -Means algorithm.

2.3 Cluster the data in 2.2 by applying the batch Fuzzy K -Means algorithm. Use the same initialization for the cluster mean vectors and a value of $\gamma = 2$.

2.4 Consider the following 2-dimensional vectors (forming the columns of \mathbf{X}

$$\mathbf{X} = \begin{bmatrix} -1 & 0 & -0.5 & -1.5 & -2 & 0 & -1 & 1 & 1.3 & 0.7 & 2.5 & 0 \\ 0 & -1 & -0.5 & -1.5 & 0 & -2 & -1.3 & 1 & 0.7 & 1.3 & 1 & 1 \end{bmatrix}. \quad (2.32)$$

1. Center the data in \mathbf{X} .
2. Standardize the data in \mathbf{X}
3. Normalize the data in \mathbf{X} using their l_2 norm.

.

2.5 Consider the following 2-dimensional vectors (forming the columns of \mathbf{X}

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 2 & 3 & 3 & 4 \\ 1 & 2 & 3 & 2 & 3 & 4 \end{bmatrix}. \quad (2.33)$$

Project (map) this data to the one-dimensional space determined by PCA.

Chapter 3

Probability-based Learning

Probability-based learning, using as a basis the Bayesian decision theory, quantifies the tradeoffs between all possible classification decisions using their probabilities and the costs that follow such decisions. Let us denote by $\mathcal{C} = \{c_1, \dots, c_K\}$ the set of all possible classification outcomes (i.e. all classes). We denote as $P(c_k)$ the *a priori probability* of class k , expressing the probability that any new sample will belong to class k . Since $P(c_k)$ express probability values, we have:

$$\sum_{k=1}^K P(c_k) = 1. \quad (3.1)$$

In order to better understand the concept of a priori probabilities, let us consider the classification problem for which all available training images are illustrated in Figure 3.1. Let us assume that the persons depicted in the first row of that figure belong to the first class (i.e. the class $c_1 = \text{young}$), while those depicted in the second row belong to the second class (i.e. the class $c_2 = \text{old}$). Supposing that the proportion of all young and old persons in the world can be expressed by the proportion of the samples belonging to the the two classes of our classification problem, then $P(c_1) = 5/10 = 0.5$ and $P(c_2) = 5/10 = 0.5$.

Given only the above information, the probability of a new (unknown) person depicted in a new facial image to belong to the first class is equal to 0.5 (which is equal to the probability of the second class too). Even in cases where $P(c_1) > P(c_2)$, classifying a new sample based only on the a priori probabilities of the two classes makes no sense, since all new samples would be classified to the first class.

Usually, decisions are made based on a measurable variable x (or a set of variables \mathbf{x}). In our example, such a variable can be the age of the person depicted in the facial image (and is illustrated above/below the corresponding facial image in Figure 3.1). We denote by $P(c_k|x)$ the *conditional probability* of class c_k , given the observation of x . In our example $P(c_1|x)$ expresses the probability of class *young* given the age of a persons (e.g. $x = 30$). In a similar manner, we

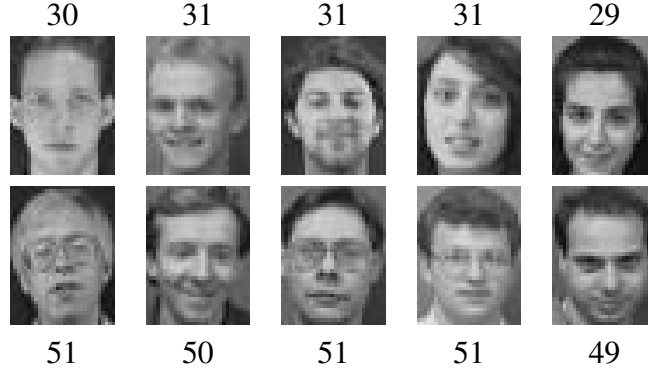


Figure 3.1: *Facial images depicting people belonging to two classes (first row: c_1 = young and second row: c_2 = old)*

can define the *class-conditional probability* $p(x|c_k)$, expressing the probability of observing x given that the sample it corresponds to belongs to class c_k ¹. In our example, $p(x|c_1)$ expresses the probability of observing an age value of x (in the real value domain, e.g. $x = 30.5$) given that the person depicted in the new facial image belongs to c_1 , i.e. is a young person.

Given the above definitions, we can define the *joint probability* of c_k and x as:

$$p(c_k, x) = P(c_k|x)p(x) = p(x|c_k)P(c_k), \quad (3.2)$$

where

$$p(x) = \sum_{k=1}^K p(x|c_k)P(c_k). \quad (3.3)$$

In order to better understand Eq. 3.2, we can see that the probability of the first image in the first row of figure 3.1 to depict a person being 30 years old and belonging to class *young* is given by:

$$p(c_1, 30) = P(c_1|30)p(30) = 1 \frac{1}{10} = \frac{1}{10}. \quad (3.4)$$

In a similar way, the same probability can be calculated using the third term in 3.2, as follows:

$$p(c_1, 30) = p(30|c_1)P(c_1) = \frac{1}{5} \frac{1}{2} = \frac{1}{10}. \quad (3.5)$$

¹Here we use capital $P(\cdot)$ to denote the probability mass function and the small $p(\cdot)$ to denote the probability density function. While their definition is similar, we can, roughly, think of the probability mass function to be used for discrete variables, while the probability density function for continuous variables.

Using the equality of the second and third terms in 3.2, we obtain:

$$P(c_k|x) = \frac{p(x|c_k)P(c_k)}{p(x)}. \quad (3.6)$$

Equation 3.6 is the Bayes' formula connecting the posterior probability of a class as a function of its a priori probability and the observation.

Given the conditional probabilities of each class $P(c_k, |x)$, $k = 1, \dots, K$, the probability of error is given by:

$$P(error|x) = \begin{cases} P(c_1|x), & \text{if } x \text{ is classified to } c_2 \\ P(c_2|x), & \text{if } x \text{ is classified to } c_1 \end{cases} \quad (3.7)$$

Assuming that the conditional probability $P(c_k|x)$ can be approximated with a continuous function, e.g. as in Figure 3.2, then the total error can be calculated by:

$$P(error) = \int_{-\infty}^{\infty} P(error, x)dx = \int_{-\infty}^{\infty} P(error|x)p(x)dx. \quad (3.8)$$

Thus, the optimal decision function corresponds to the place where:

$$P(c_1|x) = P(c_2|x). \quad (3.9)$$

Using the above, the Bayes' decision rule is given by:

$$\text{Decide } c_1 \text{ if } P(c_1|x) > P(c_2|x), \text{ otherwise decide } c_2. \quad (3.10)$$

That is, we decide that an observation x should be classified to the class it can better "describe". The above decision rule can be extended to more than two classes in a straightforward manner. That is, given a set of classes c_k , $k = 1, \dots, K$, an observation x and the corresponding conditional probabilities $P(c_k|x)$, $k = 1, \dots, K$, the decision rule is:

$$\text{Decide } c_l \text{ for which } P(c_l|x) > P(c_i|x), i \neq l. \quad (3.11)$$

The probability $p(x|c_k)$ is also called *likelihood* of c_k with respect to x . Given an observation x , we can observe that its a priori probability $p(x)$ is a constant factor (given by Eq. 3.3) scaling the probability $P(c_k|x)$ in the range of $[0, 1]$. Assuming also that $P(c_k) = \frac{1}{K}$, $k = 1, \dots, K$, then the Bayes' formula in 3.6 becomes:

$$P(c_k|x) = \alpha p(x|c_k), \quad (3.12)$$

where α is a scalar absorbing the contribution of the remaining (constant) terms. From Eq. 3.12, we can see that the probability of a class given an observation x (and equal a priori class probabilities) is proportional to the sample's likelihood. Thus, by classifying the new sample based on the highest conditional probability, corresponds to classifying the sample based on its maximum likelihood. This process is referred to as *Maximum Likelihood Classification*.

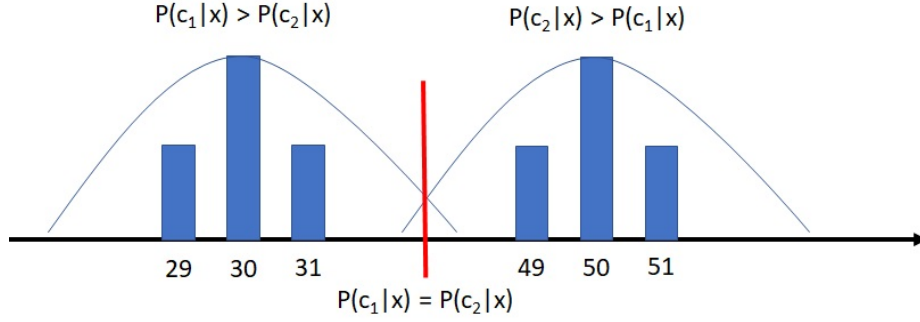


Figure 3.2: *Conditional probabilities of two classes and the corresponding decision function.*

3.1 Risk-based decision functions

Until now we discussed about the case where the decision function is a function of only one continuous variable x . Let us assume that the observation corresponds to a feature vector $\mathbf{x} \in \mathbb{R}^D$. As in the above case, the Bayes' formula holds:

$$P(c_k|\mathbf{x}) = \frac{p(\mathbf{x}|c_k)P(c_k)}{p(\mathbf{x})}, \quad (3.13)$$

where

$$p(\mathbf{x}) = \sum_{k=1}^K p(\mathbf{x}|c_k)P(c_k). \quad (3.14)$$

Suppose that given the observation \mathbf{x} , we take the action of classifying the new sample to class c_i . We call this action by α_i . We define the loss function $\lambda(\alpha_i|c_k)$ describing the loss incurred by taking the action α_i given that the correct class is c_k . Then, we can define the *risk* of taking action α_i given the observation \mathbf{x} as follows:

$$R(\alpha_i|\mathbf{x}) = \sum_{k=1}^K \lambda(\alpha_i|c_k)P(c_k|\mathbf{x}). \quad (3.15)$$

Thus, given an observation \mathbf{x} , the best action is the one minimizing the risk. The minimum overall risk is called Bayes risk and it corresponds to the best performance that can be achieved.

Let us consider the two-class classification case. Here the action α_1 corresponds to deciding that the correct class is c_1 and the action α_2 corresponds to deciding that the correct class is c_2 . From Eq. 3.15, the risks corresponding to each of these actions are given by:

$$R(\alpha_1|\mathbf{x}) = \lambda(\alpha_1|c_1)P(c_1|\mathbf{x}) + \lambda(\alpha_1|c_2)P(c_2|\mathbf{x}) \quad (3.16)$$

$$R(\alpha_2|\mathbf{x}) = \lambda(\alpha_2|c_1)P(c_1|\mathbf{x}) + \lambda(\alpha_2|c_2)P(c_2|\mathbf{x}) \quad (3.17)$$

We classify \mathbf{x} to c_1 if $R(\alpha_1|\mathbf{x}) < R(\alpha_2|\mathbf{x})$, or:

$$(\lambda(\alpha_2|c_1) - \lambda(\alpha_1|c_1)) P(c_1|\mathbf{x}) > (\lambda(\alpha_1|c_2) - \lambda(\alpha_2|c_2)) P(c_2|\mathbf{x}). \quad (3.18)$$

By substituting the posterior probabilities in Eq. 3.18 with the Bayes's formula in Eq. 3.6, the decision rule for classifying \mathbf{x} to c_1 becomes:

$$(\lambda(\alpha_2|c_1) - \lambda(\alpha_1|c_1)) p(\mathbf{x}|c_1)P(c_1) > (\lambda(\alpha_1|c_2) - \lambda(\alpha_2|c_2)) p(\mathbf{x}|c_2)P(c_2). \quad (3.19)$$

or (taking the form of the *likelihood ratio*):

$$\frac{p(\mathbf{x}|c_1)}{p(\mathbf{x}|c_2)} > \frac{(\lambda(\alpha_1|c_2) - \lambda(\alpha_2|c_2)) P(c_2)}{(\lambda(\alpha_2|c_1) - \lambda(\alpha_1|c_1)) P(c_1)}. \quad (3.20)$$

3.2 Decision functions of the Normal Density

In the case where samples are formed by variables (or features) which are continuous valued, we usually make assumptions on the distributions they are sampled from. While in the real world samples can follow any type of distribution, the Normal Distribution has received much attention due to its analytical tractability. Moreover, it has been shown that several natural phenomena can be adequately well modeled by using the Normal Distribution. As stated by the Central Limit Theorem, the aggregate effect of a large number of small, independent random disturbances will lead to a Gaussian distribution. As an example, randomly corrupted versions of a feature vector can be modeled by using a multivariate normal distribution. Given such a set of corrupted variables x , or a function of x , $f(x)$ (for $f(x) = x$ the two cases are the same), the *expected value* can be calculated as follows:

$$\mathcal{E}[f(x)] = \int_{-\infty}^{\infty} f(x)p(x)dx. \quad (3.21)$$

In the case where our variables are discrete, the integral above is replaced with the summation over all samples in the set at hand \mathcal{D} , i.e.:

$$\mathcal{E}[f(x)] = \sum_{x \in \mathcal{D}} f(x)P(x), \quad (3.22)$$

where $P(x)$ is the probability mass of x .

The Normal (or Gaussian) Density of a continuous variable x is given by:

$$p(x) = \frac{1}{\sqrt{2\pi} \sigma} \exp \left[-\frac{1}{2} \left(\frac{x - \mu}{\sigma} \right)^2 \right], \quad (3.23)$$

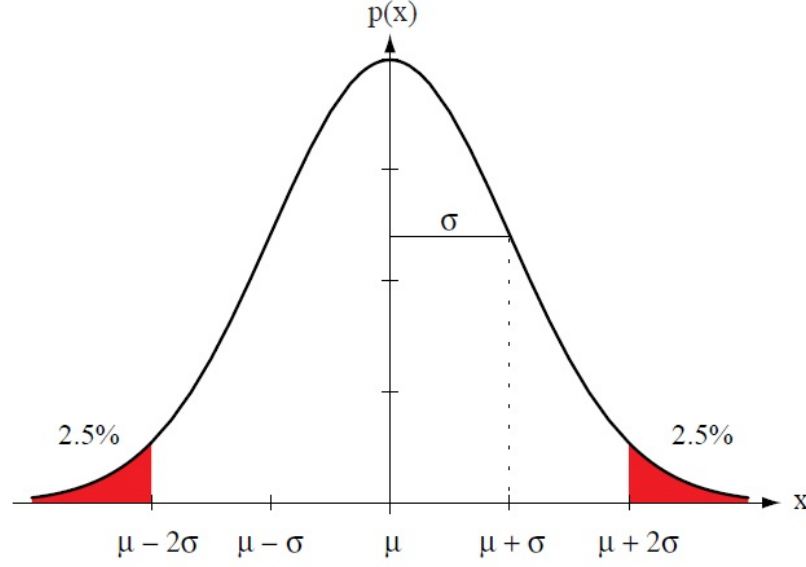


Figure 3.3: Normal Density of a real-valued variable x .

where $\pi \simeq 3.14159$, and μ and σ express the mean value and the standard deviation of the distribution, and σ^2 is the corresponding variance:

$$\mu = \mathcal{E}[x] = \int_{-\infty}^{\infty} xp(x)dx, \quad (3.24)$$

and

$$\sigma^2 = \mathcal{E}[(x - \mu)^2] = \int_{-\infty}^{\infty} (x - \mu)^2 p(x)dx. \quad (3.25)$$

We say that a variable x satisfying the above follows a Normal Distribution or has a Normal Density. Using these two parameters (i.e. μ and σ) the Normal Distribution can be completely specified. We use the notation $p(x) \sim N(\mu, \sigma^2)$ in order to denote that x follows a Normal Distribution *centered* at μ and having variance σ^2 . These two parameters also define properties of the Normal Distribution described in Figure 3.3.

In the case where our samples are represented by D -dimensional vectors $\mathbf{x} \in \mathbb{R}^D$ following a multivariate Normal Distribution, the above notations can be extended accordingly. In this case:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left[-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right], \quad (3.26)$$

where $|\cdot|$ denotes the determinant of a matrix, and $\boldsymbol{\mu} \in \mathbb{R}^D$ and $\Sigma \in \mathbb{R}^{D \times D}$ are the mean vector and the covariance matrix of the Normal Density, respectively:

$$\boldsymbol{\mu} = \mathcal{E}[\mathbf{x}] = \int \mathbf{x} p(\mathbf{x}) d\mathbf{x} \quad (3.27)$$

and

$$\Sigma = \mathcal{E}[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T] = \int (\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T p(\mathbf{x}) d\mathbf{x}. \quad (3.28)$$

The above notations can be calculated by using 1-D calculations (like in the case of real-valued Normal Density) as follows:

$$\mu_d = \mathcal{E}[x_d] \quad (3.29)$$

and

$$\Sigma_{ij} = \mathcal{E}[(x_i - \mu_i)(x_j - \mu_j)]. \quad (3.30)$$

By defining $\boldsymbol{\mu}$ and Σ , the multivariate Normal Density $p(\mathbf{x}) \sim N(\boldsymbol{\mu}, \Sigma)$ can be completely specified.

The covariance matrix Σ defines some important properties of the distribution. Since it is defined over outer products of vectors, it can be shown that it is a positive semi-definite matrix. In the case where Σ is positive definite, then $|\Sigma| > 0$. The diagonal elements Σ_{ii} are the variances of each dimension, while the off-diagonal elements Σ_{ij} are the covariances of the various dimensions of the feature space. If the i -th and j -th dimension of the feature space are statistically independent, then $\Sigma_{ij} = 0$. If all off-diagonal elements are zero, i.e. $\Sigma_{ij} = 0$, $i \neq j$, then the multivariate Normal Distribution degenerates to a product of D Normal Distributions defined on 1D (real-valued) variables.

Sometimes it is convenient to transform a Normal Density $N(\boldsymbol{\mu}, \Sigma)$ to another one $N(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma})$ satisfying $\tilde{\Sigma} = c\mathbf{I}$, where $\mathbf{I} \in \mathbb{R}^{D \times D}$ is the identity matrix. This process is called *whitening*. In order to do so, we apply eigenanalysis to the matrix Σ . Let us denote this decomposition by $\Sigma = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$, where $\mathbf{U} \in \mathbb{R}^{D \times D}$ is a matrix whose columns are formed by the eigenvectors of Σ and $\mathbf{\Lambda} \in \mathbb{R}^{D \times D}$ is a diagonal matrix formed by the corresponding eigenvalues. Then, the matrix $\mathbf{W} = \mathbf{U}\mathbf{\Lambda}^{-\frac{1}{2}}$ can be used in order to transform the data, i.e. $\mathbf{y} = \mathbf{W}^T \mathbf{x}$ for which $p(\mathbf{y}) \sim N(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma})$. Here, we should note the similarity between the whitening transform for the Normal Density and the PCA transform described in Section 2.3.

Another interesting property of Σ is that it can be used in order to define a distance metric taking into account the scaling of the various feature dimensions (as well as the covariances between them) as follows:

$$d_M(\mathbf{x} - \boldsymbol{\mu}) = (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}). \quad (3.31)$$

The above distance is usually called *Mahalanobis distance* between vectors \mathbf{x} and $\boldsymbol{\mu}$. It can be seen that the Mahalanobis distance between two vectors after applying a whitening transform takes into account only the scaling of each feature dimension.

Let us now consider a two-class classification problem, in which we assume that data of each class follows a Normal Density. This means that we assume the following:

$$\begin{aligned} p(\mathbf{x}|c_1) &\sim N(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \\ p(\mathbf{x}|c_2) &\sim N(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2). \end{aligned} \quad (3.32)$$

According to the Bayes' decision rule in Eq. 3.10, given an observation \mathbf{x} , we should:

$$\text{decide } c_1 \text{ if } P(c_1|\mathbf{x}) > P(c_2|\mathbf{x}), \text{ otherwise decide } c_2. \quad (3.33)$$

Using the Bayes' formula in Eq. 3.6 the decision rule becomes:

$$\text{decide } c_1 \text{ if } p(\mathbf{x}|c_1)P(c_1) > p(\mathbf{x}|c_2)P(c_2), \text{ otherwise decide } c_2, \quad (3.34)$$

where we have discarded the term $p(\mathbf{x})$, since it is positive and appears in both sides of the inequality.

In order to make easier the calculations, we combine (3.33) with a monotonic increasing function, so that the inequality in Eq. 3.33 remains the same. That is, for $P(c_1|\mathbf{x}) > P(c_2|\mathbf{x})$, then $g(P(c_1|\mathbf{x})) > g(P(c_2|\mathbf{x}))$ if $g(\cdot)$ is a monotonic increasing function. Now, the decision rule in Eq. 3.34 can be expressed as follows:

$$\text{decide } c_1 \text{ if } g(p(\mathbf{x}|c_1)P(c_1)) > g(p(\mathbf{x}|c_2)P(c_2)), \text{ otherwise decide } c_2. \quad (3.35)$$

or

$$\text{decide } c_1 \text{ if } f(\mathbf{x}|c_1) > f(\mathbf{x}|c_2), \text{ otherwise decide } c_2. \quad (3.36)$$

Since $P(c_k)$ are real values and $p(\mathbf{x}|c_k)$ follows a Normal Distribution, we choose $g(x) = \ln(x)$. This leads to the following:

$$f(\mathbf{x}|c_k) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) - \frac{D}{2} \ln(2\pi) - \frac{1}{2} \ln(|\boldsymbol{\Sigma}_k|) + \ln(P(c_k)), \quad (3.37)$$

which highly simplifies computations.

In the following, we will consider only the case where $\boldsymbol{\Sigma}_k = \sigma^2 \mathbf{I}$, since it has a special interest. In this case, we assume that all classes are whitened (i.e. all the dimensions are of equal importance). The determinant and inverse of the covariance matrices are equal to $|\boldsymbol{\Sigma}_k| = \sigma^{2D}$ and $\boldsymbol{\Sigma}_k^{-1} = \frac{1}{\sigma^2} \mathbf{I}$, respectively. Thus:

$$f(\mathbf{x}|c_k) = -\frac{1}{2\sigma^2}(\mathbf{x} - \boldsymbol{\mu}_k)^T(\mathbf{x} - \boldsymbol{\mu}_k) - \frac{D}{2} \ln(2\pi) - \frac{1}{2} \ln(\sigma^{2D}) + \ln(P(c_k)), \quad (3.38)$$

By substituting Eq. 3.38 in Eq. 3.34, we should decide c_1 if:

$$-\frac{1}{2\sigma^2}(\mathbf{x} - \boldsymbol{\mu}_1)^T(\mathbf{x} - \boldsymbol{\mu}_1) + \ln(P(c_1)) > -\frac{1}{2\sigma^2}(\mathbf{x} - \boldsymbol{\mu}_2)^T(\mathbf{x} - \boldsymbol{\mu}_2) + \ln(P(c_2)). \quad (3.39)$$

or

$$-\frac{1}{2\sigma^2}\|\mathbf{x} - \boldsymbol{\mu}_1\|_2^2 + \ln(P(c_1)) > -\frac{1}{2\sigma^2}\|\mathbf{x} - \boldsymbol{\mu}_2\|_2^2 + \ln(P(c_2)). \quad (3.40)$$

In the case where $P(c_1) = P(c_2)$, \mathbf{x} is classified to the class that corresponds to the mean vector closest to \mathbf{x} . This classification rule is well known as the Nearest Centroid classifier.

3.3 Discrete Features

In the above, we assumed that the feature vectors \mathbf{x} are formed by continuous variables in a D -dimensional space. However, in many cases the dimensions of \mathbf{x} can be discrete values. In such cases the integrals become summations, i.e.:

$$\int p(\mathbf{x}|c_k)d\mathbf{x} \rightarrow \sum_{\mathbf{x}} P(\mathbf{x}|c_k). \quad (3.41)$$

The Bayes' formula involves probabilities instead of probability densities, i.e.:

$$P(c_k|\mathbf{x}) = \frac{P(\mathbf{x}|c_k)P(c_k)}{P(\mathbf{x})}, \quad (3.42)$$

where

$$P(\mathbf{x}) = \sum_{k=1}^K P(\mathbf{x}|c_k)P(c_k). \quad (3.43)$$

3.4 Maximum Likelihood Estimation

In practical Pattern Recognition problems, we rarely know the probabilities involved in the above described analysis. In such cases, the only information we have is a set of samples (each represented by a vector) and the corresponding labels (either class or group labels). Given this information we need to *estimate* the values of the probabilities involved in our decision functions. While in most cases the a priori probabilities $P(c_k)$ are relatively easy to estimate, the conditional probabilities $p(\mathbf{x}|c_k)$ are generally difficult to estimate. In order to make the estimation process easier, we restrict our solution by setting assumptions on the distributions of the various parameters. For example, we can assume that $p(\mathbf{x}|c_k)$ follows a Normal Distribution with mean vector $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$. The parameters of the chosen distribution are now forming the parameters $\boldsymbol{\theta}_k$ we want to estimate. Another assumption that reduces the complexity of the problem is to assume that the distributions of each class can be estimated independently.

In such a case, we split our problem in K sub-problems, each formed by a sub-set of the available samples.

For each sub-problem \mathcal{D}_k , we assume that the corresponding samples have been drawn independently according to a probability distribution $p(\mathbf{x}|c_k)$. We call this type of samples as *i.i.d.* (independent identically distributed) random variables. Given the i.i.d. samples of \mathcal{D}_k and the corresponding parameters θ_k , we have:

$$p(\mathcal{D}_k|\theta_k) = \prod_{i=1}^N p(\mathbf{x}_i|\theta_k). \quad (3.44)$$

As we saw in the beginning of this Chapter, $p(\mathcal{D}_k|\theta_k)$ is called likelihood of θ_k with respect to the set of samples $\mathbf{x}_i \in \mathcal{D}_k$. The *Maximum Likelihood Estimate* of θ_k is, thus, the value of θ_k^* that maximizes $p(\mathcal{D}_k|\theta_k)$.

If $p(\mathcal{D}_k|\theta_k)$ is a differentiable function of θ_k , then θ_k^* can be found by the standard methods of differential calculus. That is, we can define a function:

$$L(\theta_k) = \ln(p(\mathcal{D}_k|\theta_k)) = \sum_{i=1}^N \ln(p(\mathbf{x}_i|\theta_k)), \quad (3.45)$$

and θ_k^* can be obtained by solving the following optimization problem:

$$\theta_k^* = \arg \max_{\theta_k} L(\theta_k), \quad (3.46)$$

or

$$\theta_k^* = \arg \max_{\theta_k} \sum_{i=1}^N \ln(p(\mathbf{x}_i|\theta_k)). \quad (3.47)$$

Thus, θ_k^* can be obtained by calculating the derivative of $L(\theta_k)$ and setting it equal to zero:

$$\nabla_{\theta_k} L(\theta_k) = \mathbf{0}. \quad (3.48)$$

As an example, let us assume that the samples of the sub-problem \mathcal{D}_k follow a Normal Density with mean value μ_k and variance σ_k^2 , which are unknown. The set of parameters to be estimated is now $\theta_k = [\mu_k \ \sigma_k^2]^T$. For one sample x_i we have:

$$L(\theta_k) = \ln(p(x_i|\theta_k)) = -\frac{1}{2} \ln(2\pi\theta_{k,2}) - \frac{1}{2\theta_{k,2}}(x_i - \theta_{k,1})^2, \quad (3.49)$$

where we set $\theta_{k,1} = \mu_k$ and $\theta_{k,2} = \sigma_k^2$. The partial derivatives with respect to $\theta_{k,1}$ and $\theta_{k,2}$ are:

$$\frac{\partial L(\theta_k)}{\partial \theta_{k,1}} = \frac{1}{\theta_{k,2}}(x_i - \theta_{k,1}) \quad (3.50)$$

and

$$\frac{\partial L(\boldsymbol{\theta}_k)}{\partial \theta_{k,2}} = -\frac{1}{2\theta_{k,2}} + \frac{(x_i - \theta_{k,1})^2}{2\theta_{k,2}^2} \quad (3.51)$$

By aggregating for all samples and setting equal to zero, we obtain:

$$\sum_{i=1}^N \frac{1}{\theta_{k,2}^*} (x_i - \theta_{k,1}^*) = 0 \quad (3.52)$$

and

$$\sum_{i=1}^N \left(\frac{(x_i - \theta_{k,1}^*)^2}{\theta_{k,2}^{*2}} - \frac{1}{\theta_{k,2}^*} \right) = 0, \quad (3.53)$$

leading to:

$$\mu = \frac{1}{N} \sum_{i=1}^N x_i \quad (3.54)$$

and

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2. \quad (3.55)$$

3.5 Expectation-Maximization

Sometimes, in order to solve a problem in which there are more than one parameters we need to apply an alternating iterative optimization process. At each step of this process, we fix all parameters except one, and optimize a reduced criterion in order to find a better value for the overall optimization problem. This process is repeated multiple times, on each of which a different parameter is optimized and ends when no change is observed on the overall optimization criterion. The above described process, can be sometimes described by using probabilistic terms and is called *Expectation-Maximization (EM)*.

As an example, let us describe the K -Means clustering algorithm (Section 2.4, Algorithm 2) as an Expectation-Maximization process. At each iteration t , given the cluster labels $l_i(t-1)$ corresponding to each vector \mathbf{x}_i , $i = 1, \dots, N$ and the cluster mean vectors $\boldsymbol{\mu}_k(t-1)$, $k = 1, \dots, K$, corresponding to the previous iteration, K -Means applies the following two processes:

1. assign new labels $l_i(t)$ to all vectors \mathbf{x}_i , $i = 1, \dots, N$ and
2. update the the cluster mean vectors $\boldsymbol{\mu}_k(t)$, $k = 1, \dots, K$ using $l_i(t)$, $i = 1, \dots, N$.

Step 1 corresponds to the Expectation step of EM, i.e. given the cluster mean vectors $\boldsymbol{\mu}_k(t-1)$, $k = 1, \dots, K$, each vector \mathbf{x}_i is expected to belong to the cluster for which the distance from \mathbf{x}_i to the corresponding cluster mean vector is minimal. After finishing this processing step, the Maximization step of EM follows, which states that (under the assumption that samples of each cluster follow a Normal Distribution) the optimal vector for representing each cluster is the mean vector calculated over the vectors belonging to the corresponding cluster (as we have seen in Section 3.4). The EM-based optimization process of K -Means clustering is shown in Algorithm 4.

Algorithm 4: EM-based K -Means

- 1: Initialize the parameters $\boldsymbol{\mu}_k(0)$, $k = 1, \dots, K$, $t = 0$
 - 2: **Do** $t \leftarrow t + 1$
 - 3: **E step:** calculate the expected labels $l_i(t)$, $i = 1, \dots, N$
 - 4: **M step:** maximize the conditional probabilities:
 - 5: $p(\mathcal{D}_k | \boldsymbol{\mu}_k)(t-1)$, $k = 1, \dots, K$
 - 6: **until** no change in $\boldsymbol{\mu}_k$, $k = 1, \dots, K$
-

3.6 Questions and Exercises

3.1 Two classes formed by the following samples:

$$c_1 = \begin{bmatrix} -1 & 0 & -0.5 & -1.5 & -2 & 0 & -1 \\ 0 & -1 & -0.5 & -1.5 & 0 & -2 & -1.3 \end{bmatrix} \quad (3.56)$$

$$c_2 = \begin{bmatrix} 1 & 1.3 & 0.7 & 2.5 & 0 \\ 1 & 0.7 & 1.3 & 1 & 1 \end{bmatrix}. \quad (3.57)$$

The probability of a sample x given class c_k , $k = 1, 2$ is calculated as a function of the relative distance of x from the corresponding class mean vector \mathbf{m}_k , i.e.:

$$p(\mathbf{x} | c_k) = \frac{e^{-\|\mathbf{x} - \mathbf{m}_k\|_2}}{\sum_{l=1}^K e^{-\|\mathbf{x} - \mathbf{m}_l\|_2}}. \quad (3.58)$$

Classify the following samples:

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_5] = \begin{bmatrix} 0 & 1 & -1 & 0.7 & -0.2 \\ 0 & 1 & 0 & -0.2 & 1.5 \end{bmatrix}. \quad (3.59)$$

3.2 Two classes formed by the following samples:

$$c_1 = \begin{bmatrix} -1 & 0 & -0.5 & -1.5 & -2 & 0 & -1 \\ 0 & -1 & -0.5 & -1.5 & 0 & -2 & -1.3 \end{bmatrix} \quad (3.60)$$

$$c_2 = \begin{bmatrix} 1 & 1.3 & 0.7 & 2.5 & 0 \\ 1 & 0.7 & 1.3 & 1 & 1 \end{bmatrix}. \quad (3.61)$$

The probability of a sample x given class c_k , $k = 1, 2$ is calculated as a function of the relative distance of x from the corresponding class mean vector \mathbf{m}_k , i.e.:

$$p(\mathbf{x}|c_k) = \frac{e^{-\|\mathbf{x}-\mathbf{m}_k\|_2}}{\sum_{l=1}^K e^{-\|\mathbf{x}-\mathbf{m}_l\|_2}}. \quad (3.62)$$

Moreover, the loss function $\lambda(\alpha_j|c_k)$ is given by:

$$\mathbf{\Lambda} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \quad (3.63)$$

Classify the following samples:

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_5] = \begin{bmatrix} 0 & 1 & -1 & 0.7 & -0.2 \\ 0 & 1 & 0 & -0.2 & 1.5 \end{bmatrix}. \quad (3.64)$$

3.3 Solve the classification problem in 3.2 using the following risks:

$$\mathbf{\Lambda} = \begin{bmatrix} 0.3 & 0.8 \\ 0.7 & 0.2 \end{bmatrix}. \quad (3.65)$$

3.4 Two classes formed by the following samples:

$$c_1 = \begin{bmatrix} -1 & 0 & -0.5 & -1.5 & -2 & 0 & -1 \\ 0 & -1 & -0.5 & -1.5 & 0 & -2 & -1.3 \end{bmatrix} \quad (3.66)$$

$$c_2 = \begin{bmatrix} 1 & 1.3 & 0.7 & 2.5 & 0 \\ 1 & 0.7 & 1.3 & 1 & 1 \end{bmatrix}. \quad (3.67)$$

We assume that each class follows a Normal Distribution with covariance matrix:

$$\Sigma_1 = \Sigma_2 = \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \quad (3.68)$$

The probability of a sample x given class c_k , $k = 1, 2$ is calculated as a function of the relative distance of x from the corresponding class mean vector \mathbf{m}_k , i.e.:

$$p(\mathbf{x}|c_k) = \frac{e^{-\|\mathbf{x}-\mathbf{m}_k\|_2}}{\sum_{l=1}^K e^{-\|\mathbf{x}-\mathbf{m}_l\|_2}}. \quad (3.69)$$

Classify the following samples:

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_5] = \begin{bmatrix} 0 & 1 & -1 & 0.7 & -0.2 \\ 0 & 1 & 0 & -0.2 & 1.5 \end{bmatrix}. \quad (3.70)$$

3.5 Solve the classification problem in 3.4 using the covariance matrix:

$$\Sigma = \frac{1}{2}(\Sigma_1 + \Sigma_2), \quad (3.71)$$

where Σ_k is the actual covariance matrix of class c_k .

Chapter 4

Linear Methods

Given a set of N samples, each represented by a vector $\mathbf{x}_i \in \mathbb{R}^D$, and the corresponding labels l_i we can define three classifiers that can be used in order to classify a new (unknown) sample $\mathbf{x}_* \in \mathbb{R}^D$. The first one is called *Nearest Class Centroid* (NCC) classifier. NCC represents each class c_k with the corresponding mean class vector:

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i, l_i=k} \mathbf{x}_i, \quad k = 1, \dots, K. \quad (4.1)$$

Given the class mean vectors, \mathbf{x}_* is classified to the class c_k corresponding to the smallest distance:

$$d(\mathbf{x}_*, \boldsymbol{\mu}_k) = \|\mathbf{x}_* - \boldsymbol{\mu}_k\|_2^2. \quad (4.2)$$

As we have seen in 3.2, the NCC classifier corresponds to the case where we set the assumption that each class is unimodal and follows a Normal Distribution.

For the cases where each class forms several subclasses, a variant of NCC called Nearest Subclass Classifiers (NSC) can be used. NSC assumes that each subclass m of class c_k follows a Normal Density and is represented by the mean subclass vector $\boldsymbol{\mu}_{km}$:

$$\boldsymbol{\mu}_{km} = \frac{1}{N_{km}} \sum_{i, l_i=k, q_i=m} \mathbf{x}_i. \quad (4.3)$$

Here we use the label q_i in order to denote the subclass label of vector \mathbf{x}_i and N_{km} to denote the number of samples forming subclass m of class c_k . Given the subclass mean vectors, \mathbf{x}_* is classified to the class c_k corresponding to the smallest distance:

$$d(\mathbf{x}_*, \boldsymbol{\mu}_{km}) = \|\mathbf{x}_* - \boldsymbol{\mu}_{km}\|_2^2. \quad (4.4)$$

In order to define the subclasses of each class, we usually apply a clustering algorithm (e.g. K -Means) using the samples of the corresponding class. The number

of subclasses per class is a parameter of the NSC and needs to be decided beforehand.

In the extreme case where the number of subclasses per class is equal to the number of its samples, then the resulting classifier is called Nearest Neighbor (NN) classifier. That is, NN classifier classifies \mathbf{x}_* to the class of the sample closest to it.

4.1 Fisher Discriminant Analysis

When the number of dimensions of the feature space in which the data representations live in is high (comparable, or even higher than the number of samples), the application of statistical techniques is problematic. This is, roughly, because the number of parameters is higher than the number of observations, and thus, their estimation is infeasible. This problem is usually called as *curse of dimensionality*. For this reason, dimensionality reduction techniques, i.e. techniques that can find an optimal (with respect to an associated criterion) mapping from the original feature space \mathbb{R}^D to a low-dimensional feature space \mathbb{R}^d , $d < D$ are important.

Let us assume that the our problem involves two classes c_1 and c_2 and we would like to define a *linear transformation* (or *linear projection*) mapping the vectors $\mathbf{x}_i \in \mathbb{R}^D$, $i = 1, \dots, N$ to the one-dimensional space. Such a projection is defined by a vector $\mathbf{w} \in \mathbb{R}^D$, such that:

$$y_i = \mathbf{w}^T \mathbf{x}_i. \quad (4.5)$$

We say that $y_i \in \mathbb{R}$ is the image of \mathbf{x}_i in \mathbb{R} using the projection \mathbf{w} .

Suppose that the class labels of all samples are given and noted by l_i , $i = 1, \dots, N$. That is, $l_i = 1$ if \mathbf{x}_i belongs to class c_1 and $l_i = 2$ if \mathbf{x}_i belongs to class c_2 . Given the class labels, the class mean vectors can be calculated as follows:

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i, l_i=k} \mathbf{x}_i, \quad (4.6)$$

where N_k is the number of samples belonging to class c_k . Using Eq. 4.5 and Eq. 4.6, the mean value of each class in \mathbb{R} is given by:

$$m_k = \frac{1}{N_k} \sum_{i, l_i=k} y_i = \frac{1}{N_k} \sum_{i, l_i=k} \mathbf{w}^T \mathbf{x}_i = \mathbf{w}^T \boldsymbol{\mu}_k. \quad (4.7)$$

Assuming that the two classes follow Normal Distributions, the distance between the two projected means:

$$|m_1 - m_2| = |\mathbf{w}^T (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)| \quad (4.8)$$

is a good indicator of the discrimination of the two classes, since the higher the distance (compared to the variances of the two classes) the more discriminated will be the two classes in \mathbb{R} . The variances of the two classes in \mathbb{R} are given by:

$$\sigma_k^2 = \frac{1}{N_k} \sum_{i, l_i=k} (y_i - m_k)^2. \quad (4.9)$$

Fisher Linear Discriminant determines the optimal projection vector \mathbf{w} by maximizing the following criterion:

$$\mathcal{J} = \frac{(m_1 - m_2)^2}{\sigma_1^2 + \sigma_2^2}. \quad (4.10)$$

In order to optimize \mathcal{J} , we need to express it in terms of \mathbf{w} and \mathbf{x}_i , $i = 1, \dots, N$. To do so, we define the class scatter matrices (we usually drop the scaling factor $1/N_k$):

$$\mathbf{S}_k = \sum_{i, l_i=k} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T, \quad (4.11)$$

and the within-class scatter matrix:

$$\mathbf{S}_w = \mathbf{S}_1 + \mathbf{S}_2 = \sum_{k=1}^2 \sum_{i, l_i=k} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T. \quad (4.12)$$

Then:

$$\begin{aligned} \sigma^2 &= \sigma_1^2 + \sigma_2^2 = \sum_{k=1}^2 \sum_{i, l_i=k} (\mathbf{w}^T \mathbf{x}_i - \mathbf{w}^T \boldsymbol{\mu}_k)^2 \\ &= \sum_{k=1}^2 \sum_{i, l_i=k} \mathbf{w}^T (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T \mathbf{w} = \mathbf{w}^T \mathbf{S}_w \mathbf{w}. \end{aligned} \quad (4.13)$$

Also:

$$\begin{aligned} (m_1 - m_2)^2 &= (\mathbf{w}^T \boldsymbol{\mu}_1 - \mathbf{w}^T \boldsymbol{\mu}_2)^2 \\ &= \mathbf{w}^T (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \mathbf{w} \\ &= \mathbf{w}^T \mathbf{S}_b \mathbf{w}, \end{aligned} \quad (4.14)$$

where we have set:

$$\mathbf{S}_b = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T. \quad (4.15)$$

Thus, Eq. 4.10 can now be expressed as:

$$\mathcal{J}(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_b \mathbf{w}}{\mathbf{w}^T \mathbf{S}_w \mathbf{w}}. \quad (4.16)$$

The solution of 4.16 is given by solving the generalized eigenanalysis problem:

$$\mathbf{S}_b \mathbf{w} = \lambda \mathbf{S}_w \mathbf{w}, \quad (4.17)$$

or (assuming that \mathbf{S}_w is non-singular):

$$\mathbf{S}_w^{-1} \mathbf{S}_b \mathbf{w} = \lambda \mathbf{w} \quad (4.18)$$

and is equal to:

$$\mathbf{w} = \mathbf{S}_w^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2). \quad (4.19)$$

Here we should note that, for the two-class case, the rank of \mathbf{S}_b is equal to one (because it is calculated as an outer product of one vector) restricting the number of possible solutions to one.

4.2 Linear Discriminant Analysis

Linear Discriminant Analysis extends Fisher Linear Discriminant in problems involving more than two classes. In order to do so, let us define the total scatter matrix as follows:

$$\begin{aligned} \mathbf{S}_T &= \sum_{k=1}^K \sum_{i, l_i=k} (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T \\ &= \sum_{k=1}^K \sum_{i, l_i=k} (\mathbf{x}_i - \boldsymbol{\mu}_k + \boldsymbol{\mu}_k - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu}_k + \boldsymbol{\mu}_k - \boldsymbol{\mu})^T \\ &= \sum_{k=1}^K \sum_{i, l_i=k} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T + \sum_{k=1}^K \sum_{i, l_i=k} N_k (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T \\ &= \mathbf{S}_w + \sum_{k=1}^K N_k (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T \\ &= \mathbf{S}_w + \mathbf{S}_b. \end{aligned} \quad (4.20)$$

Thus, in this case we have:

$$\mathbf{S}_b = \sum_{k=1}^K N_k (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T, \quad (4.21)$$

which is a matrix with rank equal to $K - 1$, restricting the number of possible solutions to $K - 1$.

The optimal projection matrix $\mathbf{W} \in \mathbb{R}^{D \times K-1}$ can be, thus, calculated by optimizing the criterion:

$$\mathcal{J}(\mathbf{W}) = \frac{\text{Tr}(\mathbf{W}^T \mathbf{S}_b \mathbf{W})}{\text{Tr}(\mathbf{W}^T \mathbf{S}_w \mathbf{W})}, \quad (4.22)$$

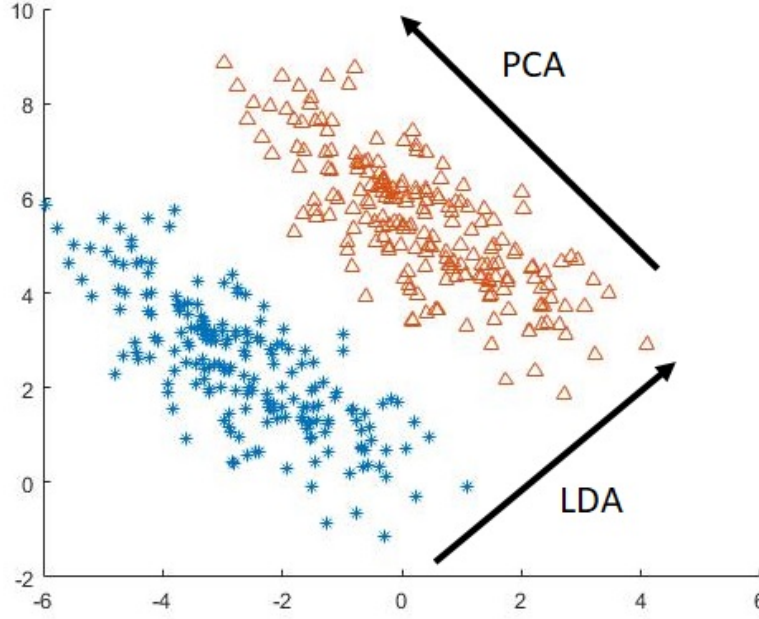


Figure 4.1: Two classes and the corresponding PCA and LDA axes.

where $Tr(\cdot)$ denotes the trace operator of a matrix. The solution of Eq. 4.22 is given by sequentially applying generalized eigenanalysis:

$$\mathbf{S}_b \mathbf{w} = \lambda \mathbf{S}_w \mathbf{w} \quad (4.23)$$

and keeping the eigenvectors corresponding to the maximal $K - 1$ eigenvalues in order to form the columns of \mathbf{W} .

An example of 2-dimensional data forming two classes and the corresponding LDA axis are illustrated in Figure 4.1. As can be seen, by linearly mapping the data on the LDA axis leads to perfect discrimination. On the other hand, PCA indicating the direction of the maximal variance is not able to provide a good discrimination result.

After obtaining \mathbf{W} , the image of \mathbf{x}_* in \mathbb{R}^d is given by:

$$\mathbf{y}_* = \mathbf{W}^T \mathbf{x}_*. \quad (4.24)$$

4.3 Linear Discriminant Functions

In this Chapter, we will focus our attention on decision functions that can be expressed as a linear combination of the features forming the data representations $\mathbf{x} \in \mathbb{R}^D$. As we will see in the following, this will lead to decision functions

corresponding to a hyperplane in \mathbb{R}^D (a plane in the case of $D = 3$ and a line in the case of $D = 2$) discriminating the samples belonging to different classes. We will study the two-class classification case first, and we will show how a two-class discriminant function can be generalized to problems formed by multiple classes later. Since discriminant functions are directly used for deriving a decision/decisions on new samples, we use the terms discriminant function and decision function, interchangeably.

A linear discriminant function applied to a vector $\mathbf{x} \in \mathbb{R}^D$ can be written as:

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = \sum_{d=1}^D w_d x_d + w_0. \quad (4.25)$$

In the above, $\mathbf{w} \in \mathbb{R}^D$ and w_0 are the so-called *weight vector* and *bias*, respectively. In general, \mathbf{w} expresses the orientation of the discriminant hyperplane, while w_0 expresses the displacement of the hyperplane from the origin.

The equation $g(\mathbf{x}) = 0$ defines the decision function discriminating the two classes c_1 and c_2 . Let us assume that two vectors, \mathbf{x}_1 and \mathbf{x}_2 , are points on the decision function, i.e.:

$$g(\mathbf{x}_1) = \mathbf{w}^T \mathbf{x}_1 + w_0 = 0 = \mathbf{w}^T \mathbf{x}_2 + w_0 = g(\mathbf{x}_2), \quad (4.26)$$

then

$$\mathbf{w}^T (\mathbf{x}_1 - \mathbf{x}_2) = 0 \quad (4.27)$$

which means that \mathbf{w} is normal (orthogonal) to any vector lying in the hyperplane.

The discriminant function $g(\mathbf{x})$ divides the feature space into two regions:

- if $g(\mathbf{x}) > 0$, then we decide that \mathbf{x} belongs to class c_1 ,
- if $g(\mathbf{x}) < 0$, then we decide that \mathbf{x} belongs to class c_2 ,
- in the case of $g(\mathbf{x}) = 0$, \mathbf{x} can be arbitrarily classified to either to c_1 or to c_2 , or we can say that \mathbf{x} belongs to an ambiguous region of the feature space.

Due to the above (binary) form of the discriminant function, two-class classification problems are usually called as *binary classification problems*.

In order to get a more compact (and in some cases convenient) notation, in the following, we will use an augmented version of the weight vector including the bias as an additional dimension, i.e.:

$$\tilde{\mathbf{w}} \leftarrow [w_0 \ \mathbf{w}^T]^T. \quad (4.28)$$

By using an augmented version of \mathbf{x} too, i.e.:

$$\tilde{\mathbf{x}} \leftarrow [1 \ \mathbf{x}^T]^T, \quad (4.29)$$

Eq. 4.25 becomes:

$$g(\mathbf{x}) = \tilde{\mathbf{w}}^T \tilde{\mathbf{x}}. \quad (4.30)$$

In the following (unless otherwise stated) we will use the symbols \mathbf{w} and \mathbf{x} in order to denote the augmented versions of the weight and a sample, respectively.

After defining the form of the discriminant function, we would like to determine the values of the weight vector $\tilde{\mathbf{w}}$, so that it can be used to correctly classify any D -dimensional vector to one of the two classes forming our classification problem. Let us assume that a set of vectors $\mathbf{x}_i \in \mathbb{R}^D$, each of which is followed by a (binary) label l_i , is available. Due to the binary form of the classification problem, it is usually chosen to express the labels l_i , $i = 1, \dots, N$ in the form:

$$\begin{aligned} l_i &= 1 && \text{, if } \mathbf{x}_i \text{ belongs to class } c_1 \text{ and} \\ l_i &= -1 && \text{, if } \mathbf{x}_i \text{ belongs to class } c_2. \end{aligned}$$

Given this set of vectors we can determine an optimal hyperplane for our classification problem. Here optimality is defined with respect to an objective (usually defined in terms of a cost function), expressed as an *optimization problem*.

The discriminant function, $g(\mathbf{x})$, gives a measurement of the distance of \mathbf{x} from the hyperplane. This can be seen by expressing \mathbf{x} as the summation of a vector \mathbf{x}_p parallel to the hyperplane and one perpendicular to it:

$$\mathbf{x} = \mathbf{x}_p + r \frac{1}{\|\mathbf{w}\|_2} \mathbf{w} \quad (4.31)$$

Then, since $\mathbf{w}^T \mathbf{x}_p = 0$:

$$\begin{aligned} g(\mathbf{x}) &= \mathbf{w}^T \left(\mathbf{x}_p + r \frac{1}{\|\mathbf{w}\|_2} \mathbf{w} \right) \\ &= r \frac{1}{\|\mathbf{w}\|_2} \mathbf{w}^T \mathbf{w} = r \frac{\|\mathbf{w}\|_2^2}{\|\mathbf{w}\|_2} = r \|\mathbf{w}\|_2, \end{aligned} \quad (4.32)$$

and, thus,

$$r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|_2}. \quad (4.33)$$

Using the above, we can see that the distance of the hyperplane from the origin is equal to $w_0 / \|\mathbf{w}\|_2$.

4.3.1 Linearly-separable case

Given a set of vectors \mathbf{x}_i , $i = 1, \dots, N$, each followed by a binary label $l_i \in \{1, -1\}$, we would like to determine a hyperplane expressed by the augmented

weight vector \mathbf{w} that optimally discriminates the two classes. In the case where the two classes are well-separated, a reasonable approach is to look for a weight vector that classifies all given vectors correctly.

A trick that we will use for notation convenience is related to the values of the discriminant function for samples of different classes. Let us re-write the result of applying the discriminant function:

- $g(\mathbf{w}, \mathbf{x}_i) > 0$, then we decide that \mathbf{x}_i belongs to class c_1 ,
- $g(\mathbf{w}, \mathbf{x}_i) < 0$, then we decide that \mathbf{x}_i belongs to class c_2 ,
- $g(\mathbf{w}, \mathbf{x}_i) = 0$, then we say that \mathbf{x}_i belongs to ambiguous region.

From the above, it can be seen that, given an optimal \mathbf{w}^* (i.e. a vector \mathbf{w} that can correctly classify all training samples), we have:

$$f(\mathbf{w}^*, \mathbf{x}_i) = l_i g(\mathbf{w}^*, \mathbf{x}_i) = l_i \mathbf{w}^{*T} \mathbf{x}_i \geq 0, \quad i = 1, \dots, N. \quad (4.34)$$

That is, using the function $f(\cdot)$, all samples are correctly classified by the vector \mathbf{w}^* if $f(\mathbf{w}^*, \mathbf{x}_i) \geq 0$. The question that remains is how to optimize the values of \mathbf{w} , that is, how to determine \mathbf{w}^* .

We now proceed in defining the criterion function to be used in order to optimize \mathbf{w} given \mathbf{x}_i , $i = 1, \dots, N$. Let us assume that \mathbf{w} has been initialized using an initialization process. After applying the function $f(\mathbf{w}, \mathbf{x}_i)$ to all training vectors \mathbf{x}_i , $i = 1, \dots, N$, we identify those that have been erroneously classified (i.e. those for which $f(\mathbf{w}, \mathbf{x}_i) \leq 0$) and use them in order to form the vector set $\mathcal{X}(\mathbf{w})$. This notation means that for different vectors \mathbf{w} , \mathcal{X} will, most probably, be formed by different training vectors. So, a vector \mathbf{w} for which the set \mathcal{X} is empty (i.e. there is no misclassified training sample) will be a solution of the problem.

The overall error of is equal to:

$$\mathcal{J}_p(\mathbf{w}) = \sum_{\mathbf{x}_i \in \mathcal{X}} -f(\mathbf{w}, \mathbf{x}_i) = \sum_{\mathbf{x}_i \in \mathcal{X}} -l_i \mathbf{w}^T \mathbf{x}_i. \quad (4.35)$$

Clearly, if we obtain a vector \mathbf{w}^* for which $\mathcal{J}_p(\mathbf{w}^*) = 0$, \mathbf{w}^* will be a solution to our optimization problem. The criterion in Eq. 4.35 is the so-called *Perceptron criterion function*.

In order to optimize \mathcal{J}_p , we calculate the derivative of it with respect to \mathbf{w} :

$$\nabla \mathcal{J}_p = \sum_{\mathbf{x}_i \in \mathcal{X}} (-l_i \mathbf{x}_i), \quad (4.36)$$

and we update \mathbf{w} following that gradient:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta(t) \nabla \mathcal{J}_p = \mathbf{w}(t) + \eta(t) \sum_{\mathbf{x}_i \in \mathcal{X}} l_i \mathbf{x}_i. \quad (4.37)$$

In the above $\eta(t) > 0$ expresses the rate of change and is called *learning rate*. Its value is a parameter of the learning algorithm and needs to be appropriately chosen. Since the update of the solution is done by using all the training samples, the above-described solution is called batch Perceptron algorithm and is illustrated in Algorithm 5.

Algorithm 5: Batch Perceptron

```

1: Initialize the parameters  $\mathbf{w}$ ,  $\eta(\cdot)$ ,  $t = 0$ 
2: Do  $t \leftarrow t + 1$ 
3:    $\mathcal{X} = \{\}$ 
4:    $i = 0$ 
5:   Do  $i \leftarrow i + 1$ 
6:     if  $f(\mathbf{w}, \mathbf{x}_i) < 0$ , then  $\mathbf{x}_i \rightarrow \mathcal{X}$ 
7:   until  $i = N$ 
8:    $\mathbf{w} \leftarrow \mathbf{w} + \eta(t) \sum_{\mathbf{x}_i \in \mathcal{X}} l_i \mathbf{x}_i$ 
9: until  $\mathcal{X} = \{\}$ 

```

The update of the batch Perceptron algorithm can be expressed as follows: at each iteration, the weight vector is “pushed” towards the vector $\sum_{\mathbf{x}_i \in \mathcal{X}} l_i \mathbf{x}_i$ (calculated on the misclassified samples), by adding a scaled version of it. This guarantees that, after the update, the output of the discrimination function for the vector $\sum_{\mathbf{x}_i \in \mathcal{X}} l_i \mathbf{x}_i$ will be higher.

Alternatively, one can optimize the Perceptron criterion by following an iterative process applied on the training samples sequentially. According to this, a training sample is introduced to the discriminant function and if it is erroneously classified using the current weight vector, the weight vector is updated so that (after the update) the output of the discriminant function for the same sample will be better. All samples are introduced in a sequential manner, each updating the weight vector (when needed). The introduction of all training samples is usually called an *epoch* of the iterative optimization process. Multiple epochs are performed, until all samples are correctly classified. This process is called *single-sample Perceptron* and is illustrated in Algorithm 6. Here we should note that it is usually beneficial to “shuffle” the training samples at each epoch, thus making a random sequence in which the training vectors update the weight vector.

In order to make sure that all training samples will be correctly classified and none of them will belong to the ambiguous region, we can exploit a parameter b , usually called *margin*, as illustrated in Algorithm 7.

Comparing the batch Perceptron and the sample-based Perceptron techniques, the benefit of the batch solution is that the update of the weight vector is smoothed (since it is calculated over all misclassified samples), compared to that in sample-

Algorithm 6: Single-sample Perceptron

-
- 1: Initialize the parameters $\mathbf{w}, \eta(\cdot), \theta, t = 0$
 - 2: **Do** $t \leftarrow t + 1$
 - 3: Select (randomly) a vector \mathbf{x}_i
 - 4: **if** $f(\mathbf{w}, \mathbf{x}_i) < 0$, **then** $\mathbf{w} \leftarrow \mathbf{w} + \eta(t)l_i\mathbf{x}_i$
 - 5: **until** all samples are correctly classified
-

Algorithm 7: Single-sample Perceptron with margin

-
- 1: Initialize the parameters $\mathbf{w}, \eta(\cdot), \theta, t = 0$
 - 2: **Do** $t \leftarrow t + 1$
 - 3: Select (randomly) a vector \mathbf{x}_i
 - 4: **if** $f(\mathbf{w}, \mathbf{x}_i) + b \leq 0$, **then** $\mathbf{w} \leftarrow \mathbf{w} + \eta(t)l_i\mathbf{x}_i$
 - 5: **until** $f(\mathbf{w}, \mathbf{x}_i) + b > 0$ for all i
-

based solution.

Another criterion that can be used to optimize \mathbf{w} , instead of \mathcal{J}_p , is the following:

$$\mathcal{J}_q(\mathbf{w}) = \sum_{\mathbf{x}_i \in \mathcal{X}} (l_i \mathbf{w}^T \mathbf{x}_i)^2, \quad (4.38)$$

where $\mathcal{X}(\mathbf{w})$ is again the set of training samples which have been misclassified by using \mathbf{w} . In order to appropriately weight the contribution of all samples in \mathcal{J}_q and be sure that all training samples will be correctly classified, a variation to 4.38 can be also used:

$$\mathcal{J}_r(\mathbf{w}) = \frac{1}{2} \sum_{\mathbf{x}_i \in \mathcal{X}} \frac{(l_i \mathbf{w}^T \mathbf{x}_i - b)^2}{\|\mathbf{x}_i\|_2^2}, \quad (4.39)$$

where now $\mathcal{X}(\mathbf{w})$ is the set of samples for which $l_i \mathbf{w}^T \mathbf{x}_i \leq b$. The derivative of \mathcal{J}_r with respect to \mathbf{w} is given by:

$$\nabla \mathcal{J}_r = \sum_{\mathbf{x}_i \in \mathcal{X}} \frac{\mathbf{w}^T \mathbf{x}_i - l_i b}{\|\mathbf{x}_i\|_2^2} \mathbf{x}_i, \quad (4.40)$$

and can be used in order to optimize \mathbf{w} as illustrated in Algorithm 8.

A sample-based version of Algorithm 8 is illustrated in Algorithm 9.

Lastly, let us discuss about different ways we can use in order to define the learning rate value η . One way is to arbitrarily select an adequately small value, e.g. $\eta = 0.1$ or $\eta = 0.01$, and use it during the entire training process. This fixed value should be selected based on the properties of the classification problem and the training vectors at hand. A very small value will result in very slow update and long training process, while a large value might lead to fluctuations. Another

Algorithm 8: Batch Relaxation with Margin

```

1: Initialize the parameters  $\mathbf{w}, \eta(\cdot), t = 0$ 
2: Do  $t \leftarrow t + 1$ 
3:    $\mathcal{X} = \{\}$ 
4:    $i = 0$ 
5:   Do  $i \leftarrow i + 1$ 
6:     if  $f(\mathbf{w}, \mathbf{x}_i) + b < 0$ , then  $\mathbf{x}_i \rightarrow \mathcal{X}$ 
7:   until  $i = N$ 
8:    $\mathbf{w} \leftarrow \mathbf{w} - \eta(t) \sum_{\mathbf{x}_i \in \mathcal{X}} \frac{\mathbf{w}^T \mathbf{x}_i - l_i b}{\|\mathbf{x}_i\|_2^2} \mathbf{x}_i$ 
9: until  $\mathcal{X} = \{\}$ 

```

Algorithm 9: Single-sample Relaxation with Margin

```

1: Initialize the parameters  $\mathbf{w}, \eta(\cdot), \theta, t = 0$ 
2: Do  $t \leftarrow t + 1$ 
3:   Select (randomly) a vector  $\mathbf{x}_i$ 
4:   if  $f(\mathbf{w}, \mathbf{x}_i) + b < 0$ , then  $\mathbf{w} \leftarrow \mathbf{w} - \eta(t) \frac{\mathbf{w}^T \mathbf{x}_i - l_i b}{\|\mathbf{x}_i\|_2^2} \mathbf{x}_i$ 
5: until all samples are correctly classified

```

approach is to use a variable $\eta(t)$, with its value decreasing as the number of iterations (or epochs) increase. Again, the rate at which $\eta(t)$ changes is important. If it is too fast, the weight will convergence pre-maturely with non-optimal results.

4.3.2 Non-linearly separable case

In the cases where the two classes are not linearly separable, the approaches discussed in the previous subsection are expected to provide sub-optimal results. In this Section, we will describe two procedures that can be used in order to achieve an acceptable performance in non-linearly separable cases.

Instead of focusing on the misclassified samples, the Minimum Square Error (MSE) solution tries to map all training vectors to some arbitrary target values. That is, it seeks the weight vector \mathbf{w} that can satisfy the following:

$$\mathbf{w}^T \mathbf{x}_i = b_i, \quad i = 1, \dots, N, \quad (4.41)$$

or by using a matrix form:

$$\mathbf{X}^T \mathbf{w} = \mathbf{b}. \quad (4.42)$$

In the above, $\mathbf{X} \in \mathbb{R}^{D \times N}$ is a matrix having as columns the training vectors, i.e. $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ and $\mathbf{b} \in \mathbb{R}^N$ is a vector formed by the *target values*, i.e. $\mathbf{b} = [b_1, \dots, b_N]^T$.

In order to determine the optimal \mathbf{w}^* , we form the following optimization problem:

$$\mathcal{J}_s = \sum_{i=1}^N (\mathbf{w}^T \mathbf{x}_i - b_i)^2 = \|\mathbf{X}^T \mathbf{w} - \mathbf{b}\|_2^2. \quad (4.43)$$

That is, by minimizing \mathcal{J}_s we determine the optimal \mathbf{w} that minimizes the error vector $\mathbf{e} = \mathbf{X}^T \mathbf{w} - \mathbf{b}$. To solve the optimization problem in Eq. 4.43, we calculate the derivative of \mathcal{J}_s with respect to \mathbf{w} and we set it equal to zero, i.e.:

$$\begin{aligned} \nabla \mathcal{J}_s &= \nabla [(\mathbf{X}^T \mathbf{w} - \mathbf{b})^T (\mathbf{X}^T \mathbf{w} - \mathbf{b})] \\ &= \nabla [\mathbf{w}^T \mathbf{X} \mathbf{X}^T \mathbf{w} - 2\mathbf{w}^T \mathbf{X} \mathbf{b} + \mathbf{b}^T \mathbf{b}] \\ &= 2\mathbf{X} \mathbf{X}^T \mathbf{w} - 2\mathbf{X} \mathbf{b}, \end{aligned} \quad (4.44)$$

and

$$\nabla \mathcal{J}_s = 0 \Rightarrow \mathbf{X} \mathbf{X}^T \mathbf{w} = \mathbf{X} \mathbf{b} \Rightarrow \mathbf{w} = (\mathbf{X} \mathbf{X}^T)^{-1} \mathbf{X} \mathbf{b} = \mathbf{X}^\dagger \mathbf{b}, \quad (4.45)$$

where the matrix $\mathbf{X}^\dagger = (\mathbf{X} \mathbf{X}^T)^{-1} \mathbf{X}$ is the pseudo-inverse of \mathbf{X}^T . Here we should note that the matrix $\mathbf{X} \mathbf{X}^T$ should be invertible (i.e. it should be non-singular) in order for \mathbf{X}^\dagger to exist. This is the case when \mathbf{x}_i , $i = 1, \dots, N$ are i.i.d. samples and $D \leq N$. In a different case, the pseudo-inverse is defined as follows:

$$\mathbf{X}^\dagger = \lim_{\epsilon \rightarrow 0} (\mathbf{X} \mathbf{X}^T + \epsilon \mathbf{I})^{-1} \mathbf{X}, \quad (4.46)$$

that is, we employ a *regularized version* of \mathbf{X}^\dagger which changes the elements in the diagonal of the matrix $\mathbf{X} \mathbf{X}^T$ by adding a small scaling version of the identity matrix, thus making the matrix $(\mathbf{X} \mathbf{X}^T + \epsilon \mathbf{I})$ invertible.

In the above, the target values b_i , $i = 1, \dots, N$ can be arbitrarily chosen. However, they are very commonly selected as follows:

$$\begin{aligned} b_i &= 1 \quad , \text{ if } \mathbf{x}_i \text{ belongs to class } c_1 \text{ and} \\ b_i &= -1 \quad , \text{ if } \mathbf{x}_i \text{ belongs to class } c_2, \end{aligned}$$

or, in the case where l_i express binary labels, $b_i = l_i$, $i = 1, \dots, N$.

Another way to optimize \mathcal{J}_s in Eq. 4.43 is by applying an iterative optimization process, as illustrated in Algorithm 10.

4.3.3 Use of Linear Programming Algorithms

All the above algorithms solve simultaneously linear inequalities. Such optimization problems can be solved by formulating suitable linear programming problems, where the linear inequalities are used as constraints.

Algorithm 10: Iterative LMS

```

1: Initialize the parameters  $\mathbf{w}$ ,  $\eta(\cdot)$ ,  $\mathbf{b}$ ,  $\theta$ ,  $t = 0$ 
2: Do  $t \leftarrow t + 1$ 
3:    $\mathbf{w} \leftarrow \mathbf{w} - \eta(t)\mathbf{X}(\mathbf{X}^T\mathbf{w} - \mathbf{b})$ 
4: until  $\eta(t)\mathbf{X}(\mathbf{X}^T\mathbf{w} - \mathbf{b}) < \theta$ 

```

A classic linear programming problem that can be solved by the *simplex method* is the following:

$$z = \mathbf{a}^T \mathbf{u}, \quad (4.47)$$

subject to the constraints:

$$\mathbf{A}\mathbf{u} \geq \beta \quad (4.48)$$

$$\mathbf{u} \geq 0, \quad (4.49)$$

where $\mathbf{a} \in \mathbb{R}^m$ is a *cost vector*, $\mu \in \mathbb{R}^l$ and $\mathbf{A} \in \mathbb{R}^{l \times m}$. Comparing Eqs. 4.48 and 4.42, we can see that the two optimization problems are similar, when using $\mathbf{u} = \mathbf{w}$. However, the constraint in Eq. 4.49 restricts the solution of \mathbf{u} to have only for positive values. To address this issue, we split the vector \mathbf{w} in two vectors:

$$\mathbf{w} = \mathbf{w}^+ - \mathbf{w}^-, \quad (4.50)$$

where \mathbf{w}^+ is a vector having only the positive elements in \mathbf{w} :

$$\mathbf{w}^+ = \frac{1}{2}(|\mathbf{w}| + \mathbf{w}), \quad (4.51)$$

and \mathbf{w}^- is a vector having only the (absolute values of the) negative elements in \mathbf{w} :

$$\mathbf{w}^- = \frac{1}{2}(|\mathbf{w}| - \mathbf{w}). \quad (4.52)$$

In the above $|\cdot|$ expresses the absolute value of a vector, applied in an element-wise manner. Since both \mathbf{w}^+ and \mathbf{w}^- are non-negative, we can form \mathbf{u} with the elements of \mathbf{w}^+ and \mathbf{w}^- in order to satisfy the constraint in Eq. 4.48.

Let us now revisit the linearly separable case. Given a set of N vectors \mathbf{x}_i , $i = 1, \dots, N$, we would like to determine a weight vector \mathbf{w} that satisfies:

$$l_i \mathbf{w}^T \mathbf{x}_i \geq b_i > 0, \quad i = 1, \dots, N. \quad (4.53)$$

Note that here b_i denotes the margin used for sample i (not to be confused with the target values used in the previous subsection). Also, we might choose to use the same margin for all samples, i.e. $b_i = b$, $i = 1, \dots, N$.

In order to express this criterion as a linear programming problem, we introduce a parameter $\tau \geq 0$ and we write:

$$l_i \mathbf{w}^T \mathbf{x}_i + \tau \geq b_i > 0, \quad i = 1, \dots, N. \quad (4.54)$$

Comparing Eqs. 4.53 and 4.54, we can observe that they are equivalent for $\tau = 0$. Thus, we can consider the following problem:

$$\text{Minimize } \tau \text{ such that } l_i \mathbf{w}^T \mathbf{x}_i + \tau \geq b_i \text{ and } \tau \geq 0. \quad (4.55)$$

The above optimization problem can be solved by the simplex method when it is formulated as follows:

$$\text{Minimize: } z = \mathbf{a}^T \mathbf{u}, \quad (4.56)$$

subject to the constraints:

$$\mathbf{A} \mathbf{u} \geq \boldsymbol{\beta} \quad (4.57)$$

$$\mathbf{u} \geq 0 \quad (4.58)$$

where

$$\mathbf{a} = \begin{bmatrix} \mathbf{0}_D \\ \mathbf{0}_D \\ 1 \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} \mathbf{w}^+ \\ \mathbf{w}^- \\ \tau \end{bmatrix}, \quad \boldsymbol{\beta} = \mathbf{b}, \quad \mathbf{A} = \begin{bmatrix} l_1 \mathbf{x}_1^T & -l_1 \mathbf{x}_1^T & 1 \\ l_2 \mathbf{x}_2^T & -l_2 \mathbf{x}_2^T & 1 \\ \vdots & \vdots & \vdots \\ l_N \mathbf{x}_N^T & -l_N \mathbf{x}_N^T & 1 \end{bmatrix} \quad (4.59)$$

where $\mathbf{0}_D \in \mathbb{R}^D$ is a vector of zeros.

In the case where the solution is zero, the samples are linearly separable, and the simplex method gives a good solution. If the solution is positive, there is no separating vector, but it is proved that the samples are not linearly separable.

For the cases where the two classes are not linearly separable, an acceptable solution can be obtained by optimizing the Perceptron criterion in Eq. 4.35, i.e.:

$$\mathcal{J}_p(\mathbf{w}) = \sum_{x_i \in \mathcal{X}} -l_i \mathbf{w}^T \mathbf{x}_i \quad (4.60)$$

where $\mathcal{X}(\mathbf{w})$ is the set of samples misclassified by using \mathbf{w} . In order to avoid the trivial solution $\mathbf{w} = \mathbf{0}$, we optimize a variation of the above criterion:

$$\mathcal{J}'_p(\mathbf{w}) = \sum_{x_i \in \mathcal{X}'} (b_i - l_i \mathbf{w}^T \mathbf{x}_i), \quad (4.61)$$

where now $\mathbf{x}_i \in \mathcal{X}'(\mathbf{w})$ if $l_i \mathbf{w}^T \mathbf{x}_i \leq b_i$.

Since the criterion $\mathcal{J}'_p(\mathbf{w})$ is not a linear function of all samples, we introduce N artificial variables and we optimize the following problem:

$$\text{Minimize: } \alpha = \sum_{i=1}^N \tau_i, \quad (4.62)$$

subject to the constraints:

$$\tau_i \geq b_i - l_i \mathbf{w}^T \mathbf{x}_i \quad (4.63)$$

$$\tau_i \geq 0. \quad (4.64)$$

For any fixed value of \mathbf{w} , the minimum value of z is equal to $\mathcal{J}'_p(\mathbf{w})$. This is due to the constraints leading to $\tau_i = \max(0, b_i - l_i \mathbf{w}^T \mathbf{x}_i)$.

Minimization of $\mathcal{J}'_p(\mathbf{w})$ is equivalent to the following linear programming optimization problem:

$$\text{Minimize: } z = \mathbf{a}^T \mathbf{u}, \quad (4.65)$$

subject to the constraints:

$$\mathbf{A} \mathbf{u} \geq \boldsymbol{\beta} \quad (4.66)$$

$$\mathbf{u} \geq 0 \quad (4.67)$$

where

$$\mathbf{a} = \begin{bmatrix} \mathbf{0}_D \\ \mathbf{0}_D \\ \mathbf{1}_N \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} \mathbf{w}^+ \\ \mathbf{w}^- \\ \boldsymbol{\tau} \end{bmatrix}, \quad \boldsymbol{\beta} = \mathbf{b}, \quad \mathbf{A} = \begin{bmatrix} l_1 \mathbf{x}_1^T & -l_1 \mathbf{x}_1^T & 1 & 0 & \dots & 0 \\ l_2 \mathbf{x}_2^T & -l_2 \mathbf{x}_2^T & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ l_N \mathbf{x}_N^T & -l_N \mathbf{x}_N^T & 0 & 0 & \dots & 1 \end{bmatrix} \quad (4.68)$$

The values $\mathbf{w} = \mathbf{0}$ and $\tau_i = b_i$ is a feasible solution that can be used to start the simplex algorithm.

4.4 Generalized Linear Discriminant Functions

Until now, we have been using linear decision functions of the form:

$$g(\mathbf{x}) = w_0 + \sum_{d=1}^D w_d x_d. \quad (4.69)$$

The above decision function corresponds to a linear decision function (i.e. a hyperplane in \mathbb{R}^D) separating the feature space in two regions. In the following,

we will describe two ways that can be used in order to define non-linear decision functions in \mathbb{R}^D .

The first way is to augment the decision function in Eq. 4.69, by adding more terms involving the products of pairs of the elements of \mathbf{x} :

$$g(\mathbf{x}) = w_0 + \sum_{d=1}^D w_d x_d + \sum_{d=1}^D \sum_{l=1}^D Q_{dl} x_d x_l = w_0 + \mathbf{w}^T \mathbf{x} + \mathbf{x}^T \mathbf{Q} \mathbf{x}, \quad (4.70)$$

where $\mathbf{Q} \in \mathbb{R}^{D \times D}$ is an additional weight matrix. Since $x_d x_l = x_l x_d$, $Q_{dl} = Q_{ld}$, i.e. \mathbf{Q} is a symmetric matrix. Since \mathbf{Q} is a parameter of the classifier, we can define it to be a symmetric and nonsingular matrix. The properties of the classifier are then defined by the properties of the matrix $\tilde{\mathbf{W}} = \frac{1}{(\mathbf{w}^T \mathbf{Q}^{-1} \mathbf{w} - 4w_0)} \mathbf{Q}$. If $\tilde{\mathbf{W}}$ is a positive scaled version of the identity matrix, the decision function corresponds to a hypersphere in \mathbb{R}^D . If $\tilde{\mathbf{W}}$ is positive definite, the decision function corresponds to a hyperellipsoid. If $\tilde{\mathbf{W}}$ is not positive semi-definite (i.e. some of its eigenvalues are positive and others negative), the decision function corresponds to a hyperhyperboloid.

A second way to obtain a nonlinear decision function is to define augmented data representations by including additional terms involving the elements of \mathbf{x} . For example, in the case where the data are represented by one-dimensional values x , then an augmented data representation of the form:

$$\phi = [1 \ x \ x^2] \quad (4.71)$$

can be used in order to define a linear decision function in \mathbb{R}^3 by using:

$$g(\mathbf{y}) = \mathbf{w}^T \phi \quad (4.72)$$

which corresponds to a nonlinear decision function in \mathbb{R} . In the case where the data are represented by two-dimensional vectors $\mathbf{x} = [x_1 \ x_2]^T$, then an augmented data representation of the form:

$$\phi = [1 \ x_1 \ x_2 \ x_1 x_2]^T \quad (4.73)$$

can be used in order to define a linear decision function in \mathbb{R}^4 corresponding to a nonlinear decision function in \mathbb{R}^2 .

4.5 Extension to multi-class classification

Binary decision functions can be combined in order to define multi-class classification schemes in two ways. Let us assume that our classification problem is formed by C classes and the training vectors \mathbf{x}_i , $i = 1, \dots, N$ are followed by

class labels l_i , where $l_i \in \{c_1, \dots, c_K\}$. Such a multi-class classification problem can be split in $\frac{K(K-1)}{2}$ binary classifiers defined on pairs of classes. For example, in a K -class classification problem involving classes c_1 , c_2 and c_3 ($K = 3$), we define the following binary classifiers:

$$\begin{aligned} g_{12}(\mathbf{x}) & \text{ discriminating classes } c_1 \text{ and } c_2, \\ g_{23}(\mathbf{x}) & \text{ discriminating classes } c_2 \text{ and } c_3, \\ g_{13}(\mathbf{x}) & \text{ discriminating classes } c_1 \text{ and } c_3. \end{aligned} \quad (4.74)$$

In order to define the decision function $g_{kl}(\cdot)$, a subset of the original training set is used, which is formed only by the training vectors belonging to classes c_k and c_l . During testing, a new vector \mathbf{x}_* is introduced to all binary classification problems. Each classifier decides in which of the two classes \mathbf{x}_* belongs to and the final classification result corresponds to the class which has been identified as the correct class in most cases. This classification results combination scheme is called *majority voting*. The overall multi-class classification scheme is called *One-versus-One* classification scheme.

A second way to combine binary classifiers for obtaining a multi-class classification scheme is called *One-versus-Rest* (or sometimes *One-versus-All*) classification scheme. According to this, the K -class classification problem is split in K binary problems. For example, in a $K = 3$ -class classification problem involving classes c_1 , c_2 and c_3 , we define the following binary classifiers:

$$\begin{aligned} g_1(\mathbf{x}) & \text{ discriminating classes } c_1 \text{ from } c_2 \text{ and } c_3, \\ g_2(\mathbf{x}) & \text{ discriminating classes } c_2 \text{ from } c_1 \text{ and } c_3, \\ g_3(\mathbf{x}) & \text{ discriminating classes } c_3 \text{ from } c_1 \text{ and } c_2. \end{aligned} \quad (4.75)$$

In order to define the decision function $g_k(\cdot)$, the entire training set is used, where the samples belonging to class c_k form the positive class, while samples belonging to classes c_l , $l \neq k$ are used to form a new class (called negative class). During testing, a new vector \mathbf{x}_* is introduced to all binary classification problems. Each classifier gives a response (e.g. the distance from the decision hyperplane) and \mathbf{x}_* is classified to the class providing the highest response.

The One-versus-Rest classification scheme can be easily implemented by using K MSE binary classifiers (Section 4.3.2). In this case, given training vectors \mathbf{x}_i followed by multi-class labels $l_i \in \{c_1, \dots, c_K\}$, we design N target vectors $\mathbf{t}_i \in \mathbb{R}^K$ such that:

$$\mathbf{t}_{ij} = 1, \text{ if } l_i = j \text{ and } \mathbf{t}_{ij} = -1 \text{ or } \mathbf{t}_{ij} = 0, \text{ if } l_i \neq j. \quad (4.76)$$

It can be shown that the selection of 0 or -1 in the case where $l_i \neq j$ is not important. Then, the weight vectors \mathbf{w}_k , $k = 1, \dots, K$ are used to form the

columns of the weight matrix $\mathbf{W} \in \mathbb{R}^{D \times K}$ and the target vectors \mathbf{t}_i are used to form the columns of the target matrix $\mathbf{T} \in \mathbb{R}^{K \times N}$. \mathbf{W} is obtained by solving for:

$$\mathcal{J}_s = \|\mathbf{W}^T \mathbf{X} - \mathbf{T}\|_F^2, \quad (4.77)$$

where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. The above described training process is based on the idea of data regression to desired targets and, as we will see in Chapter 6, it has direct connections to the training of multi-layer neural networks.

4.6 Questions and Exercises

4.1 Show that maximizing the LDA criterion in Eq. (4.22) is equivalent to minimizing $\mathcal{J}_2(\mathbf{W})$:

$$\mathcal{J}_2(\mathbf{W}) = \frac{\text{Tr}(\mathbf{W}^T \mathbf{S}_w \mathbf{W})}{\text{Tr}(\mathbf{W}^T \mathbf{S}_T \mathbf{W})} \quad (4.78)$$

4.2 Two classes formed by the following samples:

$$c_1 = \begin{bmatrix} -1 & 0 & -0.5 & -1.5 & -2 & 0 & -1 \\ 0 & -1 & -0.5 & -1.5 & 0 & -2 & -1.3 \end{bmatrix} \quad (4.79)$$

$$c_2 = \begin{bmatrix} 1 & 1.3 & 0.7 & 2.5 & 0 \\ 1 & 0.7 & 1.3 & 1 & 1 \end{bmatrix}. \quad (4.80)$$

Classify the following samples using the Nearest Centroid and Nearest Neighbor classifiers:

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_5] = \begin{bmatrix} 0 & 1 & -1 & 0.7 & -0.2 \\ 0 & 1 & 0 & -0.2 & 1.5 \end{bmatrix}. \quad (4.81)$$

4.3 Two classes formed by the following samples:

$$c_1 = \begin{bmatrix} -1 & 0 & -0.5 & -1.5 & -2 & 0 & -1 \\ 0 & -1 & -0.5 & -1.5 & 0 & -2 & -1.3 \end{bmatrix} \quad (4.82)$$

$$c_2 = \begin{bmatrix} 1 & 1.3 & 0.7 & 2.5 & 0 \\ 1 & 0.7 & 1.3 & 1 & 1 \end{bmatrix}. \quad (4.83)$$

Calculate the data projections in the 1-dimensional feature space determined by

applying Fisher Discriminant Analysis.

4.4 Two classes formed by the following samples:

$$c_1 = \begin{bmatrix} -1 & 0 & -0.5 & -1.5 & -2 & 0 & -1 \\ 0 & -1 & -0.5 & -1.5 & 0 & -2 & -1.3 \end{bmatrix} \quad (4.84)$$

$$c_2 = \begin{bmatrix} 1 & 1.3 & 0.7 & 2.5 & 0 \\ 1 & 0.7 & 1.3 & 1 & 1 \end{bmatrix}. \quad (4.85)$$

Train a Perceptron using the above training data. Initialize the weights as $\mathbf{w}(0) = [0.1 \ 0.1 \ 0]^T$ and use a learning rate equal to $\eta = 0.01$. Use as positive class c_1 and as negative class c_2 .

Classify the following samples using the trained model:

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_5] = \begin{bmatrix} 0 & 1 & -1 & 0.7 & -0.2 \\ 0 & 1 & 0 & -0.2 & 1.5 \end{bmatrix}. \quad (4.86)$$

4.5 Show that for a multi-class classifier with K linear discriminant functions of the form:

$$g_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0}, \quad k = 1, \dots, K, \quad (4.87)$$

the decision regions are convex by showing that if $\mathbf{x}_1 \in \mathcal{R}_k$ and $\mathbf{x}_2 \in \mathcal{R}_k$, then $\lambda \mathbf{x}_1 + (1 - \lambda) \mathbf{x}_2 \in \mathcal{R}_k$ for $0 \leq \lambda \leq 1$.

4.6 Discuss why if samples from two categories are distinct (i.e. no feature point is labelled by both categories), there always exists a nonlinear mapping to a higher dimension that leaves the points linearly separable.

4.7 Suppose that for each training point \mathbf{x}_i in class c_1 there exists a (symmetric) point in class c_2 equal to $-\mathbf{x}_i$. Prove that the separating hyperplane of LMS solution passes through the origin.

Chapter 5

Kernel-based Learning

5.1 Support Vector Machine and Kernels

Support Vector Machine is an algorithm that can be used in order to define the parameters of a linear decision function. Let us assume that the training set is formed by vectors $\mathbf{x}_i \in \mathbb{R}^D$, $i = 1, \dots, N$, each followed by a binary label $l_i \in \{-1, 1\}$. Let us also assume that we have defined a data transformation function $\phi(\cdot)$ mapping any vector in \mathbb{R}^D to a new feature space \mathcal{F} , i.e.:

$$\mathbf{x}_i \in \mathbb{R}^D \xrightarrow{\phi(\cdot)} \phi_i \in \mathcal{F}. \quad (5.1)$$

In the following, we make the assumption that $\phi(\mathbf{x}) = \mathbf{x}$, which leads to a linear decision function (and also it means that \mathcal{F} is actually \mathbb{R}^D). As we will see later, we can also define other types of $\phi(\cdot)$ leading to nonlinear decision functions.

Let us now define a linear decision function in \mathcal{F} using the transformed training data ϕ_i , $i = 1, \dots, N$:

$$g(\phi_i) = \mathbf{w}^T \phi_i - b, \quad (5.2)$$

where $\mathbf{w} \in \mathcal{F}$ is a weight vector living in the feature space \mathcal{F} and b is the bias term. Thus, we have:

$$l_i g(\phi_i) \geq 0 \Rightarrow l_i (\mathbf{w}^T \phi_i - b) \geq 0. \quad (5.3)$$

Let us also assume that we would like the decision function to separate the two classes with a margin q (as we have also seen in Chapter 4.3). Here we use the letter q for the margin, since in SVM literature the letter b is widely used to denote the bias term. This means that:

$$\begin{aligned} \mathbf{w}^T \phi_i - b &\geq q, \text{ for } l_i = 1 \text{ and} \\ \mathbf{w}^T \phi_i - b &\leq -q, \text{ for } l_i = -1. \end{aligned} \quad (5.4)$$

That is, q expresses the minimal distance between the decision hyperplane and the closest to it training samples. Using Eq. 4.33, and considering the closest to the hyperplane vector ϕ_m we have:

$$\frac{l_m g(\phi_m)}{\|\mathbf{w}\|_2} = q. \quad (5.5)$$

That is, maximizing the margin q , corresponds to minimization of $\|\mathbf{w}\|_2$. Since scaling the norm of \mathbf{w} does not change the direction of \mathbf{w} , we usually make the convention that $q\|\mathbf{w}\|_2 = 1$, leading to:

$$l_i g(\phi_i) \geq 1, \quad i = 1, \dots, N. \quad (5.6)$$

Based on this idea, the optimization problem of Support Vector Machine (SVM) tries to maximize the margin, while classifying correctly as many training samples as possible. This can be expressed as follows:

$$\mathcal{J}_{SVM} = \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_{i=1}^N \xi_i, \quad (5.7)$$

subject to the constraints:

$$l_i(\mathbf{w}^T \phi_i - b) \geq 1 - \xi_i, \quad i = 1, \dots, N, \quad (5.8)$$

$$\xi_i \geq 0. \quad (5.9)$$

The set of ξ_i (usually called *slack variables*) in Eq. 5.7 expresses an optimization problem balancing between the minimization of the norm of \mathbf{w} (e.g. maximization of the margin) and the minimization of the training error (which is defined in Eq. 5.8). Minimizing the slack variables (each of which is non-negative, as defined in Eq. 5.9), corresponds to the minimization of the training error. In addition, minimization of the first term in Eq. 5.7 corresponds to the maximization of the margin (from Eq. 5.5).

In order to optimize \mathcal{J}_{SVM} subject to the constraints in Eqs. 5.8 and 5.9, we define the Lagrangian:

$$\mathcal{L} = \frac{1}{2} \mathbf{w}^T \mathbf{w} + c \sum_{i=1}^N \xi_i - \sum_{i=1}^N \beta_i \xi_i - \sum_{i=1}^N \alpha_i [l_i(\mathbf{w}^T \phi_i - b) - 1 + \xi_i]. \quad (5.10)$$

By determining the saddle points of \mathcal{L} with respect to \mathbf{w} , b and ξ_i , we obtain:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^N \alpha_i l_i \phi_i, \quad (5.11)$$

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{i=1}^N \alpha_i l_i = 0, \quad (5.12)$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = 0 \Rightarrow c - \alpha_i - \beta_i = 0. \quad (5.13)$$

By substituting (5.11), (5.12) and (5.13) in (5.10), the solution is obtained by solving the following quadratic convex optimization problem:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j l_i l_j \phi_i^T \phi_j + \sum_{i=1}^N \alpha_i, \quad (5.14)$$

subject to the constraint $0 \leq \alpha_i \leq c$, $i = 1, \dots, N$. The optimization problem in Eq. 5.14 can be also written in a matrix form as:

$$\max_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^T (\mathbf{l}^T \circ \mathbf{K}) \boldsymbol{\alpha} + \mathbf{1}^T \boldsymbol{\alpha}, \quad (5.15)$$

where $\mathbf{l} = [l_1, \dots, l_N]^T$ is a vector formed by the labels, \circ denotes the element-wise multiplication operator and $\mathbf{K} \in \mathbb{R}^{N \times N}$ is the so-called *kernel matrix* calculated by:

$$\mathbf{K} = \boldsymbol{\Phi}^T \boldsymbol{\Phi}. \quad (5.16)$$

and $\boldsymbol{\Phi} = [\phi_1, \dots, \phi_N]$. The optimization problem in Eq. 5.15 is a quadratic optimization problem, which (in the case where \mathbf{K} is positive semi-definite) has a global optimum solution. For a function $\phi(\mathbf{x}) = \mathbf{x}$, it can be shown that \mathbf{K} is always positive semi-definite.

The weight matrix \mathbf{w} can be calculated by using Eq. 5.11, i.e.:

$$\mathbf{w} = \sum_{i=1}^N \alpha_i l_i \phi_i. \quad (5.17)$$

That is, the elements of $\boldsymbol{\alpha} \in \mathbb{R}^N$ denote the samples contributing to the calculation of the decision function. Such samples fall in the boundary between the two classes and are called *support vectors*.

The value of b can be calculated by finding a sample for which $\alpha_i > 0$ and compute (derived by Eq. 5.6):

$$b = \mathbf{w}^T \phi_i - l_i. \quad (5.18)$$

Until now, we have considered only the case where $\phi(\mathbf{x}) = \mathbf{x}$. However, as can be observed by Eq. 5.15, the solution of the SVM optimization problem does not depend on the nature of $\phi(\cdot)$. That is, as long as the kernel matrix $\mathbf{K} \in \mathbb{R}^{D \times D}$ is a positive semi-definite matrix, an optimal solution can be found. Since the elements of \mathbf{K} are defined on pairs of samples, i.e. $\mathbf{K}_{ij} = \phi_i^T \phi_j$, we can define any function $\kappa(\cdot, \cdot)$ involving sample pairs, as long as the corresponding \mathbf{K} is positive semi-definite. Such functions are usually called *kernel functions*. Two popular kernel functions are:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|_2^2} \quad (5.19)$$

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^d. \quad (5.20)$$

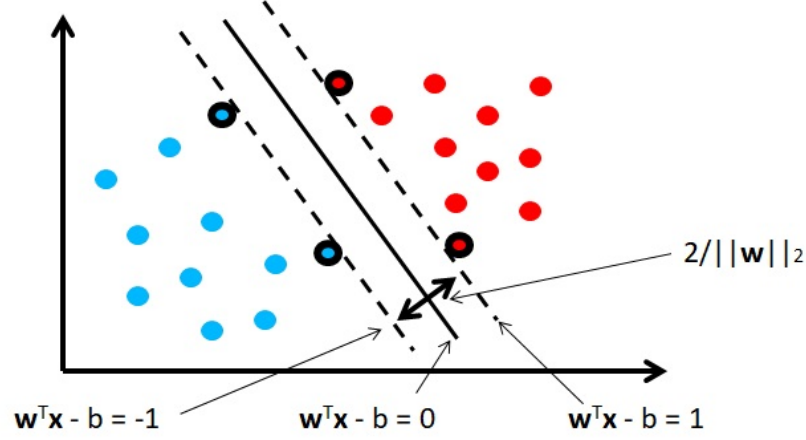


Figure 5.1: Two classes and the corresponding linear decision function obtained by SVM.

Eq. 5.19 is the so-called RBF kernel function with parameter $\gamma > 0$, while Eq. 5.20 is the so-called polynomial kernel function with parameter $d > 0$.

Using a generic kernel function $\kappa(\cdot, \cdot)$ and Eq. 5.11 the decision function becomes:

$$g(\mathbf{x}_*) = \mathbf{w}^T \phi_* - b = \sum_{i=1}^N l_i \alpha_i \phi_i^T \phi_* - b = \boldsymbol{\alpha}^T \mathbf{L} \mathbf{k}_* - b, \quad (5.21)$$

where $\mathbf{L} \in \mathbb{R}^{N \times N}$ is a diagonal matrix with its diagonal elements equal to $L_{ii} = l_i$, b is given by Eq. 5.18 and $\mathbf{k}_* \in \mathbb{R}^N$ is given by:

$$\mathbf{k}_* = [\kappa(\mathbf{x}_1, \mathbf{x}_*), \dots, \kappa(\mathbf{x}_N, \mathbf{x}_*)]^T. \quad (5.22)$$

An example with two classes discriminated by applying SVM is illustrated in Figure 5.1. Support vectors are the samples illustrated with black outline.

5.2 Least-Mean-Square Regression

Let us now consider the problem of mapping a set of samples, represented by the transformed data representations $\phi_i \in \mathcal{F}$, $i = 1, \dots, N$, to a set of vectors $\mathbf{t}_i \in \mathbb{R}^K$ using:

$$\mathbf{W}^T \phi_i = \mathbf{t}_i, \quad i = 1, \dots, N, \quad (5.23)$$

where $\mathbf{W} \in \mathbb{R}^{|\mathcal{F}| \times K}$ is the so-called regression matrix. Here we use the general case having an arbitrary function $\phi(\cdot)$. The case of $\phi(\mathbf{x}) = \mathbf{x}$ and $\mathbf{t}_i \in \mathbb{R}$ defined in Section 4.3.2 is a special case.

In order to define an regression matrix \mathbf{W} satisfying the equalities in Eq. 5.23 in an optimal manner, we minimize the following criterion:

$$\begin{aligned}\mathcal{J}_{LSE} &= \|\mathbf{W}^T \Phi - \mathbf{T}\|_F^2 \\ &= \text{Tr}(\mathbf{W}^T \Phi \Phi^T \mathbf{W} - 2\mathbf{W}^T \Phi \mathbf{T} + \mathbf{T} \mathbf{T}^T),\end{aligned}\quad (5.24)$$

where $\Phi = [\phi_1, \dots, \phi_N]$, $\mathbf{T} = [\mathbf{t}_1, \dots, \mathbf{t}_N]$ and $\text{Tr}(\cdot)$ is the trace operator of a matrix. By setting the derivative of \mathcal{J}_{LSE} with respect to \mathbf{W} equal to zero, we obtain:

$$\nabla \mathcal{J}_{LSE} = 0 \Rightarrow 2\Phi \Phi^T \mathbf{W} = 2\Phi \mathbf{T}^T \quad (5.25)$$

leading to:

$$\mathbf{W} = (\Phi \Phi^T)^{-1} \Phi \mathbf{T}^T = \Phi^\dagger \mathbf{T}^T. \quad (5.26)$$

In the case where the matrix $\Phi \Phi^T$ is singular, a regularized version of Φ^\dagger (as in Eq. 4.46) is used. For the cases where the data representations ϕ_i , $i = 1, \dots, N$ can be directly computed, e.g. for $\phi(\mathbf{x}) = \mathbf{x}$, the solution in Eq. 5.26 can be directly used.

For a general function $\phi(\cdot)$, the data representations ϕ_i , $i = 1, \dots, N$ cannot be directly computed. In this case, we exploit Eqs. 5.11 and 5.16. According to the first one, the solution can be expressed as a linear combination of the training samples, i.e.:

$$\mathbf{W} = \Phi \mathbf{A}, \quad (5.27)$$

where $\mathbf{A} \in \mathbb{R}^{N \times K}$. Thus, the criterion in Eq. 5.24 becomes:

$$\begin{aligned}\mathcal{J}_{LSE} &= \|\mathbf{A}^T \Phi^T \Phi - \mathbf{T}\|_F^2 = \|\mathbf{A}^T \mathbf{K} - \mathbf{T}\|_F^2 \\ &= \text{Tr}(\mathbf{A}^T \mathbf{K} \mathbf{K}^T \mathbf{A} - 2\mathbf{A}^T \mathbf{K} \mathbf{T} + \mathbf{T} \mathbf{T}^T).\end{aligned}\quad (5.28)$$

By setting the derivative of \mathcal{J}_{LSE} with respect to \mathbf{A} equal to zero, we obtain:

$$\nabla \mathcal{J}_{LSE} = 0 \Rightarrow 2\mathbf{K} \mathbf{K}^T \mathbf{A} = 2\mathbf{K} \mathbf{T}^T \quad (5.29)$$

leading to:

$$\mathbf{A} = (\mathbf{K} \mathbf{K}^T)^{-1} \mathbf{K} \mathbf{T}^T = \mathbf{K}^\dagger \mathbf{T}^T. \quad (5.30)$$

In the case where \mathbf{K} is nonsingular, $\mathbf{K}^\dagger = \mathbf{K}^{-1}$.

5.3 Kernel Discriminant Analysis

Revisiting the Linear Discriminant Analysis in Section 4.2, we can define a linear projection from a generic feature space \mathcal{F} defined on the data representations

ϕ_i , $i = 1, \dots, N$. In this case, the within-class scatter matrix is defined as:

$$\mathbf{S}_w = \sum_{k=1}^K \sum_{i, l_i=k} (\phi_i - \boldsymbol{\mu}_k)(\phi_i - \boldsymbol{\mu}_k)^T, \quad (5.31)$$

where the mean vector of class k is defined as:

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i, l_i=k} \phi_i = \frac{1}{N_k} \boldsymbol{\Phi} \mathbf{1}_k, \quad (5.32)$$

where $\mathbf{1}_k \in \mathbb{R}^N$ as a vector having its elements for which $l_i = k$ equal to one and the remaining elements equal to zero.

The total scatter matrix is defined as:

$$\mathbf{S}_b = \sum_{k=1}^K N_k (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T, \quad (5.33)$$

where:

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{i=1}^N \phi_i = \frac{1}{N} \boldsymbol{\Phi} \mathbf{1}, \quad (5.34)$$

where $\mathbf{1} \in \mathbb{R}^N$ as a vector of ones.

Let us now define the matrix $\mathbf{J}_k \in \mathbb{R}^{N \times N}$ as a diagonal matrix having in its diagonal the vector $\mathbf{1}_k$. Then, \mathbf{S}_w becomes:

$$\begin{aligned} \mathbf{S}_w &= \sum_{k=1}^K \left(\boldsymbol{\Phi} \mathbf{J}_k - \frac{1}{N_k} \boldsymbol{\Phi} \mathbf{1}_k \mathbf{1}_k^T \right) \left(\boldsymbol{\Phi} \mathbf{J}_k - \frac{1}{N_k} \boldsymbol{\Phi} \mathbf{1}_k \mathbf{1}_k^T \right)^T \\ &= \sum_{k=1}^K \boldsymbol{\Phi} \left(\mathbf{J}_k - \frac{1}{N_k} \mathbf{1}_k \mathbf{1}_k^T \right) \left(\mathbf{J}_k - \frac{1}{N_k} \mathbf{1}_k \mathbf{1}_k^T \right)^T \boldsymbol{\Phi}^T \\ &= \boldsymbol{\Phi} \left(\sum_{k=1}^K \left(\mathbf{J}_k - \frac{1}{N_k} \mathbf{1}_k \mathbf{1}_k^T \right) \left(\mathbf{J}_k - \frac{1}{N_k} \mathbf{1}_k \mathbf{1}_k^T \right)^T \right) \boldsymbol{\Phi}^T \\ &= \boldsymbol{\Phi} \mathbf{L}_w \boldsymbol{\Phi}^T. \end{aligned} \quad (5.35)$$

In a similar manner, \mathbf{S}_b becomes:

$$\begin{aligned} \mathbf{S}_b &= \sum_{k=1}^K N_k \left(\frac{1}{N_k} \boldsymbol{\Phi} \mathbf{1}_k \mathbf{1}_k^T - \frac{1}{N} \boldsymbol{\Phi} \mathbf{1} \mathbf{1}^T \right) \left(\frac{1}{N_k} \boldsymbol{\Phi} \mathbf{1}_k \mathbf{1}_k^T - \frac{1}{N} \boldsymbol{\Phi} \mathbf{1} \mathbf{1}^T \right)^T \\ &= \boldsymbol{\Phi} \left(\sum_{k=1}^K N_k \left(\frac{1}{N_k} \mathbf{1}_k \mathbf{1}_k^T - \frac{1}{N} \mathbf{1} \mathbf{1}^T \right) \left(\frac{1}{N_k} \mathbf{1}_k \mathbf{1}_k^T - \frac{1}{N} \mathbf{1} \mathbf{1}^T \right)^T \right) \boldsymbol{\Phi}^T \\ &= \boldsymbol{\Phi} \mathbf{L}_b \boldsymbol{\Phi}^T. \end{aligned} \quad (5.36)$$

Thus, the criterion in Eq. 4.22 becomes:

$$\mathcal{J}(\mathbf{W}) = \frac{\text{Tr}(\mathbf{W}^T (\Phi \mathbf{L}_b \Phi^T) \mathbf{W})}{\text{Tr}(\mathbf{W}^T (\Phi \mathbf{L}_w \Phi^T) \mathbf{W})}, \quad (5.37)$$

and by using Eq. 5.27, it can be transformed to the following optimization problem:

$$\begin{aligned} \mathcal{J}(\mathbf{A}) &= \frac{\text{Tr}(\mathbf{A}^T (\mathbf{K} \mathbf{L}_b \mathbf{K}^T) \mathbf{A})}{\text{Tr}(\mathbf{A}^T (\mathbf{K} \mathbf{L}_w \mathbf{K}^T) \mathbf{A})} \\ &= \frac{\text{Tr}(\mathbf{A}^T \mathbf{S}_b^{(A)} \mathbf{A})}{\text{Tr}(\mathbf{A}^T \mathbf{S}_w^{(A)} \mathbf{A})}. \end{aligned} \quad (5.38)$$

The solution of Eq. 5.38 is given by sequentially applying generalized eigen-analysis:

$$\mathbf{S}_b^{(A)} \mathbf{a} = \lambda \mathbf{S}_w^{(A)} \mathbf{a} \quad (5.39)$$

and keeping the eigenvectors corresponding to the maximal $K - 1$ eigenvalues in order to form the columns of \mathbf{A} .

After obtaining \mathbf{A} , the image of ϕ_* in \mathbb{R}^d is given by:

$$\mathbf{y}_* = \mathbf{A}^T \mathbf{k}_*. \quad (5.40)$$

5.4 Questions and Exercises

5.1 Two classes are formed by the following samples:

$$c_1 = \begin{bmatrix} 1 & 2 & 2 \\ 1 & 2 & 0 \end{bmatrix} \quad (5.41)$$

$$c_2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (5.42)$$

Plot these six points and construct by inspection the decision function of the Support Vector Machine classifier. Draw the weight vector, the optimal margin and the support vectors.

5.2 Show why the kernel matrix \mathbf{K} needs to be positive definite in order the SVM classifier to have a solution.

5.3 Show that the linear SVM classifier always finds the optimal hyper-plane in

the linear case.

5.4 Using Eqs. (5.34) and (5.35) imagine how KDA can be extended in order to exploit any type of pair-wise relationships between the data of a class (for \mathbf{S}_w) and between data of different classes (for \mathbf{S}_b). In order to do this, try to visualize the ‘shape’ of each matrix.

Chapter 6

Multilayer Neural Networks

The linear discriminant functions we have seen in the previous chapters, can be described by using a neural network notation. That is, the linear discriminant function of the form:

$$f(\mathbf{w}, \mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0, \quad (6.1)$$

where we have used the augmented version of the weight and sample vectors, \mathbf{w} and \mathbf{x} , can be described by the *two-layer feedforward neural network* illustrated in Figure 6.1. The decision function of such a two-layer neural network has the form of a hyperplane. In the following we will use the augmented vectors notation, as defined in Eq. 4.30. We will describe how one can form neural networks of multiple layers in order to define arbitrary nonlinear decision functions later in this chapter.

There are several ways to describe a neural network formed by multiple layers of neurons, or as they are usually called *multi-layer neural networks*. Here we will follow a biologically inspired approach. Let us consider the biological neuron model illustrated in Figure 6.2. Biological neurons are formed by the main

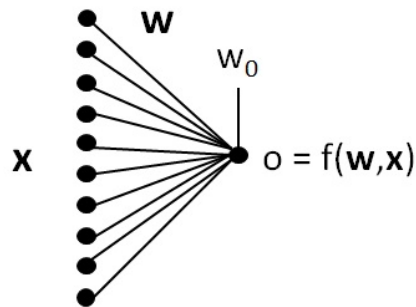


Figure 6.1: A *two-layer neural network implementing a linear discriminant function*.

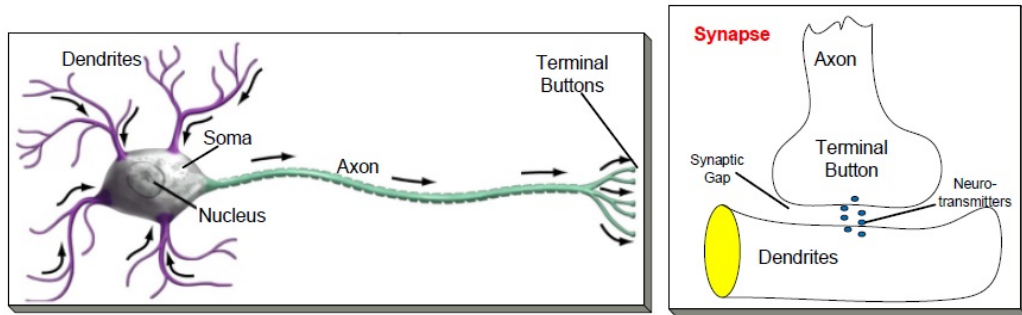


Figure 6.2: A biological neuron (left) with the direction of the signal flow and a synapse (right).

body (called *soma*) which receives electrical signals through connections called *dendrites*. These electrical signals are combined in the soma and are transmitted through the *axon* to be received by other neurons, the dendrites of which are connected to the terminal buttons through the so-called *synapse* (shown in the right side of Figure 6.2). In the synapse we can observe the *synaptic gap* which is a gap between the terminal button of one neuron and the dendrites of another one. If the electrical signal of the first neuron does not exceed a threshold value, then nothing happens. If the electrical signal exceeds a threshold, then the synaptic gap is broken and it is received by the dendrites of the second neuron through the so-called *neuro-transmitters*. We say that in this case the first neuron *fires* (or *is activated*) and passes electrical information to the second neuron. In biological brains each neuron receives information from multiple neurons and transmits information to other neurons. Neurons belonging to the same hierarchical level form a layer of biological neurons. Multiple layers of neurons are *stacked* one after the other, thus, allowing information flow from neurons belonging to lower hierarchical levels to neurons of higher hierarchical levels.

Multi-layer neural networks (or as they are usually called *artificial neural networks* in an analogy to the biological neural networks described above) follow the same information processing flow described above, but with some simplifications that allow us to efficiently compute their responses and train their parameters. The model of an *artificial neuron* is illustrated in Figure 6.3. In a similar manner as the biological neurons, the neuron receives information x_d , $d = 1, \dots, D$ from other neurons and combines it by using the weight w . The combined information o is then introduced to a (usually nonlinear) function $f(\cdot)$ which controls whether the neuron's output o is high enough to activate it. For this reason, $f(\cdot)$ is usually called *activation function*.

Neurons like the one of Figure 6.3 are stacked one on top of the other in order to form a layer. Multiple layers can be stacked one on top of the other in order to

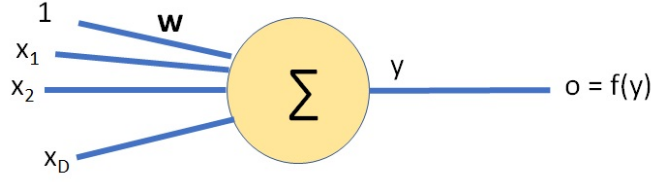


Figure 6.3: An artificial neuron approximating the functionality of a biological neuron.

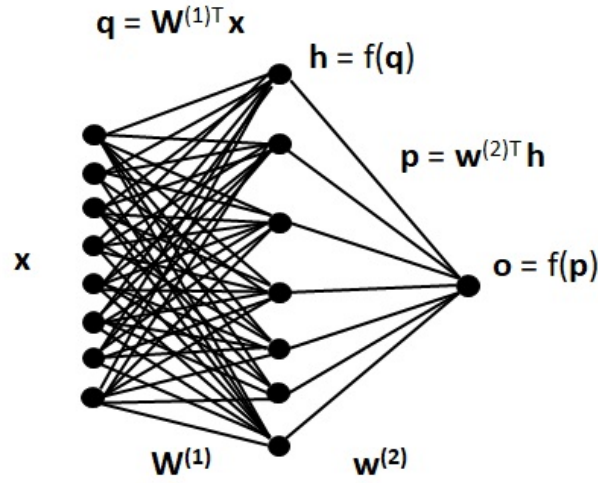


Figure 6.4: A three-layer neural network.

form multi-layer topologies. A three-layer neural network is illustrated in Figure 6.4. In this network, the first layer is formed by D neurons and receives as input a vector \mathbf{x} representing a sample. This layer is usually called *input layer*. The second layer is formed by seven neurons, each receiving information from all neurons in the first layer. This information is weighted by using the values in the weight matrix $\mathbf{W}^{(1)} \in \mathbb{R}^{D \times 7}$. This information is scaled with using the activation function of each neuron and is used to form the output vector of the second layer $\mathbf{h} \in \mathbb{R}^7$. The third layer, then, receives as input the elements of \mathbf{h} and weights them using the weight vector $\mathbf{w}^{(2)}$.

The above can be expressed as follows:

$$h_k = f\left(\mathbf{W}_k^{(1)T} \mathbf{x}\right), \quad k = 1, \dots, K = 7 \quad (6.2)$$

$$\begin{aligned}
o &= f(\mathbf{w}^{(2)T} \mathbf{h}) = f\left(\sum_{k=1}^K w_k^{(2)} h_k\right) \\
&= f\left(\sum_{k=1}^K w_k^{(2)} f(\mathbf{W}_k^{(1)T} \mathbf{x})\right). \tag{6.3}
\end{aligned}$$

Here we assumed that the activation functions of the second and third layers of the neural network are the same. However, as we will see in the following, this is not always the case. In fact, by using different activation functions for the last layer (which is usually called as *output layer*) we can express different properties for the neural network.

A case of particular interest is the one where the activation function of the output layer is linear, i.e. when $o = p$ in Figure 6.4. Then, we have:

$$o = \mathbf{w}^{(2)T} \mathbf{h} = \sum_{k=1}^K w_k^{(2)} h_k = \sum_{k=1}^K w_k^{(2)} f(\mathbf{W}_k^{(1)T} \mathbf{x}). \tag{6.4}$$

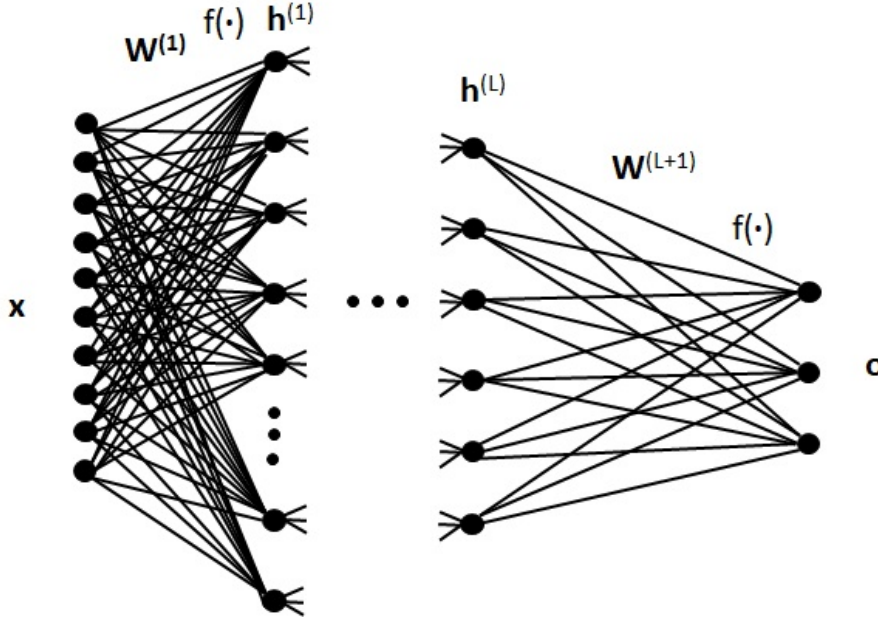
That is, this case corresponds to a linear decision function defined on the data representations \mathbf{h} . It has been also shown that neural networks having the above structure can approximate any continuous function, given appropriate selection of $f(\cdot)$ and number of neurons in the hidden layer.

In the general case, one can stack multiple layers between the input and the output layers, as illustrated in Figure 6.5. As can be seen in this figure, the output of each layer l corresponds to a vector \mathbf{h}_l having dimensions equal to the number of neurons of that layer. These vectors have been obtained by nonlinearly mapping the input vector \mathbf{x} using multiple levels of nonlinear transformations. Thus, each vector \mathbf{h}_l can be regarded as a sample representation of a higher level of abstraction.

There are several algorithms for determining the parameters (i.e. the weight values and the parameter values of the activation functions) of a neural network, most of which based on gradient-based updates. In the following we will describe the most widely used approach, the one based on backpropagating the network's error for a set of training samples and their targets.

6.1 The Backpropagation Algorithm

We now proceed to describe how to *optimize* the parameters of the multi-layer network. These are randomly initialized and updated based on the response of the network for a set of labeled training data. That is, given a set of network's parameters, we introduce a labeled vector \mathbf{x} to the network and we calculate its output \mathbf{o} . We compare this output with a desired output \mathbf{t} (i.e. response of the

Figure 6.5: A multi-layer neural network having L hidden layers.

network that would be optimal when introducing the specific input vector). This comparison is expressed in the form of an error function (or criterion). Then, the weights are adjusted so that, if the same input vector will be introduced after the adjustment, the network's error is reduced.

Let us consider as error function the following criterion:

$$\mathcal{J}(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^K (t_k - o_k)^2 = \frac{1}{2} \|\mathbf{t} - \mathbf{o}\|_2^2, \quad (6.5)$$

where $\mathbf{t} \in \mathbb{R}^K$ is a target vector for a K -class classification problem and $\mathbf{o} \in \mathbb{R}^K$ expresses the network's output. The (error) backpropagation update rule is based on gradient descent:

$$\Delta \mathbf{w} = -\eta \frac{\partial \mathcal{J}}{\partial \mathbf{w}} \quad (6.6)$$

or in an element-wise form:

$$\Delta w_{ij} = -\eta \frac{\partial \mathcal{J}}{\partial w_{ij}}. \quad (6.7)$$

After calculating the gradient, the weights are updated as follows:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta \mathbf{w}. \quad (6.8)$$

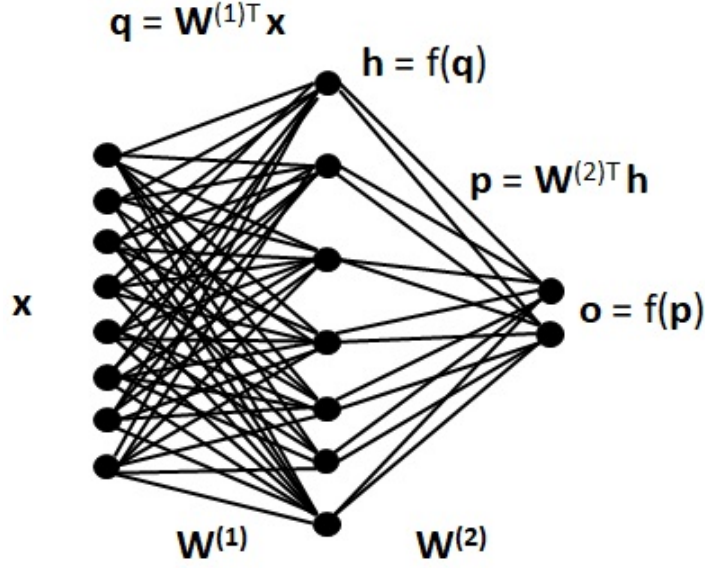


Figure 6.6: A three-layer neural network having two output neurons.

Let us take as an example the three-layer neural network of Figure 6.6, where we assume that both hidden and output layer neurons have an activation function $f(\cdot)$ and we use the criterion in Eq. 6.5 for optimizing the network's weights.

We first focus on the weights $\mathbf{W}^{(2)}$ connecting the hidden layer's neurons with the output neurons. For the weight $W_{jk}^{(2)}$ connecting the hidden layer neuron j and the output layer neuron k , we have:

$$\frac{\partial \mathcal{J}}{\partial W_{jk}^{(2)}} = \frac{\partial \mathcal{J}}{\partial o_k} \frac{\partial o_k}{\partial p_k} \frac{\partial p_k}{\partial W_{jk}^{(2)}}. \quad (6.9)$$

The last term in Eq. 6.9 is given by:

$$\frac{\partial p_k}{\partial W_{jk}^{(2)}} = \frac{\partial}{\partial W_{jk}^{(2)}} \left(\sum_{l=1}^{|\mathbf{h}|} W_{lk}^{(2)} h_l \right) = \frac{\partial}{\partial W_{jk}^{(2)}} (W_{jk}^{(2)} h_j) = h_j. \quad (6.10)$$

The second term in Eq. 6.9 is given by:

$$\frac{\partial o_k}{\partial p_k} = \frac{\partial}{\partial p_k} f(p_k) = f'(p_k). \quad (6.11)$$

This is why the Backpropagation algorithm requires the activation function $f(\cdot)$ to be differentiable. The first term in Eq. 6.9 is given by:

$$\frac{\partial \mathcal{J}}{\partial o_k} = \frac{\partial}{\partial o_k} \left(\frac{1}{2} (t_k - o_k)^2 \right) = o_k - t_k. \quad (6.12)$$

That is, Eq. 6.9 becomes:

$$\frac{\vartheta \mathcal{J}}{\vartheta W_{jk}^{(2)}} = (o_k - t_k) f'(p_k) h_j. \quad (6.13)$$

For the weights $\mathbf{W}^{(1)}$ connecting the input neurons with the hidden layer's neurons we work in a similar manner. For the weight $W_{ij}^{(1)}$ connecting the input neuron i with the hidden layer neuron j , we have:

$$\frac{\vartheta \mathcal{J}}{\vartheta W_{ij}^{(1)}} = \frac{\vartheta \mathcal{J}}{\vartheta h_j} \frac{\vartheta h_j}{\vartheta q_j} \frac{\vartheta q_j}{\vartheta W_{ij}^{(1)}}. \quad (6.14)$$

The last term in Eq. 6.14 is given by:

$$\frac{\vartheta q_j}{\vartheta W_{ij}^{(1)}} = \frac{\vartheta}{\vartheta W_{ij}^{(1)}} \left(\sum_{l=1}^D W_{lj}^{(1)} x_l \right) = \frac{\vartheta}{\vartheta W_{ij}^{(1)}} (W_{ij}^{(1)} x_i) = x_i. \quad (6.15)$$

The second term in Eq. 6.14 is given by:

$$\frac{\vartheta h_j}{\vartheta q_j} = \frac{\vartheta}{\vartheta q_j} f(q_j) = f'(q_j). \quad (6.16)$$

The calculation of the first term in Eq. 6.14 is more complicated. This is because neuron j belongs to the hidden layer and, thus, it affects the criterion \mathcal{J} through all values in the weight $\mathbf{W}^{(2)}$. We have:

$$\frac{\vartheta \mathcal{J}}{\vartheta h_j} = \sum_{k=1}^K \left(\frac{\vartheta \mathcal{J}}{\vartheta o_k} \frac{\vartheta o_k}{\vartheta p_k} \frac{\vartheta p_k}{\vartheta h_j} \right) = \sum_{k=1}^K \left(\frac{\vartheta \mathcal{J}}{\vartheta o_k} \frac{\vartheta o_k}{\vartheta p_k} W_{jk}^{(2)} \right). \quad (6.17)$$

However, we can see in Eqs. 6.11 and 6.12, the first two terms have been already calculated during our calculations for the update of $\mathbf{W}^{(2)}$. Thus, Eq. 6.14 becomes:

$$\frac{\vartheta \mathcal{J}}{\vartheta W_{ij}^{(1)}} = \sum_{k=1}^K \left(\frac{\vartheta \mathcal{J}}{\vartheta o_k} \frac{\vartheta o_k}{\vartheta p_k} W_{jk}^{(2)} \right) f'(q_j) x_i. \quad (6.18)$$

In the general case where the neural network is formed by more than three layers, the weight connecting the i -th neuron in layer l and the j -th neuron in layer $l + 1$ is updated using the gradient:

$$\frac{\vartheta \mathcal{J}}{\vartheta W_{ij}^{(l)}} = \delta_j^l h_i^l, \quad (6.19)$$

where δ_j^l expresses the *sensitivity* of the error with respect to neuron j in layer l and is given by:

$$\delta_j^l = \left(\sum_{q \in \mathcal{L}_{l+1}} \delta_q^{l+1} W_{jq}^{l+1} \right) f'(h_j^{l+1}). \quad (6.20)$$

After updating the gradient the weights are updated by:

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \eta \frac{\partial \mathcal{J}}{\partial W_{ij}^{(l)}}, \quad (6.21)$$

where $\eta > 0$ is the learning rate parameter.

The training process updating the network's weights using randomly selected training samples is illustrated in Algorithm 11. This process is called *stochastic Backpropagation* algorithm.

Algorithm 11: Stochastic Backpropagation

- 1: Initialize the network and its weights $\mathbf{W}^{(l)}$, $l = 1, \dots, L$, $\eta(\cdot)$, θ , $t = 0$
 - 2: **Do** $t \leftarrow t + 1$
 - 3: Select (randomly) a vector \mathbf{x}_i
 - 4: Calculate the network's output for \mathbf{x}_i
 - 5: Update all network's weights $\mathbf{W}^{(l)}$, $l = 1, \dots, L$
 - 6: $W_{ij}^{(l)} \leftarrow W_{ij}^{(l)} - \eta(t) \delta_j^l h_i^l$
 - 7: **until** $\nabla \mathcal{J}(\mathbf{W}) < \theta$
-

In the above process, the network's weights are updated based on a single training sample. In practice we usually update the network's weights based on the errors of it in the entire training set (leading to a batch version of the Backpropagation algorithm) or in a subset of the training set (leading to a mini-batch version of the Backpropagation algorithm). Since the process followed in both cases is the same (only the set of samples used to update the network's weights changes), we will refer to this process as *batch Backpropagation* algorithm and it is illustrated in Algorithm 12.

Updating the network's weights using the Backpropagation algorithm (and in general with any gradient-based algorithm) almost everytime leads to decreased criterion value when it is measured on the training data. However, we should remember that the overall objective is to determine weights that can provide good predictions on unseen data. For this reason, it is common to split the set of available labeled data in two subsets: a large part of it is used for updating the parameters of the network and a smaller set that will be used in order to monitor the *generalization* of the network on data it is not trained on. We call the first set of data as training set and the second one as validation set. Setting the assumption that the validation set can represent the challenges appearing in future unseen data, we select the network's parameter values corresponding to the best performance on the validation set. In Figure 6.7 we illustrate the training error, validation error and test error obtained during the training of a neural network.

Algorithm 12: Batch Backpropagation

```

1: Initialize the network and its weights  $\mathbf{w}$ ,  $\eta(\cdot)$ ,  $\theta$ ,  $t = 0$ ,  $M$ 
2: Do  $t \leftarrow t + 1$ 
3:   Set:  $m = 0$ ,  $\Delta W_{ij}^l$ ,  $l = 1, \dots, L$ 
4:   do  $m = m + 1$ 
5:     Select (randomly) a vector  $\mathbf{x}_i$  (not selected before)
6:     Calculate the network's output for  $\mathbf{x}_i$ 
7:      $\Delta W_{ij}^l \leftarrow W_{ij}^{(l)} + \eta(t) \delta_j^l h_i^l$ 
8:   until  $m = M$ 
9:   Update all network's weights  $\mathbf{W}^{(l)}$ ,  $l = 1, \dots, L$ 
10:   $W_{ij}^{(l)} \leftarrow W_{ij}^{(l)} - \eta(t) \Delta W_{ij}^l$ 
11: until  $\nabla \mathcal{J}(\mathbf{W}) < \theta$ 

```

As can be seen, while keeping the weights of the network corresponding to the best validation error does not necessarily mean that they will correspond to the optimal case in terms of test error, by doing so we are able to avoid cases where the network is over-trained, i.e. when it provides a very low error value on the training data while its error on the test data is very high.

6.2 Activation functions

In the above, we have seen that in order to define non-linear decision function, one needs to define neural networks formed by neurons equipped with activation functions $f(\cdot)$. In addition, we have seen that in order to effectively train the network's weights using the Backpropagation algorithm, these functions need to be differentiable. In the following, we provide a set of widely used activation functions and their derivatives.

Name	Equation	Derivative
Identity	$f(x) = x$	$f'(x) = 1$
Logistic	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
HyperbTan	$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan	$f(x) = \tanh^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
ReLU	$f(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ x, & \text{if } x > 0 \end{cases}$	$f'(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{if } x > 0 \end{cases}$
Softmax	$f_i(\mathbf{x}) = \frac{e^{x_k}}{\sum_{l=1}^K e^{x_l}}$, $k = 1, \dots, K$	$\frac{\partial f_i(\mathbf{x})}{\partial x_j} = f_i(\mathbf{x})(\delta_{ij} - f_j(\mathbf{x}))$

The softmax activation function is commonly used for the neurons of the last layer of a neural network, in order to obtain probability-like response for a given input

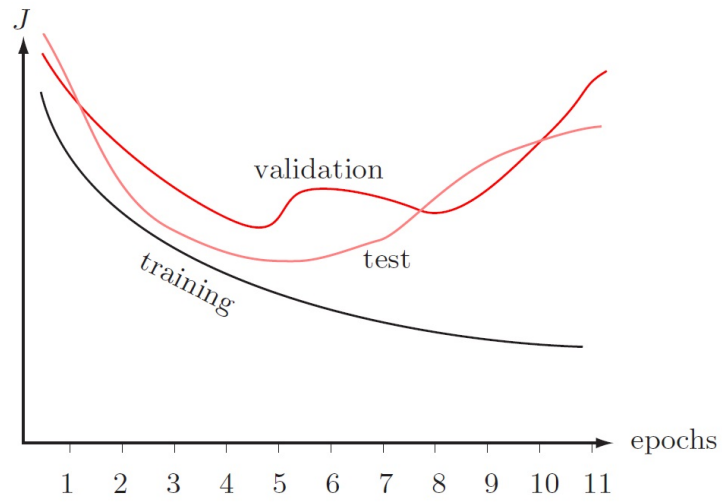
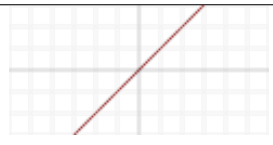
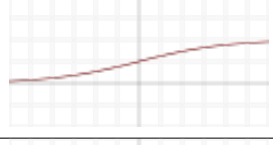
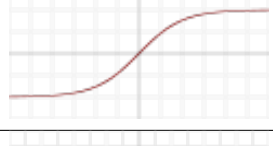
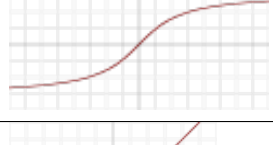



Figure 6.7: Error surfaces of a network's training process.

vector. δ_{ij} is the Kronecker delta taking values $\delta_{ii} = 1$ and $\delta_{ij} = 0$, for $i \neq j$.

Name	Plot
Identity	
Logistic	
ArcTan	
ArcTan	
ReLU	

6.3 Practical matters

When using a multi-layer neural network to solve a classification problem, it is important to make the correct decisions regarding the adopted architecture of the network, its nonlinearities (defined through the activation functions of the network's neurons), appropriate pre-processing steps applied on the input data, weights initialization and learning rates for updating them.

While by using a large number of neurons multi-layer neural networks can approximate any decision function, it is usually preferred to employ smaller networks that are able to achieve a good training error. Such networks are expected to provide better performance on unseen data, due to the fact that the number of their parameters is smaller (and comparable with the number of training data). Moreover, they require shorter training processes and classification is faster due to the smaller number of parameters. For these reasons, whenever there are some information indicating the use of a specific type of activation functions, such information is valuable.

Data pre-processing is also very important. A common pre-processing step in training neural networks is to standardize the training data (as in Section 2.1). Such a pre-processing step helps in avoiding the network's weight values to be trapped in regions of the activation function making their output saturated. Depending on the problem at hand other data pre-processing steps might also be beneficial in weights updating and processing speed. Here we should note that, the parameters of the pre-processing step should be kept and should be used for pre-processing new (test) samples accordingly.

Data pre-processing and activation function choices are also connected with the network's weights initialization strategy followed. When initializing weights in a given layer, we choose weights randomly from a single distribution to help ensure uniform learning. Because data standardization gives positive and negative values equally, on average, we want positive and negative weights as well. Thus, we choose weights from a uniform distribution, where $-\gamma < w_{ij} < +\gamma$. If the value of γ is selected to be too small, depending on the chosen activation function, the response of that layer might be too small making the learning process of the entire network difficult. If the value of γ is selected to be too big, then the neurons in that layer might saturate even before the actual training is done.

For multi-layer neural networks consisting of many layers, the selection of the optimal learning rate η is not trivial. In principle, as long as a network is trained until convergence, a small enough learning rate will mostly affect the speed of training. However, since training of deep neural networks (i.e. networks formed by a large number of layers) is computationally demanding, training is not allowed to continue until convergence. In such cases, a small learning rate might affect the quality of the final network. Selection strategies based on the second derivative of

the error with respect to the network's weights have been proposed, however, in most cases an arbitrary learning rate value is used which decreases as a function of the number of update iterations.

When training neural networks using a sample based or mini-batch based process, the direction of the gradients for the successive updates might be inconsistent. This might result in oscillations and slow convergence. Learning with *momentum* allows a smoother update of the network's parameters:

$$W_{ij}^{(l)}(t+1) = W_{ij}^{(l)}(t) - \eta \left(\frac{\partial \mathcal{J}}{\partial W_{ij}^{(l)}(t)} + \alpha \frac{\partial \mathcal{J}}{\partial W_{ij}^{(l)}(t-1)} \right). \quad (6.22)$$

The values of α should be less than 1.0. A typical choice is $\alpha = 0.9$. It can be seen that, in this way, the new gradient corresponding to the error of the network for the samples at hand is used for refining the direction of the update, allowing for a smoother (and usually faster) convergence. Thus, momentum can be seen as a process averaging the stochastic variations in the weight updates during stochastic (sample-based) learning.

Algorithm 13: Stochastic Backpropagation with momentum

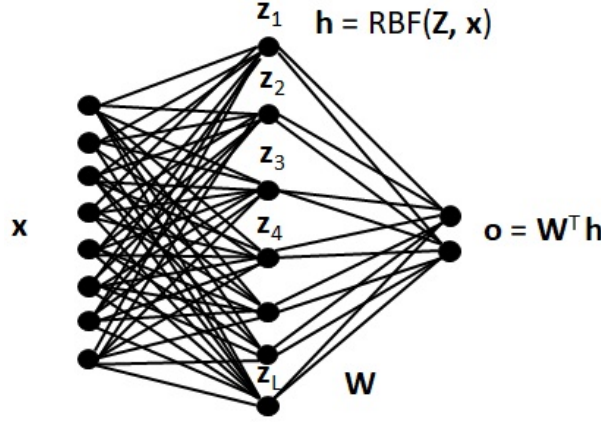
- 1: Initialize the network and its weights \mathbf{W} , $\eta(\cdot)$, θ , α , $t = 0$
 - 2: Set $h_i^l(0) = 0$ and $\delta_j^l(0) = 0$
 - 3: **Do** $t \leftarrow t + 1$
 - 4: Select (randomly) a vector \mathbf{x}_i
 - 5: Calculate the network's output for \mathbf{x}_i
 - 6: Calculate $h_i^l(t)$ and $\delta_j^l(t)$
 - 7: Update all network's weights $\mathbf{W}^{(l)}$, $l = 1, \dots, L$
 - 8: $W_{ij}^{(l)} \leftarrow W_{ij}^{(l)} - \eta(t) (\delta_j^l(t) h_i^l(t) + \alpha \delta_j^l(t-1) h_i^l(t-1))$
 - 9: **until** $\nabla \mathcal{J}(\mathbf{W}) < \theta$
-

6.4 Radial Basis Function network

Radial Basis Function (RBF) network is a neural network formed by three layers: the input layer which receives an input vector $\mathbf{x} \in \mathbb{R}^D$, a hidden layer formed by a set of L data prototypes $\mathbf{z}_l \in \mathbb{R}^D$ and an output layer defining the discriminant function. A RBF network is illustrated in Figure 6.8.

The hidden layer is equipped with the Radial Basis Function defined on an input pattern \mathbf{x} and a prototype \mathbf{z}_l as follows:

$$h_l = RBF(\mathbf{z}_l, \mathbf{x}) = \exp \left(-\frac{\|\mathbf{x} - \mathbf{z}_l\|_2^2}{\sigma^2} \right), \quad l = 1, \dots, L. \quad (6.23)$$

Figure 6.8: *Radial Basis Function network.*

The responses of all RBF neurons in the hidden layer of the network are used to form the hidden layer's output vector:

$$\mathbf{h} = [h_1, \dots, h_L]^T. \quad (6.24)$$

The network's output neurons are equipped with a linear activation function, i.e.:

$$\mathbf{o} = \mathbf{W}^T \mathbf{h}. \quad (6.25)$$

Since the vectors \mathbf{z}_l , $l = 1, \dots, L$ represent data prototypes, they are usually calculated by applying a clustering algorithm on the training data \mathbf{x}_i , $i = 1, \dots, N$, like K -Means (Section 2.4).

Given a set of prototype vectors \mathbf{z}_l , $l = 1, \dots, L$, RBF networks optimize the Mean Square Error between the network's outputs \mathbf{o}_i and targets \mathbf{t}_i on a given set of training vectors \mathbf{x}_i , $i = 1, \dots, N$:

$$\mathcal{J}_{MSE} = \sum_{i=1}^N \|\mathbf{t}_i - \mathbf{o}_i\|_2^2 = \sum_{i=1}^N \|\mathbf{t}_i - \mathbf{W}^T \mathbf{h}_i\|_2^2 = \|\mathbf{T} - \mathbf{W}^T \mathbf{H}\|_F^2, \quad (6.26)$$

where $\mathbf{H} = [\mathbf{h}_1, \dots, \mathbf{h}_N]$ and $\mathbf{T} = [\mathbf{t}_1, \dots, \mathbf{t}_N]$. The network's output weights are given by (Section 4.3.2):

$$\mathbf{W} = \mathbf{H}^\dagger \mathbf{T}^T \quad (6.27)$$

6.5 Auto-Encoder networks

An Auto-Encoder network (or simply *Auto-Encoder* - *AE*) is a three layer neural network trained in an unsupervised manner. An Auto-Encoder is illustrated in

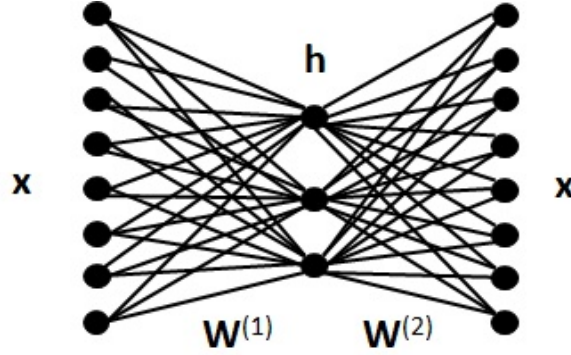
Figure 6.9: *Auto-Encoder network.*

Figure 6.9. The main idea of the AE is to reconstruct the input vector through passing from a *bottleneck* h . In this way, the information encoded in the training data \mathbf{x}_i in \mathbb{R}^D is compressed in a lower-dimensional feature space \mathbb{R}^d , $d < D$. A variant of the AE networks tries to reconstruct a noisy variant of the input $\tilde{\mathbf{x}}$, which is created by randomly setting equal to zero a percentage of its elements. This type of AE is called *Denoising Auto-Encoders*. The weights of the network are trained by applying the Backpropagation algorithm.

Auto-Encoders have been used in order to form deep neural network architectures in the cases where the number of training data is small. This is done by training K AEs in a hierarchical manner. An example can be seen in Figure 6.10. In this case, given a set of training vectors \mathbf{x}_i an AE defining a new data representation $\mathbf{h}_{1,i}$ is trained. Then, the new data representations $\mathbf{h}_{1,i}$ are used in order to train a second AE defining a second representation $\mathbf{h}_{2,i}$. After defining the two data representations, the weights of the first three layers in Figure 6.10 are used to initialize the weights of a four-layer network formed by adding on top of the third layer (i.e. the layer \mathbf{h}_2 in Figure 6.10) an output layer.

6.6 Self-Organizing Map

Self-Organizing Map (SOM) is another type of neural network trained in an unsupervised way. It is used in order to define a set of K prototype vectors. In the training phase, training vectors \mathbf{x}_i , $i = 1, \dots, N$ are used to produce a topographic map (lattice) of the input data, in which the spatial locations of the resulting prototypes in the lattice are indicative of intrinsic statistical features of \mathbf{x}_i . The training procedure for constructing the SOM is based on three procedures:

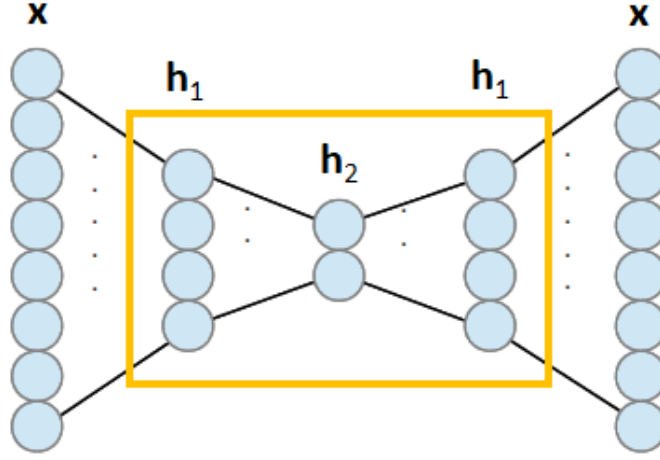


Figure 6.10: Two Auto-Encoders trained in a hierarchical manner to form a four layer network.

Competition

For each of the training vectors $\mathbf{x}_i \in \mathbb{R}^D$, its Euclidean distance from every SOM weight, $\mathbf{w}_j \in \mathbb{R}^D$, $j = 1, \dots, K$ is calculated. The winning neuron is the one that gives the smallest distance:

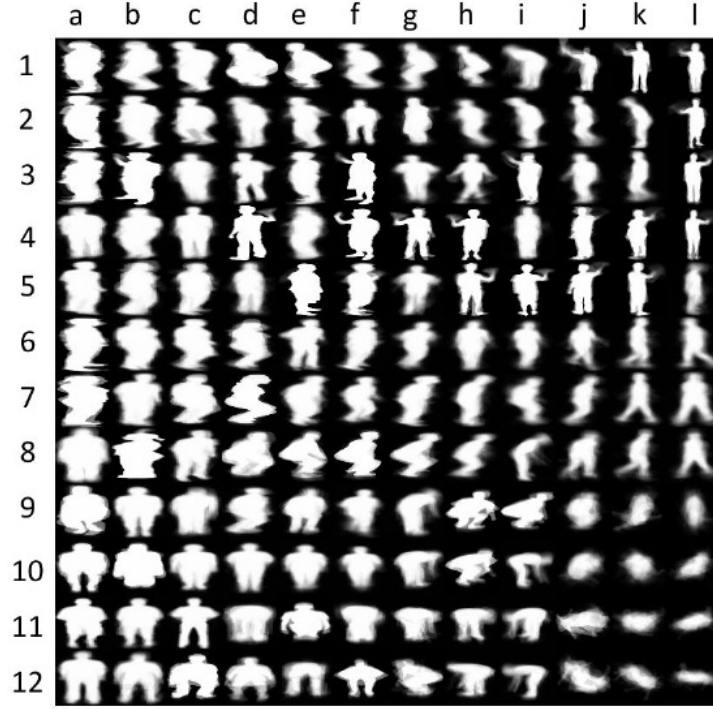
$$j^* = \arg \min_j \|\mathbf{x}_i - \mathbf{w}_j\|_2. \quad (6.28)$$

Cooperation

The winning neuron j^* indicates the center of a topological neighborhood h_{j^*} . Neurons are excited depending on their lateral distance, r_{j^*} , from this neuron. A typical choice of h_{j^*} is the Gaussian function:

$$h_{j^*k}(n) = \exp\left(-\frac{r_{j^*k}^2}{2\sigma^2(n)}\right), \quad (6.29)$$

where k corresponds to the neuron at hand, n is the iteration of the algorithm, r_{j^*k} is the Euclidean distance between neurons j^* and k in the lattice space and σ is the “effective width” of the topological neighborhood. σ is a function of n : $\sigma(n) = \sigma_0 \exp(-\frac{n}{N_0})$, where N_0 is the total number of training iterations. The value of $\sigma(0)$ needs to be set in advance, e.g. by using $\sigma(0) = \frac{l_w + l_h}{4}$, l_w and l_h are the lattice width and height, respectively.

Figure 6.11: A 12×12 SOM lattice.

Adaptation

At this step, each neuron is adapted with respect to its lateral distance from the winning neuron as follows:

$$\mathbf{w}_k(n+1) = \mathbf{w}_k(n) + \eta(n)h_{j^*k}(n)(\mathbf{x}_i - \mathbf{w}_k(n)), \quad (6.30)$$

where $\eta(n)$ is the learning-rate parameter: $\eta(n) = \eta(0) \exp(-\frac{n}{N_0})$. $\eta(0) = 0.1$ in our experiments.

An example SOM lattice formed by $K = 12 \times 12$ neurons trained using vectorized human body silhouettes is illustrated in Figure 6.11. As can be seen, neighboring neurons correspond to similar human body poses.

6.7 Convolutional Neural Networks

All techniques described above take as input data representations having a vectorial form. That is, in order to use these techniques for practical applications, one needs to define a way of representing the samples (which might be images, videos, text, etc.) in a vector space \mathbb{R}^D . Such, human-designed, data representations are usually called as *hand-crafted data representations*. A special type of

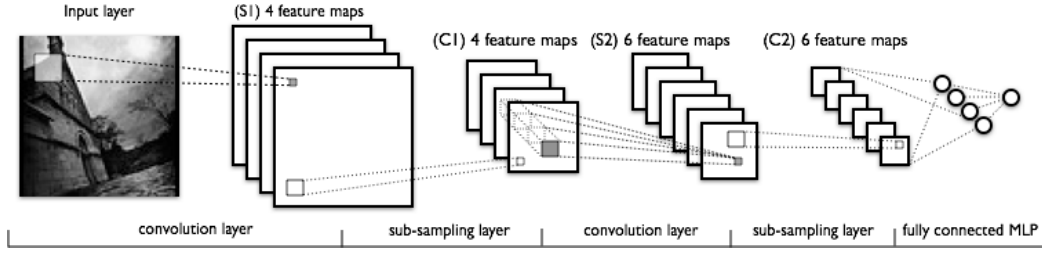


Figure 6.12: *Convolutional Neural Network applied directly on an image.*

neural network that can receive as input *raw data*, i.e. directly exploit the data, is called Convolutional Neural Network (CNN). While CNNs can be used in many applications involving any type of data (e.g. images, videos, time-series, vectors), they have been motivated by applications involving images and have been widely used in image-based machine learning applications. A CNN applied on an image is illustrated in Figure 6.12.

As can be seen in Figure 6.12 a CNN is formed by three types of layers:

- Convolutional layers
- Sub-sampling layers and
- Fully connected layers (MLP)

A convolution layer is illustrated in Figure 6.13. Let us assume that at layer $m - 1$ the output of the CNN layer is a set of N_{m-1} images (or as they are commonly called *feature maps*). In the case where layer m is the first layer, the feature maps correspond to the input image. In the example of Figure 6.13, $N_{m-1} = 4$. Then, the convolutional layer m is equipped with N_m multi-dimensional filters. Each multi-dimensional filter is formed by a set of N_{m-1} filters, each of which is used to *convolve* the corresponding feature map in layer $m - 1$ and produce a feature map at layer m . Since layer m has N_m multi-dimensional filters, the number of feature maps in that layer will be equal to N_m . After the calculation of the feature maps of layer m , a (usually nonlinear) activation function is applied on each of their elements. That is, in the case of the convolution in Figure 6.13, the value of w^1 in map I_m placed at location x and y is given by placing the $N_{m-1} = 4$ filters at the places denoted by the various colors in the N_{m-1} maps of layer $m - 1$ and calculating:

$$I_m(y, x) = f \left(\sum_{d=1}^D \sum_{i=0}^1 \sum_{j=0}^1 w_{ij}^{0d} I_{m-1}^d(y + i, x + j) \right) \quad (6.31)$$

where $f(\cdot)$ is the chosen activation function of that layer.

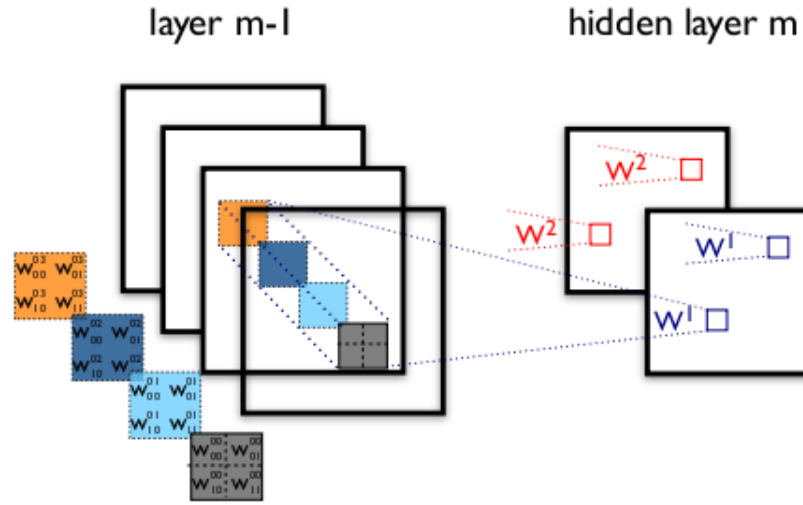


Figure 6.13: A convolutional layer.

A sub-sampling layer takes as input a feature map and reduces its dimensions by following a fusion scheme. The most widely adopted fusion scheme in image analysis is *max-pooling*. Max-pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value. It is useful in vision for two reasons:

- By eliminating non-maximal values, it reduces computation for upper layers.
- It provides a form of translation invariance.

Another widely used fusion scheme is the *average pooling*, which in essence corresponds to resizing the input image using linear interpolation.

After the application of several convolutional and sub-sampling layers, the size of each feature map becomes equal to 1×1 . This layer now becomes a vector-based neural network layer which can be followed by a number of vector-based (or multi-layer Perceptron - MLP) layers like those described in the beginning of this Chapter.

Training of the weights (i.e. the MLP weights and the filters of the CNN layers) is performed by following Backpropagation, in a similar manner as in multi-layer neural networks. For the update of the convolutional layer weights, special care is taken in order to keep the *spatial connectivity* property of these layers.

6.8 Questions and Exercises

6.1 Show that if an activation function of a three-layer neural network is linear, then this network is equivalent to a two-layer network. Based on that, explain why a three-layer neural network with linear activation functions cannot define nonlinear decision functions.

6.2 Consider a three-layer neural network with D input neurons, L neurons in the hidden layer and K output neurons.

1. How many weights are there in the network?
2. Compute the number of operations that need to be performed in order to classify a vector \mathbf{x} .

.

6.3 Use Eq. (6.2) to show why the input-to-hidden layer weights must be different from each other (e.g. initialized random) or else learning cannot proceed well.

6.4 Write the algorithm for optimizing the parameters of a three-layer neural network trained using $\mathbf{x}_i, i = 1, \dots, N$ and the corresponding labels $l_i \in \{c_1, \dots, c_K\}$.

6.5 Write the algorithm for optimizing the parameters of an RBF network trained using $\mathbf{x}_i, i = 1, \dots, N$ and the corresponding labels $l_i \in \{c_1, \dots, c_K\}$.

6.6 Show that an RBF network trained on $\mathbf{x}_i, i = 1, \dots, N$ can approximate any continuous function (which is defined by N sampled values $t_i, i = 1, \dots, N$). How can this be extended to any three-layer feedforward neural network?

Chapter 7

Validation

Most of the above-described techniques require the selection of a number of parameter values, before training the model. For example, Support Vector Machine in Section 5.1 requires the selection of a good value for the parameter c and K -Means in Section 2.4 requires the selection of the number of clusters K . In the cases where information related to the problem at hand can indicate a good value, e.g. when it is known beforehand that the number of groups in a clustering problem is equal to $K = 5$, one can proceed directly in solving the corresponding optimization problem using that value. However, in most cases, such information is not available and a good value for these parameters (usually called *hyper-parameters* due to that they need to be optimized at a higher level than the parameters optimization process) needs to be determined.

Let us now assume that a set of possible values of such hyper-parameters is known. For example, in the case of SVM-based classification, we know that the parameter c can take values in the set of 10^r , $r \in \{-6, \dots, 6\}$. Testing the optimization criterion value on a set of training data $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]$ is not a good practice, since the value of c providing the best criterion value on the training data does not necessarily leads to good *generalization* on unseen data.

The general solution in this problem is to evaluate the performance of the models (each using a different hyper-parameter value) on a set of data that have not been used to optimize the parameters of the models and select the model corresponding to the best performance. Such a set is usually called *validation set*. In this way, we have a way to measure ability of the model to generalize well on new data. This is, of course, by assuming that the properties of the validation set are similar with those of other new data (or as they are usually called *test data*). Such a process is useful in finding errors in implementation, comparing various models and learning algorithms and detecting over (or under) trained models.

We can define two validation procedures:

- Hold-out validation and

- K -fold Cross Validation

In the first case, the training set \mathbf{X} is randomly split in two sets, i.e. $\mathbf{X} = [\mathbf{X}_{train}|\mathbf{X}_{valid}]$. The first set \mathbf{X}_{train} formed by N_{train} samples is used in order to train the models, while the second set \mathbf{X}_{valid} formed by N_{valid} samples is used to measure their performance. Obviously, $N_{train} + N_{valid} = N$. Since it is important to train the models using as much samples as possible, we select $N_{train} > N_{valid}$. For example, we can select $N_{train} = 0.7N$ and $N_{valid} = 0.3N$. As can be seen, in this case the models are not trained on the entire dataset, which might lead to lower performance compared to the case where the entire dataset is used for training. Also, the various models have been tested only in the cases defined on the validation data. One could argue that a different random partition of the data might lead to a different optimal hyper-parameter value.

In K -fold Cross Validation, the training data are randomly split in K sets, i.e. $\mathbf{X} = [\mathbf{X}_1|\mathbf{X}_2|\dots|\mathbf{X}_K]$. Each model (corresponding to a different hyper-parameter value choice) is trained N times using all but one set. The set not used for training is used for evaluation. This process is applied K times (or folds), and at each fold a different set is used for evaluation, and the average performance of each model is calculated. The model corresponding to the best average performance is then chosen. A special case of K -fold Cross Validation is the case $K = N$, i.e. when the validation set corresponds to one sample only. This is called *Leave-One-Out Cross Validation* process.