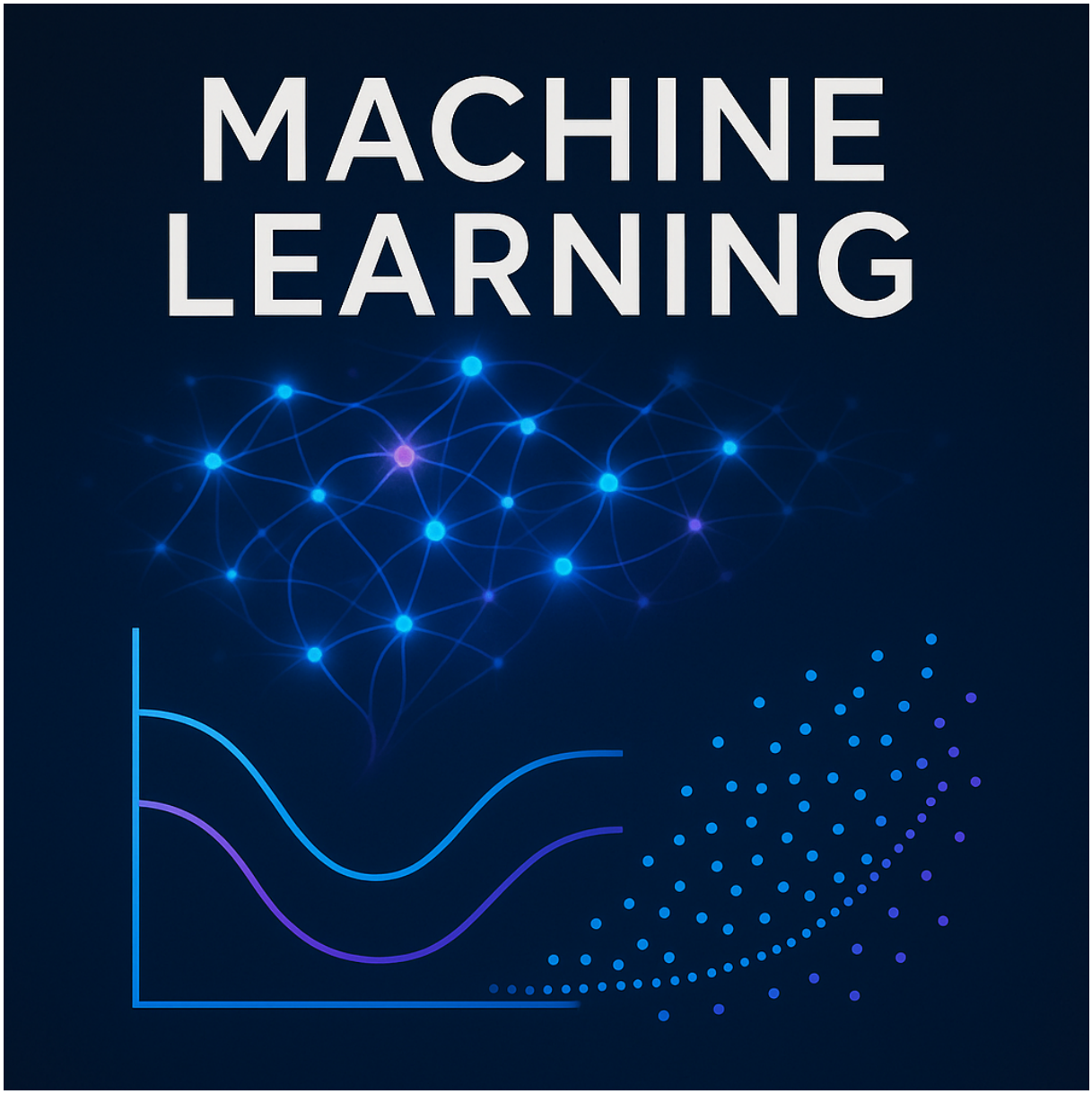
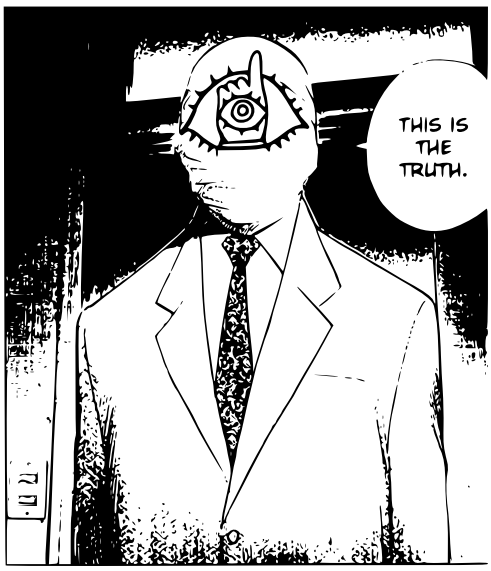
Marco Casu



SAPIENZA
UNIVERSITÀ DI ROMA

Faculty of Information Engineering, Computer Science and Statistics
Department of Computer, Control and Management Engineering
Master's degree in Artificial Intelligence and Robotics

This document summarizes and presents the topics for the Machine Learning course for the Master's degree in Artificial Intelligence and Robotics at Sapienza University of Rome. The document is free for any use. If the reader notices any typos, they are kindly requested to report them to the author.



CONTENTS

CHAPTER

1

INTRODUCTION

1.1 Basic Definition of a ML Problem

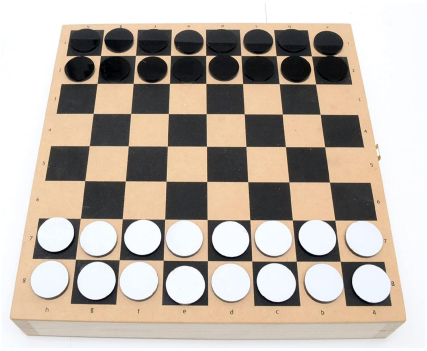
In this chapter we will introduce the basics of what is a machine learning problem, giving a mathematical definition. informally, with machine learning we define the use of knowledge (data) to improve the performance of a given program, using past experiences.

Generally, we use machine learning to solve problems with no deterministic solutions, trying to find an approximate one (such as recognizing what animal is represented in a given photo).

Usually, a machine learning problem consists in three main component:

- T : the given task
- P : a performance metric
- E : the past experiences (the data)

Let's see an example, we want to model a program capable of playing *Checkers*.



The task T is to play the game, the performance metric P measure the ratio of win in a tournament, and the experiences is given by the past match. We can create this matches by letting the computer play against himself, or by making it play against a human. We can consider two types of target functions that models the behavior of the model:

- $ChooseMove : Board \to Move$
- $V : Board \to \mathbb{R}$

Board is the set of the possible board configurations, move is the set of feasible moves. The image of the function V should represents the validity of a board in the following sense:

- if $b \in Boards$ is a configuration that represents the win of the player, then $V(b) = 1$
- else if $b \in Boards$ is a configuration that represents the win of the opponent, then $V(b) = -1$
- else if $b \in Boards$ is a configuration that represents a draw, then $V(b) = 0$
- else, $V(b) = V(b')$ where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game.

The function V can be used to predict the next move by considering all possible boards that can be obtained from the feasible moves. We focus on the function V, this should be an optimal model, but is not computable, because we cannot tell if a play is "optimal" (this is the goal of the model), so we want to consider a function that approximate the behavior of V.

We consider a function $\hat{V} : Board \to \mathbb{R}$ defined as follows:

$$\hat{V}(b) = \sum_{i=0}^{6} w_i f_i(b) \tag{1.1}$$

where $\mathbf{w} = (w_i)$ is real coefficients, and the functions f_i represents some features of the given board:

- $f_1(b)$: number of black pieces on b
- $f_2(b)$: number of red pieces on b
- ecc...

We don't know if some features are useful to predict which move is optimal, is important that this features model the knowledge of the *domain* (in this case, the game of Checkers).

with *learning the function* $\hat{V}$, we intend to finds the coefficients $\mathbf{w}$ which make the function $\hat{V}$ more similar to V as possible. There are various method to find these coefficients.

Let's introduce some notation:

- with V we define the **target function** (always unknown and uncomputable).
- with $\hat{V}$ we define the **learned function**, the approximation of V that we want to find.
- with $V_{train}(b)$ we define the value of V obtained at b, where b is a part of a data set that is given. We will use the values of V_{train} on the data set to synthesize $\hat{V}$.
- with D we define the given data set:

$$D = \bigcup_{i=1}^{n} \{(b_i, V_{train}(b_i))\} \tag{1.2}$$

  n is the number of the available data.

The iterative method given in the Algorithm 1 is an informal example of how we can find the values for $\mathbf{w}$. In this case c is a small constant, usually in $(0, 1]$, to moderate the rate of learning.

The function $error(b)$ is computable only on the sample b given in the dataset D, the goal of the method is to converge to a local minimum for the function

$$\sum_{b_i \in D} error(b_i). \tag{1.3}$$

Once we synthesize $\hat{V}$ with this method, we can make the model play against human and track the result to use it as an additional dataset. The diagram in image 1.1 represents the process of designing an artificial intelligence agent.

Algorithm 1 LMS weight update rule

Require: D, V_{train}, k
  initialize **w** with small random values
  for k times **do**
    select a sample b from D
    $error(b) = V_{train}(b) - \hat{V}(b)$
    for each feature i **do**
      $w_i \leftarrow w_i + c \cdot f_i(b) \cdot error(b)$
    end for
  end for

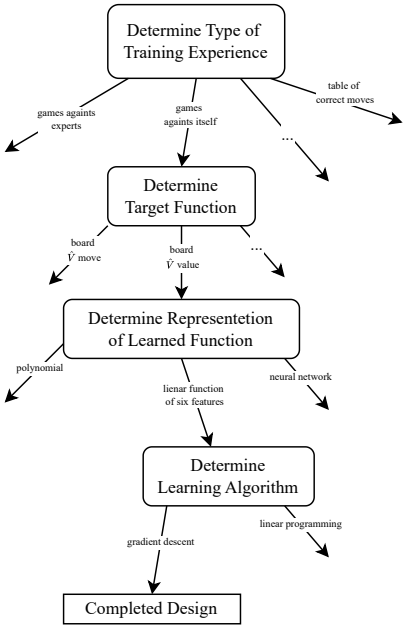


Figure 1.1: Design Choices

1.2 Types of Machine Learning Problems

There are different types of machine learning problems:

- Supervised Learning
  - Classification
  - Regression
- Unsupervised Learning
- Reinforcement Learning

Definition 1 *Given a function $f : X \to Y$, and a training set $X_D \subset X$ containing information about f,* ***learning*** *the function f means computing an approximated function $\hat{f}$ such that is much close as possible to f on X*

$$\hat{f}(x) \simeq f(x), \quad x \in X. \tag{1.4}$$

This is not a simple problem, f is not computable, so the difference $f - \hat{f}$, and X is usually uncountable ora big set, way bigger than the training set X_D.

Machine learning problems can be classified in terms of the input data set D, given a target function $f : X \to Y$ a problem is

- a **supervised learning** problem if $D = \bigcup_{i=1}^{n} \{(x_i, y_i)\} \subset X \times Y$.
- an **unsupervised learning** problem if $D = \bigcup_{i=1}^{n} \{(x_i)\} \subset X$.

- a **reinforcement learning** problem, the condition on the input dataset will be discussed later.

The problems can also be classified in terms of the target function $f : X \to Y$

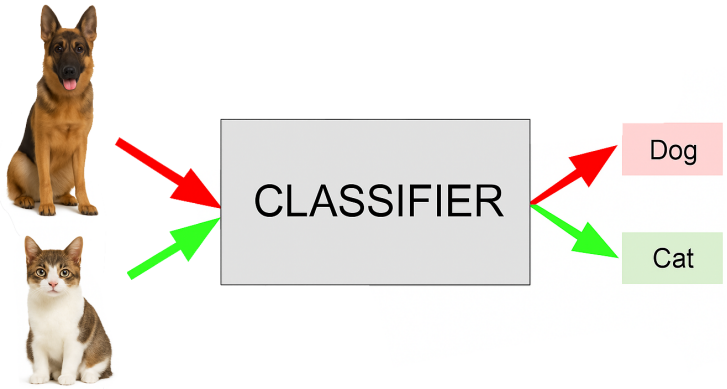
$$X = \begin{cases} A_1 \times \cdots \times A_m, \ A_i \text{ finite set} & \textbf{(Finite Discrete Problem)} \\ \mathbb{R}^n & \textbf{(Continuous)} \end{cases}$$

$$Y = \begin{cases} \mathbb{R}^k & \textbf{(Regression)} \\ \{C_1, C_2 \times C_k\} & \textbf{(Classification)} \end{cases}$$

special case (**Concept Learning**):

$$X = A_1 \times \cdots \times A_m, \ A_i \text{ finite set}$$
$$Y = \{0, 1\}$$

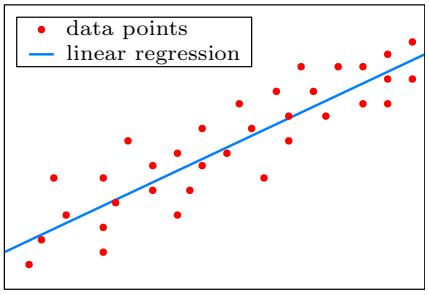
Classification problems is also known as *Pattern Recognition Problems*, the goal is to return the class to which a specific instance belong.



Some examples are:

- face/object/character recognition
- speech/sound recognition
- medical diagnosis
- document classification.

Regression problems consists in approximating real valued functions.

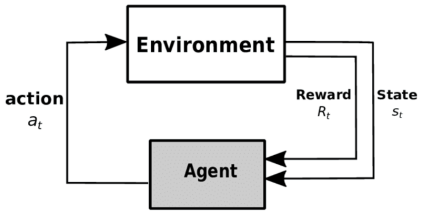


Unsupervised learning discovers data patterns without a specific output. The main goal is to understand what's normal in a dataset. *Clustering* is a key technique that groups similar data points. Applications include customer segmentation, image compression, and bioinformatics motif learning.

Reinforcement learning consists in learning a policy, which is a strategy that tells the agent what action to take in any given situation (or state) to maximize its long-term reward. This is often represented as a function that maps a state to an action. Unlike supervised learning, where the model is trained with labeled examples, in reinforcement learning agents learn through a process of trial and error.

They don't have a correct answer to guide them. Instead, they receive sparse and time-delayed rewards. This means the feedback (the reward) for a good action might not come immediately, and it might be a simple, numerical value (like +1 for winning a game or -1 for losing). Some examples are:

- Game playing: Think of programs that have learned to play chess, Go, or video games better than humans.
- Robotic tasks: A robot learning to navigate a room, pick up an object, or perform a specific manufacturing task.
- Any dynamical system with an unknown or partially known model: This is a broad category that includes things like optimizing traffic flow, managing a power grid, or controlling financial trading strategies.



In concept learning, the output consists in only two classes, the target function $c : X \to \{0, 1\}$ maps any kind of input in one of two distinct values. The following example model a program that predict if is a good day to play Tennis, the input set is

$$X = Day \times Outlook \times Temperature \times Humidity \times Wind$$

the output set is $PlayTennis = \{\text{Yes, No}\}$. An example of data samples:

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

1.3 Performance Evaluation

Usually, we call **Hypothesis** a possible learned function h, and we define H as the **Hypothesis space**, such space contains all possible function that can be learnt (all possible approximation of the target function). In this terms, a learning problem is described as a search in the hypothesis space using the given dataset D, that aims to find the best possible approximation h^*:

$$h^* = \arg\max_{h \in H} Performance(h, D) \tag{1.5}$$

1.3.1 Concept Learning

We focus now on the concept learning (binary classification), let's consider a target function

$$c : X \to \{0, 1\} \tag{1.6}$$

let $D \subset \{X, Y\}$ to be the sample set

$$D = \bigcup_{i=1}^{n} \{(x_i, c(x_i))\} \tag{1.7}$$

we denote X_D the point of X in the sample set. For now, we assume that the sample set does not have noise:

- noise dataset $D = \bigcup_{i=1}^{n} \{(x_i, c(x_i) + \varepsilon_i)\}, \quad \varepsilon_i \in \mathbb{R}$
- perfect dataset $D = \bigcup_{i=1}^{n} \{(x_i, c(x_i))\}$

in general, the dataset is noisy. We want to find a function $\hat{f}$ that approximate c, as we just said, the hypothesis space H is the set of all hypothesis h.Each hypothesis is a possible solution to the problem (1.5), it is possible to compare an hypothesis h with c on a sample in X_D.

Definition 2 *Given a target function c and a set of sample point X_D, an hypothesis h is **consistent** if*

$$h(x) = c(x), \quad \forall x \in X_D. \tag{1.8}$$

The **Version Space** $VS_{H,D}$ is the set of all consistent hypothesis:

$$VS_{H,D} = \{h \ : \ h(x) = c(x), \quad \forall x \in X_D\} \subset H. \tag{1.9}$$

A solution that does not lie in the version space is probably not a good solution.

Let's consider an example, let c to be the following target function

$$c : \mathbb{N} \to \{-, +\} \tag{1.10}$$

the dataset is

$$D = \{(1, +), (3, +), (5, +), (6, -), (8, -), (10, -)\} \tag{1.11}$$

given the hypothesis space H, we consider four different hypothesis

$$\begin{aligned} h_1(n) = + &\iff n \text{ is odd} \\ h_2(n) = + &\iff n \leq 5 \\ h_3(n) = + &\iff n \text{ is either 1 or prime} \\ h_4(n) = + &\iff n \in \{1, 3, 5\} \end{aligned}$$

if $n = 11$ we have

$$\begin{aligned} h_1(11) &= + \\ h_2(11) &= - \\ h_3(11) &= + \\ h_4(11) &= - \end{aligned}$$

we can't tell which of the four hypothesis is better than the others. Let's now consider a new hypothesis space H', defined as the power set of H

$$H' = \mathcal{P}(H) = \{I \ : \ I \subseteq H\} \tag{1.12}$$

The set H' contains more information than H, let $\theta \in H'$, we define the value of θ on x in a different way

$$\theta = \bigcup_i \{h_i\} \tag{1.13}$$

$$\theta(x) = \text{maj} \bigcup_i \{h_i(x)\} \tag{1.14}$$

where the function maj returns the element that is more frequent in a set. It's important to know that, if you have a set where two different items appear the exact same number of times, and no other item appears more often than either of them, then the majority element can't be identified, and the value of the function maj is undefined.

With the hypothesis space H', the point $n = 11$ can't be classified

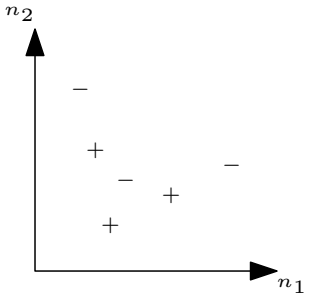
$$\theta(11) = \text{maj}\{h_1(11), h_2(11), h_3(11), h_4(11)\} = \text{maj}\{+, -, +, -\}. \tag{1.15}$$

Even if H' is more powerful than H, this hypothesis space can't classify an element that can be classified in H, this problem is called overfitting.

Now let's consider the following target function

$$c : \mathbb{N}^2 \to \{+, -\} \tag{1.16}$$

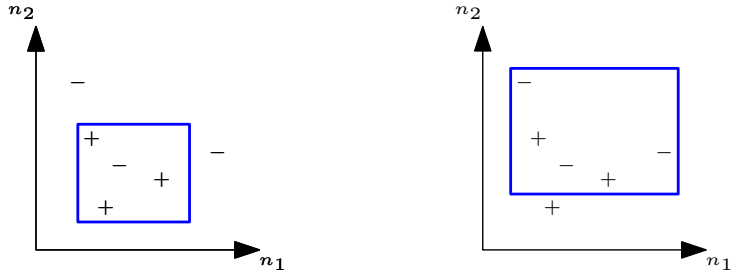
with the following dataset D that can be plotted on a 2D plane:



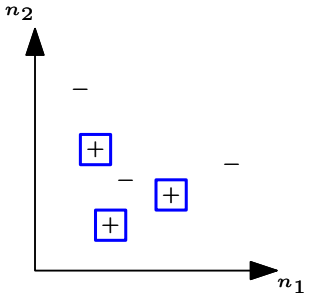
We consider H as the set of the functions that, assigns at all the points in a specific rectangle the $+$ value, and the $-$ value for all the points outside.

$$h \in H \iff \Big(\exists R := \{(x, y) \ : \ a \leq x \leq b \wedge c \leq y \leq d\} : h(x, y) = + \iff (x, y) \in R\Big)$$

For that data samples, does not exists a consistent hypothesis, because does not exists a rectangle that contains all the $+$ value point, letting outside the $-$ value points.



If we consider the hypothesis space $H' = \mathcal{P}(H)$, geometrically, we can represent a function by a finite number of rectangles, in this case consistent hypothesis are allowed, but no $-$ value points can be predicted.



Even in this example, a powerful representation of the hypothesis space lead to less generalization. This is known (as previously said) as **overfitting**, when the hypothesis space is not powerful enough, the problem could not be represented well, this is known as **underfitting**.

If n is a number that determines how big an hypothesis space, we can observe the following trend about the performance of a model.

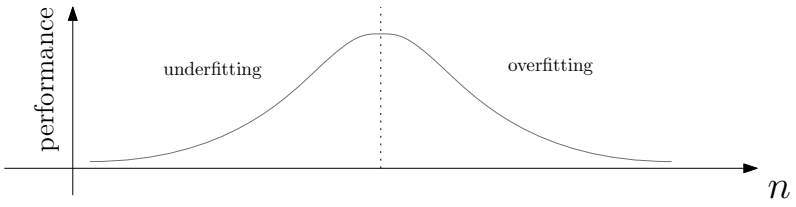


Figure 1.2: overfitting and underfitting trend

1.3.2 Error Estimation and Performance Metric

We want to define a metric for the performance of a model, to estimate how a given hypothesis h is good respect to the others. We have to introduce some statistical methods, let's consider a target function

$$f : X \to Y \tag{1.17}$$

where Y is countable (classification problem). Let $\mathcal{D}$ to be a probability distribution over X.

Definition 3 *If X is a countable subset of $\mathbb{R}^n$, $\mathcal{D}$ is a probability distribution on X if*

$$\forall x \in X, \quad \mathcal{D}(x) \in [0, 1] \tag{1.18}$$

and

$$\sum_{x \in X} \mathcal{D}(x) = 1$$

if X is an uncountable subset of $\mathbb{R}^n$, $\mathcal{D}$ is a probability distribution on X if

$$\forall x \in X, \quad \mathcal{D}(x) \geq 0 \tag{1.19}$$

$$\int_X \mathcal{D}(x)dx = 1$$

Definition 4 *Given a target function $f : X \to Y$, an hypothesis h and a probability distribution $\mathcal{D}$ on X, the **true error** is the probability that h will misclassify an instance $x \in X$ drawn according to the distribution $\mathcal{D}$:*

$$\text{error}_{\mathcal{D}}(h) = \mathbb{P}_{x \sim \mathcal{D}}(f(x) \neq h(x)) \tag{1.20}$$

Note: With $x \sim \mathcal{D}$, we mean that x was extracted from X according to the probability distribution $\mathcal{D}$. We remind that, if $\mathcal{D}(x) = p$, then, the probability of extracting x from X by taking a random element is p.

Since we can't compute f for all $x \in X$, the true error can't be computed. We need to define an estimation of the error given by the sample set extracted from X.

Definition 5 *Given a target function $f : X \to Y$, an hypothesis h and a sample (finite) set $\mathcal{S} \subset X$, the **sample error** is defined as follows:*

$$\text{error}_{\mathcal{S}}(h) = \frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} \delta(f(x) \neq h(x)) \tag{1.21}$$

where

- $\delta(\phi) = 1$ if ϕ is true
- $\delta(\phi) = 0$ if ϕ is false.

The sample error can be computed since, for each $x \in \mathcal{S}$, we know the value of $f(x)$. Is an approximation of the true error that depends from the sample set $\mathcal{S}$.

Since $\text{error}_{\mathcal{S}}(h)$ depends from the choice of $\mathcal{S}$, is not fixed, but we can model it as a random process, by extracting n random values from X according to the distribution $\mathcal{D}$ to construct $\mathcal{S}$, and we can evaluate the expected value of $\text{error}_{\mathcal{S}}(h)$ denoted

$$\mathbb{E}(\text{error}_{\mathcal{S}}(h)). \tag{1.22}$$

We want to formalize this expected value, in this case the sample error is a random variable that assign to each subset $\mathcal{S}$ of X of size n a number between 0 and 1:

$$\text{error}_{\mathcal{S}}(h) : \Omega \to [0, 1] \tag{1.23}$$
$$\Omega = \{\mathcal{S} \ : \ \mathcal{S} \subset X \wedge |\mathcal{S}| = n\} \tag{1.24}$$

in this context, n is fixed. The probability of getting a certain value γ from this random variable, is the probability to extract from X a subset $\mathcal{S}$ of n items such that, the sample error is γ, and this depends from the probability distribution $\mathcal{D}$ on X. The expected value $\mathbb{E}(\text{error}_{\mathcal{S}}(h))$ is now well defined.

1.3.3 Unbiased Estimation

Definition 6 *The **bias** is defined as the expected value of the difference between the sample error and the true error:*

$$\mathbb{E}(\text{error}_{\mathcal{S}}(h)) - \text{error}_{\mathcal{D}}(h). \tag{1.25}$$

We want to find an unbiased sample error, in such case:

$$\text{bias} = 0 \implies \mathbb{E}(\text{error}_{\mathcal{S}}(h)) = \text{error}_{\mathcal{D}}(h) \tag{1.26}$$

To compute an unbiased estimation, we need to train an evaluate an hypothesis h on different set, let D to be the dataset, we have to split D in two disjoint set

$$D = T \cup S \tag{1.27}$$
$$T \cap S = \emptyset \tag{1.28}$$

Usually, $\frac{|T|}{|D|} \simeq \frac{2}{3}$. We use T to train our learning function to get h, and then, we calculate the sample error on S

$$\text{error}_S(h) = \frac{1}{|S|} \sum_{x \in S} \delta(f(x) \neq h(x)) \tag{1.29}$$

is ideal to choose T and S such that they have similar probability distribution over the features, in this case the random variable error_S is an unbiased estimator for the true error $\text{error}_{\mathcal{D}}$.

1.3.4 The Cross Validation Algorithm and others Performance Metrics

There exists an algorithm to estimate the expected value of the sample error, more the dataset D is large, more the estimation will be precise. The algorithm 2 estimate the expected value of the error, with L is denoted a fixed learning algorithm:

- $h = L(T)$, is the result of the learning algorithm L applied on the training set T

Algorithm 2 K-Fold Cross Validation

Require: D, k, h, L
  partition D in k disjoint sets $S_1, S_2 \ldots, S_k$
  for $i = 1, 2 \ldots, k$ **do**
    $T_i \leftarrow D \backslash S_i$
    $h_i \leftarrow L(T_i)$
    $\delta_i = \text{error}_{S_i}(h_i)$
  end for
  return $\text{error}_{L,D} = \frac{1}{k} \sum_{i=1}^{k} \delta_i$

Algorithm 3 Accuracy Comparator

Require: D, k, h, L
  partition D in k disjoint sets $S_1, S_2 \dots, S_k$
  for $i = 1, 2 \dots, k$ **do**
    $T_i \leftarrow D \backslash S_i$
    $h_a \leftarrow L_a(T_i)$
    $h_b \leftarrow L_b(T_i)$
    $\delta_i = \text{error}_{S_i}(h_a) - \text{error}_{S_i}(h_b)$
  end for
  return $\bar{\delta} = \frac{1}{k} \sum_{i=1}^{k} \delta_i$

We define the accuracy of a learning algorithm L as

$$\text{accuracy} = 1 - \text{error}_{L,D}. \tag{1.30}$$

The cross-validation algorithm can be used to compare the accuracy of two different learning methods L_a and L_b, as shown in algorithm 3.

Now that we defined the sample error, we can give a formal definition of overfitting. Let h to be an hypothesis for a model, h is overfitting is exists an hypothesis h' such that

$$\text{error}_S(h) < \text{error}_S(h') \tag{1.31}$$

$$\wedge \tag{1.32}$$

$$\text{error}_{\mathcal{D}}(h) > \text{error}_{\mathcal{D}} \tag{1.33}$$

Let's consider other performance metrics in binary classification. Let $f : X \to \{-, +\}$ to be the target function ad let D to be a sample set such that, 90% of elements in D is of class $+$. An hypothesis that always returns $+$ will have an accuracy of 90%, in this scenario the dataset is unbalanced, so the accuracy is not a good performance metric for the model. We can consider a table that counts the number of points in the sample set that are well classified or misclassified by an hypothesis:

	predicted class	
true class	+	-
+	true positive	false negative
-	false positive	true negative

we can define two additional metrics that is useful when we deal with binary classification:

- the **recall** is the ability of the hypothesis to avoid false negatives and is defined as follows

$$\frac{\text{true positive}}{\text{true positive} + \text{false negative}} \tag{1.34}$$

- the **precision** is the ability of the hypothesis to avoid false positives and is defined as follows

$$\frac{\text{true positive}}{\text{true positive} + \text{false positive}} \tag{1.35}$$

the importance of these metrics depend on the application.

For the classification problems we can define an extension of the previous table, called **confusion matrix**, and report how many instances of class C_i are classified in class C_j, the main diagonal contains the accuracy for each class. An example is shown in figure 1.3.

We consider now some performance metrics for the regression problems. Let $f : X \to \mathbb{R}^d$ to be the target function, and let $\hat{f}$ to be the learned function, for each sample $(x_i, f(x_i))$ in the dataset, we can compute the euclidian distance

$$|\hat{f}(x_i) - f(x_i)|. \tag{1.36}$$

Let n to be the number of samples, the three main metrics are the following:

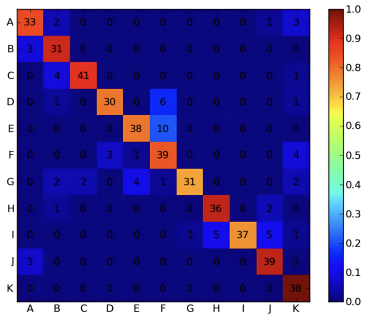


Figure 1.3: an example of a confusion matrix

- Mean Absolute Error

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{f}(x_i) - f(x_i)| \tag{1.37}$$

- Mean Squared Error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{f}(x_i) - f(x_i))^2 \tag{1.38}$$

- Root Mean Squared Error

$$RMSE = \sqrt{MSE} \tag{1.39}$$

The cross validation algorithm can be extended for the regression problems as shown in algorithm 4.

Algorithm 4 K-Fold Cross Validation for Regression

Require: D, k, h, L
  partition D in k disjoint sets $S_1, S_2 \ldots, S_k$
  for $i = 1, 2 \ldots, k$ **do**
    $T_i \leftarrow D \backslash S_i$
    $h_i \leftarrow L(T_i)$
    $\delta_i = MAE_{S_i}(h_i)$
  end for
  return $\text{error}_{L,D} = \frac{1}{k}\sum_{i=1}^{k}\delta_i$

CHAPTER

2

DECISION TREES

Let's consider the following classification problem, we have a target function

$$c : X \to C \tag{2.1}$$

and a data set

$$D = \bigcup_{i=1}^{n} \{(x_i, y_i)\} \subset X \times C \tag{2.2}$$

our approach is to define the hypothesis space H and search among all the possible solutions. Clearly we would like to find a solution $h^* \in D$ such that

$$h^*(x') \simeq f(x'), \quad \forall x' \notin D. \tag{2.3}$$

Recall: About the notation abuse, we say that $x' \in D$ if $\exists (x_i, y_i) \in D$ such that $x' = x_i$.

In this chapter we will see how to define the hypothesis space H with *decision trees*, let's assume that we are dealing with a finite problem, where

$$X = A_1 \times \cdots \times A_m \text{ with } A_i \text{ finite} \tag{2.4}$$

and clearly C is a finite set of classes.

Definition 7 *In the context of classification problems, a* ***decision tree*** *is a tree such that, each node is labeled with an attribute A_i, each edge is labeled with a value of an attribute $a_{i,j} \in A_i$, and each leaf assigns a classification value $c \in C$.*

A decision tree decides the class for each input $x \in X$, so we define the hypothesis space H as the set of all possible decision trees. The tree h' showed in figure 2.1, defines the following value

$$h'(a_{1,4}, a_{2,2}, a_{3,2}) = c_1 \tag{2.5}$$

Let's consider the following example about tennis, the input space X is

$$X = Outlook \times Temperature \times Humidity \times Wind \tag{2.6}$$
$$Outlook = \{sunny, overcast, rain\} \tag{2.7}$$
$$Temperature = \{hot, mild, cold\} \tag{2.8}$$
$$Humidity = \{normal, high\} \tag{2.9}$$
$$Wind = \{weak, strong\} \tag{2.10}$$

the target function $PlayTennis : X \to \{Yes, No\}$ should decides if the conditions are feasible to play tennis outside. The dataset is the onw shown in table 2.1.

Table 2.1: Dataset for the tennis problem

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

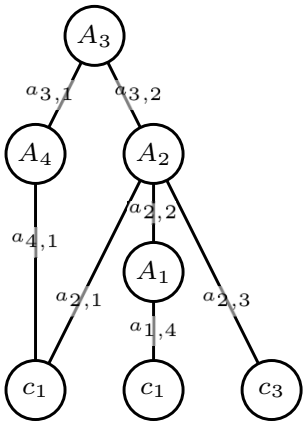
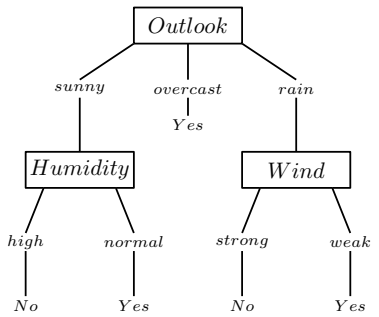


Figure 2.1: Example of a decision tree



Here's an example of a decision tree for the tennis problem. This representation is easy to classify a new instance that are not in the dataset, this tree defines rules such as: If the outlook is *overcast*, the output is Yes. The policy and the decisions are *explicit*, and provides an explanation for the outcomes.

In the hypothesis space H we have all possible decision trees, that are finite for this kind of problems. Decision trees represent a disjunction of conjunctions of constraints, that can be easily described by looking at the tree, traveling the traces of the tree backward starting from the outcomes. An example is shown in figure 2.2. The recursive algorithm 5 works to compute one decision tree given the dataset D.

We denote $N(+)$ a leaf node with the positive value, and $N(-)$ a leaf node with the negative value. If A_i is an attribute, we denote $N(A_i)$ an intermediate node with the attribute A_i. We denote $N(\ )$ a node for which a value has not yet been assigned.

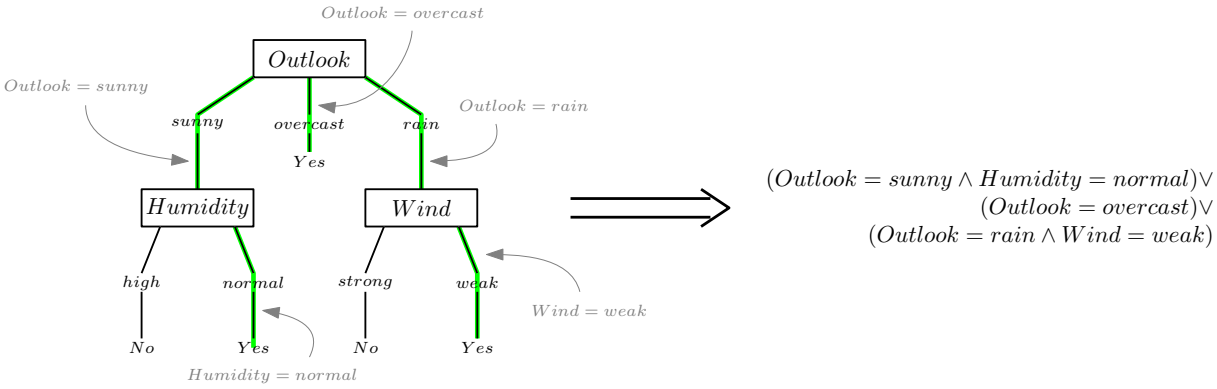


Figure 2.2: How paths in the tree transform into a logical formula

2.1 The Information Gain

The function `criteria`(Att) choose one attribute from the remaining ones in the set Att, and defines decision policy. What is the best attribute to choose during the algorithm? That procedure defines the behavior of the algorithm. Let's consider the following example for a reduced version of the tennis problem, with the dataset shown in table 2.2.

Table 2.2: Dataset for the reduced tennis problem

Day	*Outlook*	*Temperature*	*PlayTennis*
D1	Sunny	Hot	Yes
D2	Sunny	Cold	Yes
D3	Rain	Hot	No
D4	Rain	Cold	No

If we apply the algorithm on this sample, there will be two possible solutions, one for the case where we choose *Outlook* as the first choice for an attribute, one if we choose *Temperature*, the resulting trees as shown in figure 2.3.

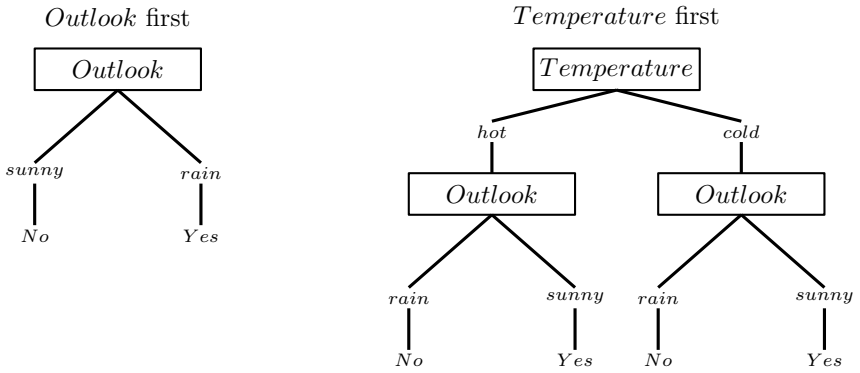


Figure 2.3: Different decision trees

In general, not making the correct choice in the attributes can lead to a tree that incapsulate less information.

Definition 8 *Given an example dataset S, let $p_\oplus$ to be the proportion of positive examples in S, and $p_\ominus = 1 - p_\oplus$ the proportion of negative examples in S, the* ***entropy*** *of S is the following value*

$$Entropy(S) = -p_\oplus \log_2 p_\oplus - p_\ominus \log_2 p_\ominus. \tag{2.11}$$

The curve of the *Entropy* function is shown in figure 2.4.

Algorithm 5 Computing Decision Tree for Binary Classification

Require: $D = \{(x_i, c_i)_i\}$, $Att = \{A_1 \dots, A_m\}$
  if $\forall (x_i, c_i) \in D, \quad c_i = +$ **then**
    return $N(+)$
  end if
  if $\forall (x_i, c_i) \in D, \quad c_i = -$ **then**
    return $N(-)$
  end if
  if $Att = \emptyset$ **then**
    $D_+ = \{(x_i, c_i) \in D \ : \ c_i = +\}$
    $D_- = D \backslash D_+$
    if $|D_+| > |D_-|$ **then**
      return $N(+)$
    end if
    if $|D_-| > |D_+|$ **then**
      return $N(-)$
    end if
    return a random choice between $N(+)$ and $N(-)$
  end if
  $A_i = \texttt{criteria}(Att) \in Att$
  we create a node $N(A_i)$
  for each $a_{i,j} \in A_i$ **do**
    create an edge $N(A_i) \longrightarrow_{a_{i,j}} N(\ )$
    $D' = \{(x_i, c_i) \in D \ : \ x_i(A_i) = a_{i,j}\}$
    $N(\ )$ =RecursiveCall$(D', Att \backslash \{A_i\})$
  end for
  return the tree with root $N(A_i)$

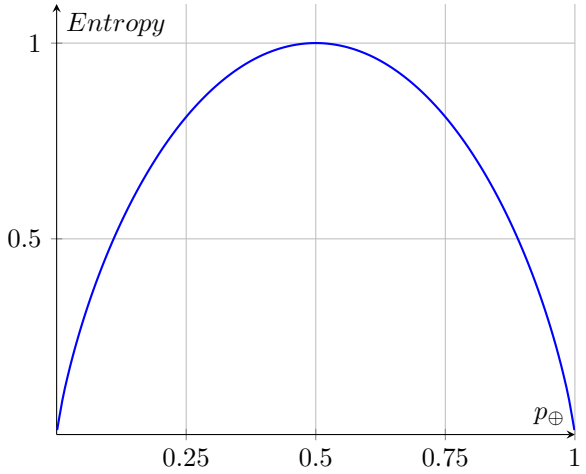


Figure 2.4: The curve of the *Entropy* function

The entropy is minimum if we have all positive samples or all negative samples. The goal is to reduce the entropy of a dataset. We can partition the dataset in function of the different attributes, and then calculate the entropy for all the partitions.

Given the table 2.2, we can create two partitions, one by selecting *Outlook*, and one by selecting *Temperature*, as shown in figure 2.5.

Splitting the dataset with outlook will generate two subsets with all positive/negative examples, so this two subsets will have an entropy of zero. In the other case, with temperature, the entropy will be 1.

In the case of multi-valued target function (classification problem that is not binary), we can define the

Day	Outlook	Temperature	PlayTennis
D1	**Sunny**	Hot	**Yes**
D2	**Sunny**	Cold	**Yes**
D3	**Rain**	Hot	**No**
D4	**Rain**	Cold	**No**

Day	Outlook	Temperature	PlayTennis
D1	Sunny	**Hot**	**Yes**
D3	Rain	**Hot**	**No**
D2	Sunny	**Cold**	**Yes**
D4	Rain	**Cold**	**No**

Figure 2.5: Partitions of the dataset

proportion of each class, denoted p_i, with n classes, the entropy is

$$Entropy(S) = \sum_{i=1}^{n} -p_i \log_2 p_i. \tag{2.12}$$

Note: We define $0 \log_2 0 = 0$.

Definition 9 *Given an example set S and an attribute A, we define the **information gain** as the expected reduction in entropy of S caused by knowing the value of the attribute A*

$$Gain(S, A) = Entropy(S) - \sum_{v \in A} \frac{|S_v|}{|S|} Entropy(S_v) \tag{2.13}$$

where

$$S_v = \{s \in S \ : \ s(A) = v\} \tag{2.14}$$

If we take in example the dataset in the table 2.1, we see that it have 9 positive samples and 5 negative samples, the entropy of the set is

$$-\frac{9}{14} \log_2 \left(\frac{9}{14}\right) - \frac{5}{14} \log_2 \left(\frac{5}{14}\right) = 0.94 \tag{2.15}$$

We want to calculate the information gain for the attribute $Wind$, that can be $Weak$ or $Strong$. If we partition the sub-dataset for this attribute, we get one dataset with 6 positive values and 2 negative values, and a sub-dataset with 3 positive values and 3 negative values.

Table 2.3: Split Dataset for the tennis problem

Outlook	*Temperature*	*Humidity*	**Wind**	*PlayTennis*
Sunny	Hot	High	Weak	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Overcast	Hot	Normal	Weak	Yes
Outlook	*Temperature*	*Humidity*	**Wind**	*PlayTennis*
Sunny	Hot	High	Strong	No
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Rain	Mild	High	Strong	No

we have that

$$|S_{weak}| = 8 \tag{2.16}$$
$$|S_{strong}| = 6 \tag{2.17}$$
$$|S| = 14 \tag{2.18}$$
$$Entropy(S_{weak}) = 0.811 \tag{2.19}$$
$$Entropy(S_{strong}) = 1 \tag{2.20}$$

The information gain is

$$Gain(S, Wind) = Entropy(S) - \frac{8}{14}Entropy(S_{weak}) - \frac{6}{14}Entropy(S_{strong}) \tag{2.21}$$

$$= 0.94 - \frac{8}{14}0.811 - \frac{6}{14} = 0.048. \tag{2.22}$$

This is a little information gain. A possible criteria for the choice of the next attribute in the algorithm 5, is to select the attribute with the highest information gain, in the case of the dataset 2.1, we have that

$$Gain(S, Outlook) = 0.245 \tag{2.23}$$

$$Gain(S, Humidity) = 0.151 \tag{2.24}$$

$$Gain(S, Wind) = 0.048 \tag{2.25}$$

$$Gain(S, Temperature) = 0.029 \tag{2.26}$$

We call `ID3` the decision tree algorithm 5 with the information gain criteria. This algorithm can be seen as an algorithm that makes a search in the space of all possible hypothesis/decision trees.

The hypothesis space of decision trees is *complete*, and we can represent all possible subsets of the input space. The algorithm returns one single hypothesis that is consistent with the dataset, this algorithm cannot find the global optimum function but a local minima.

This algorithm is not incremental and uses all the training set at each step.

2.2 Issues in Decision Trees

We have to consider the following problems:

- determining how deeply to grow the decision tree
- handling continuous attributes
- choosing appropriate attribute selection rule
- handling training data with missing attribute values
- handling attribute with different "cost".

Let D to be the dataset shown in table 2.1, we consider

$$D' = D \cup \langle sunny, hot, normal, strong, PlayTennis = No \rangle \tag{2.27}$$

`ID3` will generate two different decision trees T, T' for the different dataset D, D'. The question is:

is T' in general a better solution for this learning problem?

In the general case, we have a dataset D containing noisy data, and two decision trees T and T' obtained with different configurations of an `ID3`-like algorithm, where

$$accuracy_D(T') > accuracy_D(T) \tag{2.28}$$

we don't know if T' is a better solution, we should check the condition on a test set S

$$accuracy_S(T') > accuracy_S(T). \tag{2.29}$$

Generally, the trend in figure 2.6 is observed.

To avoid overfitting, we should stop to grow the tree when the data split is not statistically significant, or could grow a full tree and then apply the *post-prune*. With **pruning** we define the replacement of a sub-tree with a leaf node labelled as the most common class among samples filtered to the node.

If the dataset is limited, reducing the set of training examples (used as validation examples) can give bad results.

Decision trees can be expanded to *continuous values*, for example, if temperature is expressed in degrees, we could introduce a variation of the tree that will generate some "test" in terms of intervals

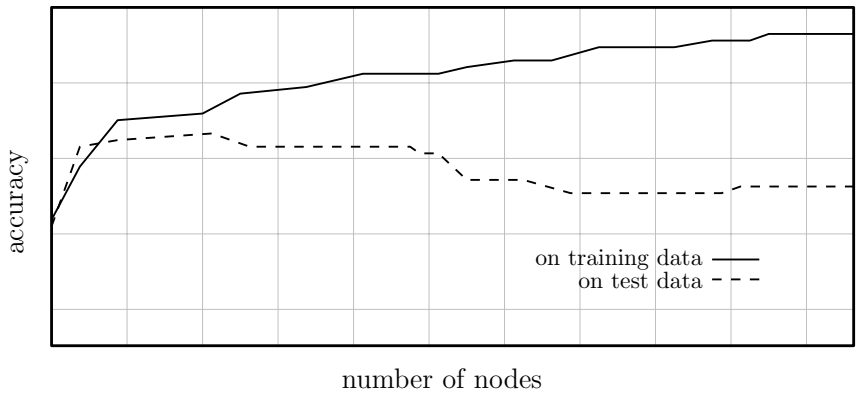


Figure 2.6: Decision trees overfitting

Algorithm 6 Post Pruning

Require: a dataset D
split data set in training D_T and validation D_V
generate a tree T using the data set D_T
while accuracy is not decreasing **do**
Evaluate the impact of pruning each node on the validation set
Greedily remove the one that most improves validation on accuracy
end while

- $Temperature = 82.5$
- $(Temperature > 72.3) = True, False$

So this decision trees will have tests on equalities and tests on real values. There are also several other ways of defining the criteria for selecting the best attributes to include as a node, there are other measures than the information gain. We cannot say that one is always better, it depends on the problem.

A dataset may have some samples with *missing* attributes, for example:

Day	*Outlook*	*Temperature*	*PlayTennis*
D1	Sunny	UNKNOWN	Yes
D2	Sunny	UNKNOWN	Yes
D3	UNKNOWN	Hot	No
D4	Rain	Cold	No

in such case, with decision trees we can use the samples by adding some information to replace the missing values

- If a node n tests an attribute A, we assign the most common value of A among other examples sorted to node n.
- We can assign the most common value for A among the other examples with the same target value.

There are other algorithms based on decision trees:

- we can generate a set of decision trees (called **forest**) with some random criteria and integrates their values into a final result.
- that integration of result could occur as a majority vote (most common class returned by all the trees)
- random forest are less sensitive to overfitting.

CHAPTER

3

PROBABILISTIC MODELS

3.1 Probability Recap

Consider the following action

$$A_t = \text{ leave the house } t \text{ minutes before the flight} \tag{3.1}$$

If we apply this action, what will be the answer to the question

> The action A_t will it get me there on time?

The answer can't be a simple *Yes* or *No*, because depends from a series of casual factors that are unpredictable (such as the traffic reports, or a possible flat tire). The probability is needed to represents the uncertainties:

- Given the available evidence, A_{25} will get me at the airport on time with probability 0.04
- Given the available evidence, A_{60} will get me at the airport on time with probability 0.84
- Given the available evidence, A_{1440} will get me at the airport on time with probability 0.999.

In this context, the set of *all possible elementary events* that can occurs is called **Sample Space**, and is usually denoted Ω.

Definition 10 *A **Discrete Probability Space** is a tuple $(\Omega, \mathcal{A}, \mathbb{P})$ where*

- *Ω is the sample set of the elementary events.*
- *$\mathcal{A} \subseteq \mathcal{P}(\Omega)$ is a σ-algebra, and contains all the events that can occurs, for which we can calculate a probability. $\mathcal{A}$ satisfies the following properties:*
  - $\Omega \in \mathcal{A}$
  - $A \in \mathcal{A} \implies A^C = \Omega \backslash A \in \mathcal{A}$
  - *if $\{A_i\} \subset \mathcal{A}$ is a countable sequence of events, then $\bigcup_i \{A_i\} \in \mathcal{A}$.*
- *$\mathbb{P}$ is the **probability measure**, is a function $\mathbb{P} : \mathcal{A} \to [0, 1]$ that assigns a probability to each event in $\mathcal{A}$ and satisfies the following properties:*
  - $\mathbb{P}(\Omega) = \sum_{\omega \in \Omega} \mathbb{P}(\{\omega\}) = 1$

– *if* $\{A_i\}$ *is a countable collection of disjoint sets in* $\mathcal{A}$*, then*

$$\mathbb{P}\left(\bigcup_i A_i\right) = \sum_i \mathbb{P}(A_i). \tag{3.2}$$

Definition 11 *Given a probability space* $(\Omega, \mathcal{A}, \mathbb{P})$*, a* ***random variable*** *is a function* $X : \Omega \to Y$*.*

The probability $\mathbb{P}$ induces a *probability distribution* on a random variable X:

$$\mathbb{P}(X = x_i) = \sum_{\omega \in \Omega \ : \ X(\omega) = x_i} \mathbb{P}(\omega). \tag{3.3}$$

Definition 12 *Given a probability space* $(\Omega, \mathcal{A}, \mathbb{P})$ *a* ***proposition*** *a is an event in* $\mathcal{A}$*, associated with a random variable* $A : \Omega \to \{True, False\}$ *such that*

$$a := A \text{ is } True := \{\omega \in \Omega \ : \ A(\omega) = True\}. \tag{3.4}$$

We can use the first order logic to combine and operates on propositions

$$\neg a := A \text{ is } False := \{\omega \in \Omega \ : \ A(\omega) = False\} \tag{3.5}$$
$$a \wedge b := A \text{ and } B \text{ is } True := \{\omega \in \Omega \ : \ A(\omega) = B(\omega) = True\} \tag{3.6}$$
$$\neg a \vee b := A \text{ is } False \text{ or } B \text{ is } True := \{\omega \in \Omega \ : \ A(\omega) = False \vee B(\omega) = True\} \tag{3.7}$$

Since a proposition is an element of $\mathcal{A}$, we can calculate the probability:

$$\mathbb{P}(\neg a \vee b) = \sum_{\omega \in \Omega \ : \ A(\omega) = False \vee B(\omega) = True} \mathbb{P}(\omega) \tag{3.8}$$

The Definition 3 refers to a probability distribution, if $X : \Omega \to Y$ is a random variable, the probability distribution $\mathcal{D}$ assigns a probability to each element in $\mathcal{P}(Y)$.

We will give now an example of a probability space. Let's consider the rolling of a dice, in this case we have

- the sample set $\Omega = \{1, 2, 3, 4, 5, 6\}$
- the σ-algebra $\mathcal{A} = \{\{1\}, \{1, 2\} \dots\}$
- the probability $\mathbb{P}$ defined as follows

$$\forall \omega \in \Omega, \ \mathbb{P}(\{\omega\}) = \frac{1}{6}. \tag{3.9}$$

The probability of getting an odd number is

$$\mathbb{P}(\{1, 3, 5\}) = \mathbb{P}(\{1\}) + \mathbb{P}(\{3\}) + \mathbb{P}(\{5\}) = \frac{1}{2} \tag{3.10}$$

Let's consider the following random variable

$$X : \Omega \to \{-20, 30\} \tag{3.11}$$
$$X(\omega) = 30 \iff \omega \in \{1, 2\} \tag{3.12}$$

X represents a gamble on the dice, if the result is less than 3, the player wins 30 euros, else, he loses 20 euros. The probability of winning is

$$\mathbb{P}(\{1, 2\}) = \frac{1}{3}. \tag{3.13}$$

If we have n random variables $\{X_i\}$, it is possible to consider the **joint probability distribution**, inducted by the joint probability function.

Let X_1, X_2 to be two distinct random variables on two distinct probability spaces

$$X_1 : \Omega_1 \to Y_1 \text{ for the space } (\Omega_1, \mathcal{A}_1, \mathbb{P}_1) \tag{3.14}$$
$$X_2 : \Omega_2 \to Y_2 \text{ for the space } (\Omega_2, \mathcal{A}_2, \mathbb{P}_2) \tag{3.15}$$

where $\mathcal{D}_1$ is the distribution on X_1 and $\mathcal{D}_2$ is the distribution on X_2, the joint distribution $\mathcal{D}$ is defined as follows:

$$\text{let } x_1 \in Y_1, \quad x_2 \in Y_2 \tag{3.16}$$

$$\mathcal{D}(X_1 = x_1,\ X_2 = x_2) = \mathbb{P}(\{(\omega_1, \omega_2) \in \Omega_1 \times \Omega_2 \ :\ X_1(\omega_1) = x_1 \wedge X_2(\omega_2) = x_2\}) \tag{3.17}$$

where $\mathbb{P} : \mathcal{A}_1 \times \mathcal{A}_2 \to [0, 1]$ is the joint probability and defines the probability that two events from the different probability space occurs.

If a and b are two events/propositions, we define the **conditional probability** as follows:

$$\mathbb{P}(a|b) = \frac{\mathbb{P}(a \wedge b)}{\mathbb{P}(b)}, \quad \text{if } \mathbb{P}(b) \neq 0 \tag{3.18}$$

it represents the probability that a occurs knowing that b is true. The **product rule** holds:

$$\mathbb{P}(a, b) = \mathbb{P}(a \wedge b) = \mathbb{P}(a|b)\mathbb{P}(b) = \mathbb{P}(b|a)\mathbb{P}(a) \tag{3.19}$$

The following equality is called **chain rule** and is needed to calculate the probability that multiple events $A_1, A_2 \dots A_n$ occurs:

$$\mathbb{P}(A_1, A_2 \dots A_n) = \prod_{i=1}^{n} \mathbb{P}(A_i|A_1, \dots A_{i-1}) = \tag{3.20}$$

$$\prod_{i=1}^{n} \left(A_i | \bigcap_{j=1}^{i-1} A_j \right). \tag{3.21}$$

Note: We will use the following abuse of notation, if $\mathbb{P}$ is the probability function for a probability space $(\Omega, \mathcal{A}, \mathbb{P})$, and X is a random variable on Ω, we use the same symbol $\mathbb{P}$ to denote a probability distribution on X

$$\mathbb{P}(A),\ A \in \mathcal{A} \ \text{ in this context } \mathbb{P} \text{ is the probability function} \tag{3.22}$$

$$\mathbb{P}(X = x_i) \ \text{ in this context } \mathbb{P} \text{ is the probability distribution.} \tag{3.23}$$

Theorem 1 *Let's consider a probability space $(\Omega, \mathcal{A}, \mathbb{P})$, let $A \in \Omega$ and let $\{B_1, B_2 \dots B_n\}$ to be a partition of Ω. The **Total Probability** formula states that:*

$$\mathbb{P}(A) = \sum_{i=1}^{n} \mathbb{P}(A \cap B_i) \tag{3.24}$$

since $\mathbb{P}(A \cap B_i) = \mathbb{P}(A|B_i)\mathbb{P}(B_i)$ we have

$$\mathbb{P}(A) = \sum_{i=1}^{n} \mathbb{P}(A|B_i)\mathbb{P}(B_i) \tag{3.25}$$

Let $X = \{X_1, X_2 \dots X_n\}$ a finite set of random variables, and let ω be an assignment of all the variables

$$\omega = (x_1, x_2 \dots x_n) \tag{3.26}$$

$$\omega \in \Omega = X_1 \times X_2 \cdots \times X_n \tag{3.27}$$

$$x_1 \in X_1 \tag{3.28}$$

$$x_2 \in X_2 \tag{3.29}$$

$$\vdots \tag{3.30}$$

$$x_n \in X_n. \tag{3.31}$$

There exists a joint probability distribution $\mathbb{P}$ on Ω. Let ϕ to be a proposition defined in terms of $X_1 \dots X_n$, for example

$$\phi = x_1 \vee x_2 \wedge \neg x_3 \tag{3.32}$$

ϕ is satisfied by all the events Ω_ϕ:

$$\Omega_\phi = \{\omega \in \Omega \ : \ \omega \text{ makes } \phi \text{ true } \} \tag{3.33}$$

the **Inference by Enumeration** formula states that, the probability that ϕ is satisfied is

$$\mathbb{P}(\phi) = \sum_{\omega \in \Omega_\phi} \mathbb{P}(\omega) \tag{3.34}$$

Definition 13 *Given a probability space $(\Omega, \mathcal{A}, \mathbb{P})$, and two events $A, B \in \mathcal{A}$, we say that A and B are **independent** if and only if*

$$\mathbb{P}(A, B) = \mathbb{P}(A \wedge B) = \mathbb{P}(A)\mathbb{P}(B). \tag{3.35}$$

Theorem 2 *Given a probability space $(\Omega, \mathcal{A}, \mathbb{P})$ and two events $A, B \in \mathcal{A}$, the **Bayes' Theorem** states that*

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)} \tag{3.36}$$

Let's now consider the *maximum likelihood problem*, given two random variabile X, Y, we want to know what is the most likely value of X knowing that $Y = y_i$

$$\arg\max_x \mathbb{P}(X = x|Y = y_i) = \arg\max_x \frac{\mathbb{P}(Y = y_i|X = x)}{\mathbb{P}(Y = y_i)} \tag{3.37}$$

for the properties of the arg max function:

$$\arg\max_x \frac{\mathbb{P}(Y = y_i|X = x)}{\mathbb{P}(Y = y_i)} = \arg\max_x \mathbb{P}(Y = y_i|X = x) \tag{3.38}$$

because $\mathbb{P}(Y = y_i)$ is constant.

3.2 Bayesian Learning

We want to use a probabilistic approach to model a machine learning problem, the methods presented in this section are surprisingly precise. Let's consider a target function for a classification problem

$$f : X \to V \tag{3.39}$$

where V is a finite set of classes, we define

$$\mathbb{P}(v|x, D) \tag{3.40}$$

$$\in X, v \in V \tag{3.41}$$

the probability that v is equal to $f(x)$ given the dataset D. Our goal is to define the learned function as follows

$$\hat{f}(x) = \arg\max_{v \in V} \mathbb{P}(v|x, D) = v^* \in V \tag{3.42}$$

In general, we would like to compute the whole distribution on V

$$\mathbb{P}(v|x, D), \quad \forall v \in V. \tag{3.43}$$

We define

$$\mathbb{P}(h|D) \tag{3.44}$$

the probability that an hypothesis $h \in H$ is the optimal solution for the problem knowing that D is the dataset. Applying the Bayes rule we get

$$\mathbb{P}(h|D) = \frac{\mathbb{P}(D|h)\mathbb{P}(h)}{\mathbb{P}(D)} \tag{3.45}$$

$\mathbb{P}(h)$ is the probability that an hypothesis $h \in H$ is the optimal solution without knowing anything about the dataset, if the hypothesis space H is finite, $\mathbb{P}(h) = \frac{1}{|H|}$. We define the **Maximum a Posteriori Hypothesis** as follows:

$$h_{MAP} = \arg\max_{h \in H} \mathbb{P}(h|D) = h_{MAP} = \arg\max_{h \in H} \frac{\mathbb{P}(D|h)\mathbb{P}(h)}{\mathbb{P}(D)} = h_{MAP} = \arg\max_{h \in H} \mathbb{P}(D|h)\mathbb{P}(h) \tag{3.46}$$

the denominator $\mathbb{P}(D)$ can be removed since doesn't affect the output of the arg max function (since it is constant). If we assume that all hypothesis have the same probability, the value can be written as

$$h_{MAP} = h_{ML} = \arg\max_{h \in H} \mathbb{P}(D|h). \tag{3.47}$$

this is the **Maximum likelihood hypothesis**. We can make a brute force approach by computing $\mathbb{P}(D|h)$ for each $h \in H$ (if h is finite).

3.2.1 Bayes Optimal Classifier

h_{MAP} is not necessary the optimal solution, let's consider $v_j \in V$, we can apply the total probability formula:

$$\mathbb{P}(v_j|x, D) = \sum_{h_i \in H} \mathbb{P}(v_j|x, h_i, D)\mathbb{P}(h_i|x, D) \tag{3.48}$$

where

- $\mathbb{P}(v_j|x, h_i, D)$ is the probability that $h_i(x) = v_j$ given D.
- The probability that $h_i(x) = v_j$ is independent from D (if h_i is given), so $\mathbb{P}(v_j|x, h_i, D) = \mathbb{P}(v_j|x, h_i)$
- $\mathbb{P}(h_i|x, D)$ is the probability that h_i is an optimal solution given x and D, since h_i does not depend on x, we have $\mathbb{P}(h_i|x, D) = \mathbb{P}(h_i|D)$

the formula can be rewritten as

$$\mathbb{P}(v_j|x, D) = \sum_{h_i \in H} \mathbb{P}(v_j|x, h_i)\mathbb{P}(h_i|D) \tag{3.49}$$

we define the **Bayes optimal classifier** as the following function

$$v_{OB}(x) = \arg\max_{v_j \in V} \sum_{h_i \in H} \mathbb{P}(v_j|x, h_i)\mathbb{P}(h_i|D) \tag{3.50}$$

usually, this is impracticable because we may can't enumerate all the possible hypothesis, or the hypothesis space could be too big. There is an example, Let's say that we have three possible hypothesis h_1, h_2, h_3 such that

$$\begin{aligned}
&\mathbb{P}(h_1|D) = 0.4, && \mathbb{P}(\ominus|x, h_1) = 0, && \mathbb{P}(\oplus|x, h_1) = 1 \\
&\mathbb{P}(h_2|D) = 0.3, && \mathbb{P}(\ominus|x, h_2) = 1, && \mathbb{P}(\oplus|x, h_2) = 0 \\
&\mathbb{P}(h_3|D) = 0.3, && \mathbb{P}(\ominus|x, h_3) = 1, && \mathbb{P}(\oplus|x, h_3) = 0
\end{aligned}$$

therefore

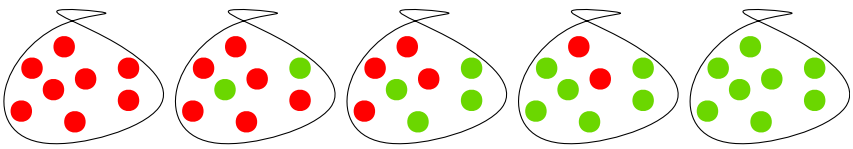
$$\sum_{i=1}^{3} \mathbb{P}(\oplus|x, h_i)\mathbb{P}(h_i|D) = 0.4$$
$$\sum_{i=1}^{3} \mathbb{P}(\ominus|x, h_i)\mathbb{P}(h_i|D) = 0.6$$

so

$$v_{OB}(x) = \arg\max_{v_j \in \{\oplus, \ominus\}} \sum_{i=1}^{3} \mathbb{P}(v_j|x, h_i)\mathbb{P}(h_i|D) = \ominus \tag{3.51}$$

The Bayes optimal classifier is the best possible learner given the same hypothesis space and the same knowledge. Is very powerful but impracticable because requires iteration on H.

Let's consider the example where the hypothesis space H consists in the following bag containing candies



with the following probability distribution

- $\mathbb{P}(h_1) = 10\%$, h_1 is the bag with 100% cherry candies.
- $\mathbb{P}(h_2) = 20\%$, h_2 is the bag with 75% cherry candies.
- $\mathbb{P}(h_3) = 40\%$, h_3 is the bag with 50% cherry candies.
- $\mathbb{P}(h_4) = 20\%$, h_4 is the bag with 25% cherry candies.
- $\mathbb{P}(h_5) = 10\%$, h_5 is the bag with 0% cherry candies.

We choose a random bag and we want to estimate what kind of bag is it, and what is the probability of extracting a candy of a specific flavor next. We define $\mathbb{P}(c)$ the probability of extract a cherry candy, $\mathbb{P}(l)$ the probability of extract a lime candy, the likelihood probability for lime candy is

$$\begin{aligned}\mathbb{P}(l, h_1) &= 0\\ \mathbb{P}(l, h_2) &= 0.25\\ \mathbb{P}(l, h_3) &= 0.5\\ \mathbb{P}(l, h_4) &= 0.75\\ \mathbb{P}(l, h_5) &= 1\end{aligned}$$

the probability of extracting a lime candy without having any data is

$$\sum_{i=1}^{5} \mathbb{P}(l|h_i)\mathbb{P}(h_i) = \frac{1}{2}. \tag{3.52}$$

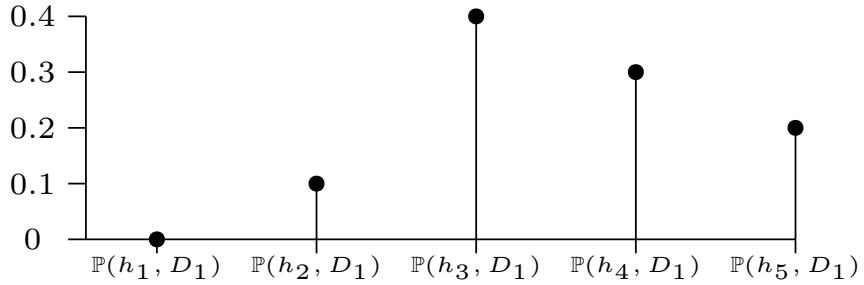
We consider now the following data set $D_1 = \{l\} = \{\text{first candy is lime}\}$, for the Bayes rule we know that

$$\mathbb{P}(h_i|D_1) = \frac{\mathbb{P}(D_1|h_i)\mathbb{P}(h_i)}{\mathbb{P}(D_1)} \tag{3.53}$$

we set $\mathbb{P}(D_1)^{-1} = \alpha$, that is a normalization factor, we have that

$$\begin{aligned}&\mathbb{P}(h_1|D_1) = \alpha\mathbb{P}(D_1|h_1)\mathbb{P}(h_1) = \alpha(0 \cdot 0.1) = 0\\ &\mathbb{P}(h_2|D_1) = \alpha\mathbb{P}(D_1|h_2)\mathbb{P}(h_2) = \alpha(0.25 \cdot 0.2) = \alpha 0.05 = 0.1\\ &\vdots\end{aligned}$$

the final distribution is



So, let's extract another candy, the dataset is now $D_2 = \{l, l\}$, the distribution become

$$\mathbb{P}(H|D_2) = \begin{pmatrix}0 & 0.038 & 0.308 & 0.346 & 0.308\end{pmatrix} \tag{3.54}$$

if $D_3 = \{l, l, l\}$ the distribution become

$$\mathbb{P}(H|D_3) = \begin{pmatrix}0 & 0.013 & 0.211 & 0.355 & 0.421\end{pmatrix} \tag{3.55}$$

we want to calculate the probability of extracting another lime candy

$$\begin{aligned}\mathbb{P}(l|D_3) &= \sum_{i=1}^{5} \mathbb{P}(l|h_i)\mathbb{P}(h_i|D_3)\\ &= 0 \cdot 0 + 0.25 \cdot 0.013 + 0.5 \cdot 0.211 + 0.75 \cdot 0.355 + 1 \cdot 0.421\\ &= 0.8\end{aligned}$$

if we use h_{MAP}, the answer will be $h_5 \implies \mathbb{P}(l|h_5) = 1$, which is not correct.

The graph shown in figure 3.1 shows how the probability that the solution is a given hypothesis h_i change with the number of lime candy extracted (with 0 cherry candy extracted).

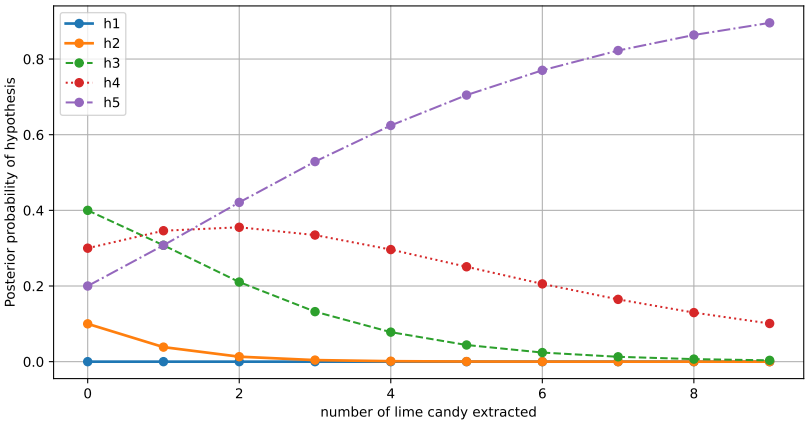


Figure 3.1: probability of each hypothesis

Let's now consider an example with a continuous hypothesis space

$$h_\theta \in H, \ \theta \in [0,1] \tag{3.56}$$
$$\mathbb{P}(c|h_\theta) = \theta \tag{3.57}$$
$$\mathbb{P}(l|h_\theta) = 1 - \theta \tag{3.58}$$

the **maximum likelihood hypothesis** is

$$h_{ML} = \arg\max_{\theta \in [0,1]} \mathbb{P}(D|h_\theta) \tag{3.59}$$

where with $\mathbb{P}(D|h_\theta)$ we denote the TODO for the properties of the arg max function, this value is equal to

$$h_{ML} = \arg\max_{\theta \in [0,1]} L(D, h_\theta) \tag{3.60}$$

if

$$L(D, h_\theta) = \log \mathbb{P}(D|h_\theta) \tag{3.61}$$

we have that

$$\mathbb{P}(D|h_\theta) = \prod_{j=1}^{|D|} \mathbb{P}(d_j|h_\theta) = \theta^c (1-\theta)^l \tag{3.62}$$

where

$$d_j \in D$$
$$l = \text{ number of lime candies in } D$$
$$c = \text{ number of cherry candies in } D.$$

So

$$L(D, h_\theta) = \log(\theta^c (1-\theta)^l) = c \log \theta + l \log(1-\theta) \tag{3.63}$$

to find the value of θ that maximize $\mathbb{P}(D|h_\theta)$ we can find the maximum value of $L(D, h_\theta)$ by considering the derivative:

$$\frac{dL}{d\theta} = \frac{c}{\theta} - \frac{l}{1-\theta}$$

we find the maximum value by imposing $\frac{dL}{d\theta} = 0$

$$\frac{c}{\theta} - \frac{l}{1-\theta} = 0 \iff \theta = \frac{c}{c+l} = \frac{c}{|D|} = \theta^* \tag{3.64}$$

so h_{θ^*} is the maximum likelihood hypothesis, this is trivial.

In the general case, we have a dataset $D = \bigcup_i d_i$, we assume a probability distribution $\mathbb{P}(d_i|h)$ and we find the Maximum likelihood estimation

$$h_{ML} = \arg\max_{h \in H} \log \mathbb{P}(D|h) \tag{3.65}$$

this can't be solved analytically for some distributions.

3.2.2 Naive Bayes Classifier

- with a discrete problem we can apply the Bayes Optimal Classifier, provides the best result but is not practical if the hypothesis space is large.
- with a continuous model we consider the maximum likelihood estimation when analytical solutions are available.

What happens if we can't apply the previous methods? We use the assumption of conditional independence.

Definition 14 *X is conditionally independent of Y given Z if*

$$\mathbb{P}(X, Y|Z) = \mathbb{P}(X|Y, Z)\mathbb{P}(Y|Z) = \mathbb{P}(X|Z)\mathbb{P}(Y|Z). \tag{3.66}$$

This assumption in practice is usually not true, so the solution will be approximated. Let's consider a target function

$$f : X \to V$$

where each instance $x \in X$ is described by different attributes

$$x = (a_1, a_2 \ldots, a_n)$$

we compute

$$\arg\max_{v \in V} \mathbb{P}(v|x, D) \tag{3.67}$$

we express x in terms of the attributes

$$\arg\max_{v \in V} \mathbb{P}(v|a_1, a_2 \ldots, a_n, D) \tag{3.68}$$

$$= \arg\max_{v \in V} \mathbb{P}(a_1, a_2 \ldots, a_n|v, D)\mathbb{P}(v|D) \tag{3.69}$$

then we assume that the attributes are conditionally independent

$$\mathbb{P}(a_1, a_2 \ldots, a_n|v, D) = \prod_{i=1}^{n} \mathbb{P}(a_i|v, D) \tag{3.70}$$

so the **Naive Bayes Classifier** is the learned function:

$$v_{NB}(x) = v_{NB}(a_1 \ldots, a_n) = \arg\max_{v \in V} \mathbb{P}(v|D) \prod_{i=1}^{n} \mathbb{P}(a_i|v, D) \tag{3.71}$$

Clearly the assumption that the attributes are independent ins wrong, consider the tennis example 2.1, if the outlook is *Rain*, the probability that the temperature is *hot* is lower than the probability that is *cool*.

The algorithm 7 show the procedure of the naive Bayes classifier estimation in the case of a classification problem.

Algorithm 7 Naive Bayes Classifier

Require: a dataset D, target function $f : X \to V$,$X = A_1 \times \cdots \times A_n$, a new instance $x = (a_1 \ldots, a_n)$
  for each $v_j \in V$ **do**
    $\hat{\mathbb{P}}(v_j|D) \longleftarrow$ `estimate`$(\mathbb{P}(v_j|D))$
    for each attribute A_k **do**
      for each $a_i^k \in A_k$ **do**
        $\hat{\mathbb{P}}(a_i^k|v_j, D) \longleftarrow$ `estimate`$(\mathbb{P}(a_i^k|v_j, D))$
      end for
    end for
  end for

After the probability $\hat{\mathbb{P}}$ are estimated, the new instance is classified

$$v_{NB} = \arg\max_{v \in V} \hat{\mathbb{P}}(v|D) \prod_{i=1}^{n} \hat{\mathbb{P}}(a_i^k|v, D). \tag{3.72}$$

How the probabilities are estimated? For $v_j \in V$ is considered the set the samples with v_j as label

$$D_{v_j} = \{(x, v) \in D \ : \ v = v_j\} \tag{3.73}$$

then the estimated probability is

$$\hat{\mathbb{P}}(v_j|D) = \frac{|D_{v_j}|}{|D|} \tag{3.74}$$

given $a_i^k \in A_k$, for $\hat{\mathbb{P}}(a_i|v_j, D)$, we consider the set of all samples such that, their A_k attribute have a_i^k as a value

$$D_{a_i^k} = \{(a^1, a^2 \ldots a^k \ldots, a^n, v) \in D \ : \ a^k = a_i^k\} \tag{3.75}$$

$$\hat{\mathbb{P}}(a_i^k|v_j, D) = \frac{|D_{a_i^k}|}{|D_{v_j|}} \tag{3.76}$$

usually, in the estimation are included two real number p, m such that

$$\hat{\mathbb{P}}(a_i^k|v_j, D) = \frac{|D_{a_i^k}| + mp}{|D_{v_j|} + m} \tag{3.77}$$

where

- p is a prior estimation for $\mathbb{P}(a_i^k|v_j, D)$
- m is a weight given to that prior estimation.

Even if the conditional independence assumption is often violated the naive Bayes classifier works surprisingly well, to make that the method works correctly, we don't need that the posteriori estimation $\hat{\mathbb{P}}(v_j|x, D)$ to be correct, but we need that

$$\arg\max_{v \in V} \hat{\mathbb{P}}(v|D) \prod_{i=1}^{n} \hat{\mathbb{P}}(a_i^k|v, D) = \arg\max_{v \in V} \mathbb{P}(v|D)\mathbb{P}(a_1 \ldots, a_n|v, D). \tag{3.78}$$

3.2.3 Text Classification

In this section we explain how to model a simple document classifier using the naive Bayes approach, the target function, given a document, returns the the class of that document (e.g. Scientific Article, Recipe, Poem, ecc...). This classifier have multiple application, in example:

- Spam classification for e-mail
- Sentiment analysis

Let f to be the target function, $C = \{c_1, \dots c_k\}$ the set of the classes, and $Docs$ the set of documents. $D = \bigcup_{i=1}^{n} \{(doc_i, c_i)\}$ is the dataset. We want to compute

$$c_{NB}(doc) = \arg\max_{c_j \in C} \mathbb{P}(c_j|D)\mathbb{P}(doc|c_j, D). \tag{3.79}$$

We want to express each document doc as a ordered list of words, we define a vocabulary

$$V = \{w_1, w_2 \dots w_n\} \tag{3.80}$$

that contains all possible words w_i that can appear in any document. Any document $doc_i \in Docs$ have a length (number of words) denoted m_i

$$doc_i = w_1 w_2 \dots w_{m_i} \tag{3.81}$$

The probability that a document doc_i is in class $c_j \in C$ given D is

$$\mathbb{P}(doc_i|c_j, D) = \prod_{k=1}^{m_i} \mathbb{P}(p_k = w_k|c_j, D) \tag{3.82}$$

where $\mathbb{P}(p_k = w_k|c_j|D)$ is the probability that, in a document doc_i of class c_j, the word p_k at position k is w_k. We assume that the order of the words does not affect the probability

$$\forall, k, k' \quad \mathbb{P}(p_k = w_k|c_j, D) = \mathbb{P}(p_{k'} = w_{k'}|c_j, D) \tag{3.83}$$

so we consider only

$$\mathbb{P}(w_k|c_j, D) = \text{ probability that } w_k \text{ occurs in a document of class } c_j \text{ given } D. \tag{3.84}$$

Each document can be represented as a n-dimensional vector, where the coordinate i is equal to k if in the document there are k words $w_i \in V$. This representation looses the information about the order of the words. In this context, we denote a vector

$$\mathbf{d} = (d_1, d_2 \dots d_n) \tag{3.85}$$

$$d_i = \text{ number of times words } w_i \text{ appear in } \mathbf{d} \tag{3.86}$$

we recall that $n = |V|$ is the number of distinct words in the vocabulary, the probability of classifying $\mathbf{d}$ as $c_j \in C$ given D is

$$\mathbb{P}(\mathbf{d}|c_j, D) = \frac{n!}{d_1! \dots d_n!} \prod_{i=1}^{n} \mathbb{P}(w_i|c_j, D)^{d_i} \tag{3.87}$$

we recall that $\mathbb{P}(w_i|c_j, D)$ is the probability of observing the word w_i given that the document $\mathbf{d}$ belongs to the class c_j. This probability is estimated with the maximum likelihood value:

$$\hat{\mathbb{P}}(\mathbf{d}|c_j, D) = \frac{\sum_{\mathbf{d} \in D} tf_{i,j} + \alpha}{\sum_{\mathbf{d} \in D} tf_j + \alpha n} \tag{3.88}$$

where

- $tf_{i,j}$ is the all-term frequency that w_i occurs in the document $\mathbf{d}$ given that $\mathbf{d}$ is of class c_j within the document dataset D.
- tf_j is the total number of words in all document of D that belongs to the class c_j.
- α is a smoothing parameter.

The algorithm 8 describes the methods that estimate the probabilities to classify new documents.

After we got the probabilities $\hat{\mathbb{P}}$, we classify a new document $doc = w_1 \dots w_m$ as follows

$$v_{NB}(doc) = \arg\max_{c_j \in C} \hat{\mathbb{P}}(c_j) \prod_{i=1}^{m} \hat{\mathbb{P}}(w_i|c_j) \tag{3.89}$$

Algorithm 8 Text Classifier

Require: a dataset of documents D, a set of classes C
  $V \longleftarrow$ all distinct words in D
  for each $c_j \in C$ **do**
    $docs_j \longleftarrow$ subset of D of document of class c_j
    $t_j \longleftarrow |docs_j|$
    $\hat{\mathbb{P}}(c_j) \longleftarrow \frac{t_j}{|D|}$
    $TF_j \longleftarrow$ total number of words in $docs_j$ (counting duplicates)
    for $w_i \in V$ **do**
      $TF_{i,j} \longleftarrow$ total number of times that w_i occurs in $docs_j$
      $\hat{\mathbb{P}}(w_i|c_j) \longleftarrow \frac{TF_{i,j}+1}{TF_j+|V|}$
    end for
  end for

3.3 Probabilistic Generative Models

3.3.1 Parametrized Hypothesis Space

Given a target function

$$f : X \to C$$

we have a parametrized hypothesis space if, the set of all possible solutions H, contains functions of the type

$$y(x, \boldsymbol{\theta}) \tag{3.90}$$

where $x \in X$ and $\boldsymbol{\theta}$ is the parameters of the function, an example is

$$y(x, \mathbf{w}) = y(x, (w_1, w_2, w_3, w_4)) = \sum_{i=1}^{4} x^i w_i \tag{3.91}$$

in this kinds of problem, the search of a solution, is the determination of the parameters $\boldsymbol{\theta}$. If the we have a high number of parameters, the model will be more powerful and the training will be more expensive (in computational terms).

A model could also be hyperparametrized, if the set of possible solution is parameterized by some fixed values that can't change during the training. An example of hyper parameter could be the grade of the polynomials functions in the hypothesis space H.

$$H = \left\{ \sum_{i=1}^{n} w_i x^i \right\}$$

n is an hyperparameter

$w_1, \dots w_n$ are the parameters

Let's consider a classification problem, the function to learn is

$$f : X \to \{c_1, c_2 \dots c_K\} \tag{3.92}$$

where the input space is continuous: $X \subseteq \mathbb{R}^d$. We need to evaluate the probability that a given point is in a given class

$$\mathbb{P}(c_k|c, D) \quad x \notin D \tag{3.93}$$

since the dataset is fixed in this context, we can write directly

$$\mathbb{P}(c_k|c) \quad x \notin D \tag{3.94}$$

is implicit that the probability depends on the dataset. The *main assumption* of this section is that there is a **Gaussian Distribution** over x, for each class c_k

$$\mathbb{P}(x|c_k) = \mathcal{N}(x, \mu_k, \Sigma_k) \tag{3.95}$$

where μ_k is the mean vector and Σ_k is the covariance matrix. To emphasize and reiterate that certain quantities are vectors, bold letters will be used in the notation.

$$\mathbb{P}(\mathbf{x}|c_k) = \mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_k, \Sigma_k) \tag{3.96}$$

3.3.2 Models for 2 Classes

Let's consider the case with two classes

$$f : X \to \{c_1, c_2\}$$

In this case, assuming that the distributions are gaussian, we have that

$$\mathbb{P}(\mathbf{x}|c_1) = \mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_1, \Sigma_1) \tag{3.97}$$
$$\mathbb{P}(\mathbf{x}|c_2) = \mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_2, \Sigma_2) \tag{3.98}$$

we recall that

$$X \subset \mathbb{R}^d \tag{3.99}$$
$$\boldsymbol{\mu}_i, \mathbf{x} \in \mathbb{R}^d \tag{3.100}$$
$$\Sigma_i \in Mat(d \times d) \tag{3.101}$$

we also have

$$\mathbb{P}(c_1) = p, \quad \mathbb{P}(c_2) = 1 - p \tag{3.102}$$

now, we have to estimate the parameters $\boldsymbol{\mu}_1, \Sigma_2, \boldsymbol{\mu}_1, \Sigma_2, p$ to use $\mathbb{P}(\mathbf{x}|c_i)$ and make predictions on samples that are not in the dataset, since for the Bayes theorem:

$$\mathbb{P}(c_1|\mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|c_1)\mathbb{P}(c_1)}{\mathbb{P}(\mathbf{x})} \tag{3.103}$$
$$\mathbb{P}(c_2|\mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|c_2)\mathbb{P}(c_2)}{\mathbb{P}(\mathbf{x})} \tag{3.104}$$

we get (for $i = 1, 2$):

$$\mathbb{P}(c_i|\mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|c_i)\mathbb{P}(c_i)}{\mathbb{P}(\mathbf{x})} = \tag{3.105}$$
$$\frac{\mathbb{P}(\mathbf{x}|c_i)\mathbb{P}(c_i)}{\mathbb{P}(\mathbf{x}|c_1)\mathbb{P}(c_1) + \mathbb{P}(\mathbf{x}|c_2)\mathbb{P}(c_2)} \tag{3.106}$$

setting

$$a(\mathbf{x}) = \ln\left(\frac{\mathbb{P}(\mathbf{x}|c_1)\mathbb{P}(c_1)}{\mathbb{P}(\mathbf{x}|c_2)\mathbb{P}(c_2)}\right) \tag{3.107}$$

we get

$$\mathbb{P}(c_i|\mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|c_i)\mathbb{P}(c_i)}{\mathbb{P}(\mathbf{x}|c_1)\mathbb{P}(c_1) + \mathbb{P}(\mathbf{x}|c_2)\mathbb{P}(c_2)} = \tag{3.108}$$
$$\frac{1}{1 + \exp(-a(\mathbf{x}))} = \sigma(a(\mathbf{x})) \tag{3.109}$$

we call σ the **sigmoid function**. This function "squeeze" the values from $\mathbb{R}$ to $[0, 1]$. By the assumption about the distribution, we know that

$$a(\mathbf{x}) = \ln\left(\frac{\mathbb{P}(\mathbf{x}|c_1)\mathbb{P}(c_1)}{\mathbb{P}(\mathbf{x}|c_2)\mathbb{P}(c_2)}\right) = \ln\left(\frac{\mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_1, \Sigma_1)p}{\mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_2, \Sigma_2)(1-p)}\right) \tag{3.110}$$

We do an ulterior assumption, the covariance matrices are equals: $\Sigma_1 = \Sigma_2$.

$$a(\mathbf{x}) = \ln\left(\frac{\mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_1, \Sigma)p}{\mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_2, \Sigma)(1-p)}\right) \tag{3.111}$$

At this point, we know that

$$\mathbb{P}(c_i|\mathbf{x}) = \sigma(a(\mathbf{x})) \tag{3.112}$$

so, given the parameters $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, p, \Sigma$, we can make prediction by the following decision rule:

- Given $\mathbf{x} \notin D$

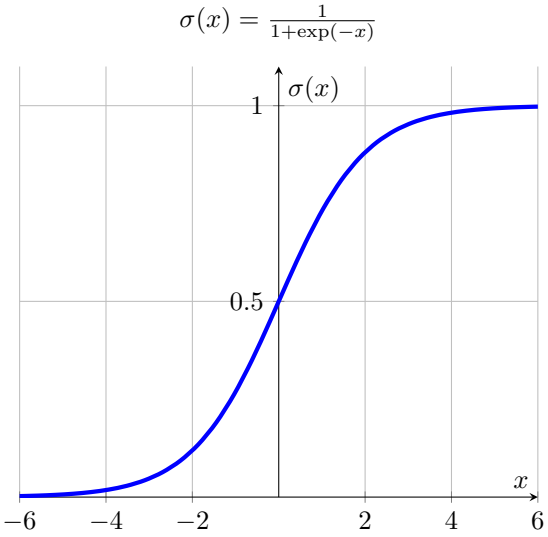


Figure 3.2: Sigmoid function curve

- if $\mathbb{P}(c_1|\mathbf{x}) > 0.5$ the point $\mathbf{x}$ is in class c_1
- else, if $\mathbb{P}(c_1|\mathbf{x}) \leq 0.5$ the point $\mathbf{x}$ is in class c_2

The parameters can be found with the maximum likelihood function, we have to introduce some notation to simplify the understanding process, the dataset is denoted:

$$D = \{(\mathbf{x}_n, t_n)_{n=1}^N\} \tag{3.113}$$

where $t_n = 1$ if $\mathbf{x}_n$ is in class c_1, else $t_n = 0$. This encoding is useful to write the likelihood function. We denote $\mathbf{t} = (t_n)_{n=1}^N$ the binary vector of the dataset's labels. The likelihood function is the probability that, given the parameters, the labels $\mathbf{t}$ are correct predictions:

$$\mathbb{P}(\mathbf{t}|p, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \Sigma, D) = \prod_{n=1}^{N} \left(p\mathcal{N}(\mathbf{x}_n, \boldsymbol{\mu}_1, \Sigma)\right)^{t_n} \left((1-p)\mathcal{N}(\mathbf{x}_n, \boldsymbol{\mu}_2, \Sigma)\right)^{(1-t_n)} \tag{3.114}$$

The ML solution is given by the parameters that maximize that function (we use the natural logarithm):

$$\hat{p}, \hat{\boldsymbol{\mu}}_1, \hat{\boldsymbol{\mu}}_2, \hat{\Sigma} = \arg \max_{p, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \Sigma} \ln \mathbb{P}(\mathbf{t}|p, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \Sigma, D) \tag{3.115}$$

In this context the function is analytical and we have a closed form solution, the parameters that maximize that function are

$$\hat{p} = \frac{N_1}{N}, \quad \text{where } N_1 = |\{(\mathbf{x}, t) \in D \ : \ t = 1\}| \tag{3.116}$$

$$\hat{\boldsymbol{\mu}}_1 = \frac{1}{N_1} \sum_{n=1}^{N} t_n \mathbf{x}_n \tag{3.117}$$

$$\hat{\boldsymbol{\mu}}_2 = \frac{1}{N_2} \sum_{n=1}^{N} (1 - t_n) \mathbf{x}_n \tag{3.118}$$

$$\tag{3.119}$$

The matrix Σ needs the definition of the two matrices

$$S_1 = \frac{1}{N_1} \sum_{(\mathbf{x}_n, t_n) \in D \ : \ t_n = 1} (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_1)(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_1)^T \tag{3.120}$$

$$S_2 = \frac{1}{N_1} \sum_{(\mathbf{x}_n, t_n) \in D \ : \ t_n = 0} (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_2)(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_2)^T \tag{3.121}$$

then

$$\hat{\Sigma} = \frac{N_1}{N} S_1 + \frac{N_2}{N} S_2 \tag{3.122}$$

note that the with $(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_i)(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_i)^T$ is denoted the matrix multiplication between a $d \times 1$ matrix and a $1 \times d$ matrix.

The function $a(\mathbf{x})$ includes non linear terms, since it includes the Gaussian distribution, apparently, is a non linear function. The following statement is crucial.

Theorem 3 *the function $a(\mathbf{x})$ is linear in $\mathbf{x}$.*

Proof: We have to show that, there exists $\mathbf{w} \in \mathbb{R}^d$ and $w_0 \in \mathbb{R}$ such that

$$a(\mathbf{x}) = \ln\left(\frac{\mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_1, \Sigma)p}{\mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_2, \Sigma)(1-p)}\right) = \mathbf{w}^T\mathbf{x} + w_0 \tag{3.123}$$

For the logarithms properties we have:

$$a(\mathbf{x}) = \ln\left(\frac{\mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_1, \Sigma)p}{\mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_2, \Sigma)(1-p)}\right) = \tag{3.124}$$

$$\ln\left(\frac{\mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_1, \Sigma)}{\mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_2, \Sigma)}\right) - \ln\left(\frac{p}{1-p}\right) \tag{3.125}$$

we expand the Gaussian distribution form:

$$\ln\left(\frac{\frac{1}{\sqrt{(2\pi)^d \det\Sigma}} \exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_1)^T\Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_1))}{\frac{1}{\sqrt{(2\pi)^d \det\Sigma}} \exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_2)^T\Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_2))}\right) - \ln\left(\frac{p}{1-p}\right) \tag{3.126}$$

since the covariance matrices are equals, the term $\frac{1}{\sqrt{(2\pi)^d \det\Sigma}}$ get cancelled

$$\ln\left(\frac{\exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_1)^T\Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_1))}{\exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_2)^T\Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_2))}\right) - \ln\left(\frac{p}{1-p}\right) \tag{3.127}$$

Due to the properties of exponentials and logarithms, the logarithm of a ratio is the difference of the logarithms:

$$-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_1)^T\Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_1) - (-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_2)^T\Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_2)) - \ln\left(\frac{p}{1-p}\right) \tag{3.128}$$

we multiply by -2 and we get

$$(\mathbf{x} - \boldsymbol{\mu}_1)^T\Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_1) - (\mathbf{x} - \boldsymbol{\mu}_2)^T\Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_2) + 2\ln\left(\frac{p}{1-p}\right) \tag{3.129}$$

$(\mathbf{x} - \boldsymbol{\mu}_i)^T\Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_i)$ are quadratic forms and can be expanded (remember that Σ is symmetric):

$$\mathbf{x}^T\Sigma^{-1}\mathbf{x} - 2\boldsymbol{\mu}_1^T\Sigma^{-1}\mathbf{x} + \boldsymbol{\mu}_1^T\Sigma^{-1}\boldsymbol{\mu}_1 - \mathbf{x}^T\Sigma^{-1}\mathbf{x} - 2\boldsymbol{\mu}_2^T\Sigma^{-1}\mathbf{x} + \boldsymbol{\mu}_2^T\Sigma^{-1}\boldsymbol{\mu}_2 + 2\ln\left(\frac{p}{1-p}\right) = \tag{3.130}$$

$$-2\boldsymbol{\mu}_1^T\Sigma^{-1}\mathbf{x} + \boldsymbol{\mu}_1^T\Sigma^{-1}\boldsymbol{\mu}_1 - 2\boldsymbol{\mu}_2^T\Sigma^{-1}\mathbf{x} + \boldsymbol{\mu}_2^T\Sigma^{-1}\boldsymbol{\mu}_2 + 2\ln\left(\frac{p}{1-p}\right) \tag{3.131}$$

reordering we get:

$$2(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T\Sigma^{-1}\mathbf{x} + (\boldsymbol{\mu}_1^T\Sigma^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2^T\Sigma^{-1}\boldsymbol{\mu}_2) + 2\ln\left(\frac{p}{1-p}\right) \tag{3.132}$$

we call r the constant $(\boldsymbol{\mu}_1^T\Sigma^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2^T\Sigma^{-1}\boldsymbol{\mu}_2) + 2\ln\left(\frac{p}{1-p}\right)$

$$2(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T\Sigma^{-1}\mathbf{x} + r \tag{3.133}$$

remember that we have to re-multiply by $-\frac{1}{2}$:

$$-(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \Sigma^{-1} \mathbf{x} - \frac{1}{2} r \tag{3.134}$$

we define

$$\mathbf{w} = (-(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \Sigma^{-1})^T \tag{3.135}$$

$$w_0 = -\frac{1}{2} r = -\frac{1}{2}(\boldsymbol{\mu}_1^T \Sigma^{-1} \boldsymbol{\mu}_1 - \boldsymbol{\mu}_2^T \Sigma^{-1} \boldsymbol{\mu}_2) + \ln\left(\frac{p}{1-p}\right) \tag{3.136}$$

in the end we have:

$$a(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 \tag{3.137}$$

■

3.3.3 Models for k Classes

In this case we have a target function

$$f : X \to \{c_1, c_2 \ldots, c_K\} \tag{3.138}$$

we encode the dataset as follows

$$D = \{(\mathbf{x}_n, \mathbf{t}_n)\}_{n=1}^N \tag{3.139}$$

where $\mathbf{t}_n = (t_{n1}, \ldots t_{nK})$ and $t_{nk} = 1 \iff$ the n-th sample is labeled as class c_k.

$$t_{nk} = 1 \iff f(\mathbf{x}_n) = c_k \tag{3.140}$$

we estimate $\mathbb{P}(c_k|\mathbf{x})$ as follows:

$$\mathbb{P}(c_k|\mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|c_k)\mathbb{P}(c_k)}{\sum_{j=1}^K \mathbb{P}(\mathbf{x}|c_j)\mathbb{P}(c_j)} = \frac{\exp(a_k)}{\sum_{j=1}^K \exp(a_j)} \tag{3.141}$$

where

$$a_j = \ln\left(\mathbb{P}(\mathbf{x}|c_j)\mathbb{P}(c_j)\right) \tag{3.142}$$

we assume that $\mathbb{P}(\mathbf{x}|c_k)$ is Gaussian

$$\mathbb{P}(\mathbf{x}|c_k) = \mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_k, \Sigma) \tag{3.143}$$

and for each class c_k, we have that

$$\mathbb{P}(c_k) = p_k \tag{3.144}$$

where

$$\sum_{j=1}^K p_j = 1 \tag{3.145}$$

so the parameters of the model are

$$p_1, p_2, \ldots p_K \in \mathbb{R} \tag{3.146}$$

$$\boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \ldots \boldsymbol{\mu}_K \in \mathbb{R}^d \tag{3.147}$$

$$\Sigma \in Mat(d \times d) \tag{3.148}$$

denoting $\mathbf{t}$ the set of labels in D

$$\mathbf{t} = \begin{pmatrix} \mathbf{t}_1 & \ldots & \mathbf{t}_N \end{pmatrix} \tag{3.149}$$

the maximum likelihood solution is given by:

$$\hat{p}_1, \ldots \hat{p}_k, \hat{\boldsymbol{\mu}}_1, \ldots \hat{\boldsymbol{\mu}}_k, \hat{\Sigma} = \arg \max_{p_k, \boldsymbol{\mu}_k, \Sigma} \ln\left(\mathbb{P}(\mathbf{t}|p_1, \ldots p_k, \boldsymbol{\mu}_1, \ldots \boldsymbol{\mu}_k, \Sigma)\right) \tag{3.150}$$

the solution can be found analytically and is

$$\hat{p}_k = \frac{N_k}{N} \text{ where } N_k = |\{(\mathbf{x}_n, \mathbf{t}_n) \in D \ : \ t_{nk} = 1\}| \tag{3.151}$$

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{N_k} \sum_{n=1}^{N} t_{nk} \mathbf{x}_n \tag{3.152}$$

$$S_k = \frac{1}{N_k} \sum_{n=1}^{N} t_{nk} (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_k)(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_k)^T \tag{3.153}$$

$$\hat{\Sigma} = \sum_{k=1}^{K} \frac{N_k}{N} S_k \tag{3.154}$$

a new sample $\mathbf{x}' \notin D$ can be predicted as follows:

$$h^*(\mathbf{x}') = \arg\max_k \frac{\exp(a_k)}{\sum_j \exp(a_j)} \tag{3.155}$$

where $a_k = \ln\left(\hat{p}_k \mathcal{N}(\mathbf{x}', \hat{\boldsymbol{\mu}}_k, \hat{\Sigma})\right)$.

3.3.4 Naive Bayes Assumption

We can consider an ulterior assumption, that the features of the samples are conditional independent, given a sample $\mathbf{x}_n \in \mathbb{R}^d$:

$$\mathbb{P}(\mathbf{x}_n | c_k) = \prod_{j=1}^{d} \mathbb{P}(x_{nj} | c_k) \tag{3.156}$$

in this case, for each class c_k, we don't have a multivariate Gaussian distribution $\mathcal{N}(\mathbf{x}_n, \boldsymbol{\mu}_k, \Sigma)$, but we have d one-dimensional Gaussian distribution

$$\mathbb{P}(x_{nj} | c_k) = \mathcal{N}(x_{nj}, \mu_{jk}, \sigma_{jk}) \tag{3.157}$$

so the parameters of the model are

$$\begin{matrix} p_k \in \mathbb{R} \\ \mu_{jk} \in \mathbb{R} \\ \sigma_{jk} \in \mathbb{R} \end{matrix} \text{ where } \begin{matrix} j = 1, \dots d \\ k = 1, \dots K \end{matrix} \tag{3.158}$$

the maximum likelihood solution can be found analytically:

$$\hat{p}_k = \frac{N_k}{N} \tag{3.159}$$

$$\hat{\mu}_{jk} = \frac{1}{N_k} \sum_{n=1}^{N} t_{nk} x_{nj} \tag{3.160}$$

$$\hat{\sigma}_{jk} = \sum_{n=1}^{N} t_{nk} (x_{nj} - \hat{\mu}_{jk})^2 \tag{3.161}$$

About the Number of Parameters

Given a classification problem

$$f : \mathbb{R}^d \to \{c_1 \dots c_K\}$$

what are the size (number of parameters) of the generative model? We have the probabilities

$$\mathbb{P}(c_k) = p_k$$

this counts as $K - 1$ parameters, since the first $K - 1$ can be used to calculate p_K

$$p_K = 1 - \sum_{k=1}^{K-1} p_k \tag{3.162}$$

then we have K vectors $\boldsymbol{\mu}_K$, of each k-th Gaussian distribution, since each vector have d real components, we have a total of kd parameter. In the end, we have

3.4 Discriminative Models

TODO

CHAPTER

4

LINEAR MODELS

4.1 Classification with Linear Models

A linear function on $\mathbf{x}$ is represented as follows

$$\mathbf{w}^T\mathbf{x} + w_0 \tag{4.1}$$

we want to use a compact notation, adding a component to the vector $\mathbf{x}$:

$$\mathbf{x} \in \mathbb{R}^d \longrightarrow \tilde{\mathbf{x}} \in \mathbb{R}^{d+1} \tag{4.2}$$

$$\mathbf{x} = \begin{pmatrix} x_1 & \dots x_d \end{pmatrix}^T \tag{4.3}$$

$$\tilde{\mathbf{x}} = \begin{pmatrix} 1 & x_1 & \dots x_d \end{pmatrix}^T \tag{4.4}$$

$$\tilde{\mathbf{w}} = \begin{pmatrix} w_0 & w_1 & \dots w_d \end{pmatrix}^T \tag{4.5}$$

so

$$\mathbf{w}^T\mathbf{x} + w_0 = \tilde{\mathbf{w}}^T\tilde{\mathbf{x}} = w_0 + w_1x_1 \dots w_dx_d. \tag{4.6}$$

In general, we have an error function $E(\mathbf{w})$ that we want to minimize

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E(\mathbf{w}) \tag{4.7}$$

Let's consider binary classification, there are a set of samples $D = \{(\mathbf{x}_n, t_n)_{n=1}^N\}$ where $\mathbf{x}_n \in \mathbb{R}^d$ and $t_n \in \{-, +\}$, we say that D is **linearly separable** if there exists an hyper plane

$$\{\mathbf{x} \ : \ \mathbf{a}^T\mathbf{x} + \mathbf{b} = 0\}$$

such that

$$t_n = + \iff \mathbf{a}^T\mathbf{x}_n + \mathbf{b} \leq 0 \tag{4.8}$$

$$t_n = - \iff \mathbf{a}^T\mathbf{x}_n + \mathbf{b} > 0 \tag{4.9}$$

In $\mathbb{R}^2$, a data set is linearly separable if there exists a line such that all and only the + samples are in one of the half space generated by the line.

If such an hyperplane exists, can be found with a linear model by tuning $d+1$ parameters $\mathbf{w}, w_0$ (we consider the compact notation and we denote $\mathbf{w}$ the whole set of parameters)

$$\mathbf{x}^T\mathbf{w} = \begin{pmatrix} 1 & x_1 & \dots & x_n \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{pmatrix}.$$

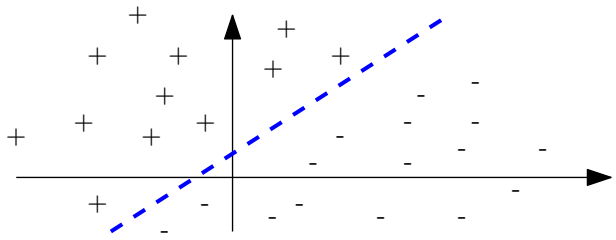


Figure 4.1: linearly separable data

The hyperplane that we want to find is called **linear discriminant**. We consider the binary notation for the samples, the target function have K classes as the image, if

$$(\mathbf{x}_n, \mathbf{t}_n) \in D$$

then $\mathbf{t}_n$ is a binary vector of K components, where the i-th coordinate of $\mathbf{t}_n$ is 1 if and only if $\mathbf{x}_n$ is labeled with class i

$$\mathbf{t}_n^T = \begin{pmatrix} 0 & 0 & \dots 1 & \dots & 0 \end{pmatrix}.$$

In the case of multiple classes, let's say K, we don't have a vector $\mathbf{w}$, but we have K linear functions $\mathbf{w}_1 \dots \mathbf{w}_k$, for each new sample $\mathbf{x}$ we calculate a prediction vector

$$\mathbf{y}(\mathbf{x}) = \begin{pmatrix} \mathbf{w}_1^T \mathbf{x} \\ \vdots \\ \mathbf{w}_K^T \mathbf{x} \end{pmatrix} \tag{4.10}$$

Since we have K vectors of d components as parameter, we can describe all the parameters in compact notation with a matrix W

$$W = \begin{pmatrix} \mathbf{w}_1 & \dots & \mathbf{w}_K \end{pmatrix} \tag{4.11}$$

the predictions are calculated by

$$\mathbf{y}(\mathbf{x}) = W^T \mathbf{x} \tag{4.12}$$

4.1.1 Least Square Methods

We define the T matrix as the matrix where the rows are the transpose vector $\mathbf{t}_n$

$$T = \begin{pmatrix} \mathbf{t}_1^T \\ \mathbf{t}_2^T \\ \vdots \\ \mathbf{t}_N^T \end{pmatrix} \in Mat(N \times K) \tag{4.13}$$

then, we define the matrix X as the matrix where the rows are the transpose vector $\mathbf{x}$ (with 1 as the first component, since we are using the compact notation).

$$X = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1d} \\ 1 & x_{21} & x_{22} & \dots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \dots & x_{Nd} \end{pmatrix} \in Mat(N \times d) \tag{4.14}$$

let $W = \begin{pmatrix} \mathbf{w}_1 & \dots & \mathbf{w}_K \end{pmatrix}$ to be the parameters of the linear model, the product XW is:

$$XW = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1d} \\ 1 & x_{21} & x_{22} & \dots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \dots & x_{Nd} \end{pmatrix} \begin{pmatrix} \mathbf{w}_0 & \mathbf{w}_1 & \dots & \mathbf{w}_K \end{pmatrix} = \tag{4.15}$$

$$\tag{4.16}$$

$$\begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1d} \\ 1 & x_{21} & x_{22} & \dots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \dots & x_{Nd} \end{pmatrix} \begin{pmatrix} w_{00} & w_{01} & \dots & w_{0d} \\ w_{10} & w_{11} & \dots & w_{1d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{K0} & w_{K1} & \dots & w_{Kd} \end{pmatrix} = \tag{4.17}$$

$$\tag{4.18}$$

$$\begin{pmatrix} \mathbf{w}_1^T\mathbf{x}_1 & \mathbf{w}_2^T\mathbf{x}_1 & \dots & \mathbf{w}_K^T\mathbf{x}_1 \\ \mathbf{w}_1^T\mathbf{x}_2 & \mathbf{w}_2^T\mathbf{x}_2 & \dots & \mathbf{w}_K^T\mathbf{x}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{w}_1^T\mathbf{x}_N & \mathbf{w}_2^T\mathbf{x}_N & \dots & \mathbf{w}_K^T\mathbf{x}_N \end{pmatrix} = \tag{4.19}$$

$$\tag{4.20}$$

$$\begin{pmatrix} \mathbf{y}(\mathbf{x}_1)^T \\ \mathbf{y}(\mathbf{x}_2)^T \\ \vdots \\ \mathbf{y}(\mathbf{x}_N)^T \end{pmatrix} \tag{4.21}$$

So XW are the predictions of the model on all the samples $\mathbf{x}_n$, the difference

$$XW - T = \begin{pmatrix} \mathbf{y}(\mathbf{x}_1)^T - \mathbf{t}_1 \\ \mathbf{y}(\mathbf{x}_2)^T - \mathbf{t}_2 \\ \vdots \\ \mathbf{y}(\mathbf{x}_N)^T - \mathbf{t}_N \end{pmatrix} \tag{4.22}$$

measure how different the prediction of the model are respect to the label on the sample data. Based on that, given a model W, we can define the **sum of squares error function**:

$$E(W) = \frac{1}{2} Tr\left((XW - T)^T (XW - T)\right) \tag{4.23}$$

Where with $Tr(A)$ we denote the trace of the matrix A. The parameters solution is found by the following optimization problem:

$$W^* = \arg\min_W \frac{1}{2} Tr\left((XW - T)^T (XW - T)\right) \tag{4.24}$$

this problem can be solved analytically and the closed form solution is

$$W^* = (X^T X)^{-1} X^T T \tag{4.25}$$

given a new sample $\mathbf{x}$, the prediction is calculated as follows:

$$\mathbf{y}(\mathbf{x}) = W^{*T}\mathbf{x} = \begin{pmatrix} \mathbf{w}_1^T\mathbf{x} \\ \mathbf{w}_2^T\mathbf{x} \\ \vdots \\ \mathbf{w}_K^T\mathbf{x} \end{pmatrix} \tag{4.26}$$

$\mathbf{x}$ is labeled in class C_k if

$$k = \arg\max_{1 \le j \le K} \mathbf{w}_j^T\mathbf{x} \tag{4.27}$$

This method is not robust to *outliers*, that is sample in D that are far from all the other samples, this points "weights" more for the linear discriminant, and the found solution will not be consistent with the dataset, as shown in figure 4.2.

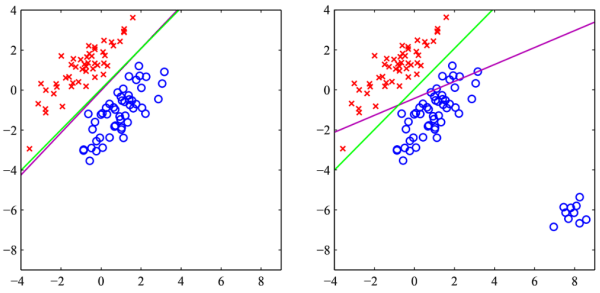


Figure 4.2: linear discriminant without and with outliers.

4.1.2 Perceptron

A perceptron is a linear model where the output is filtered by the Heaviside step function :

$$o(\mathbf{x}) = H(\mathbf{w}^T\mathbf{x}) = \begin{cases} 1 \text{ if } w^T\mathbf{x} > 0 \\ 0 \text{ if } w^T\mathbf{x} \leq 0 \end{cases} \tag{4.28}$$

In this context for binary classification, the classes are denoted -1 and $+1$. Given a dataset $D = \{(\mathbf{x}_n, t_n)_{n=1}^N\}$ with $t_n \in \{-1, +1\}$, we can consider as error function the squared error

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(t_n - o(\mathbf{x}_n)) = \frac{1}{2}\sum_{n=1}^{N}(t_n - H(\mathbf{w}^T\mathbf{x})) \tag{4.29}$$

to find the best model $\mathbf{w}$ we can apply the *gradient descent*, the partial derivative of E respect one fo the $d+1$ parameter is

$$\frac{\partial E}{\partial w_i} = \tag{4.30}$$

$$\frac{\partial}{\partial w_i}\frac{1}{2}\sum_{n=1}^{N}(t_n - H(\mathbf{w}^T\mathbf{x})) = \tag{4.31}$$

$$\frac{1}{2}\sum_{n=1}^{N}\frac{\partial}{\partial w_i}(t_n - H(\mathbf{w}^T\mathbf{x})) \tag{4.32}$$

since the function H is not differentiable, we consider the error function of the linear model without the filtering

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(t_n - \mathbf{w}^T\mathbf{x}) \tag{4.33}$$

in this case we can find the derivative analytically

$$\frac{\partial E}{\partial w_i} = \tag{4.34}$$

$$\frac{\partial}{\partial w_i}\frac{1}{2}\sum_{n=1}^{N}(t_n - \mathbf{w}^T\mathbf{x}) = \tag{4.35}$$

$$\frac{1}{2}\sum_{n=1}^{N}\frac{\partial}{\partial w_i}(t_n - \mathbf{w}^T\mathbf{x}) = \tag{4.36}$$

$$\sum_{n=1}^{N}(t_n - \mathbf{w}^T\mathbf{x}_n)\cdot(-\mathbf{x}_{ni}) \tag{4.37}$$

so we can compute the gradient of E at each point

$$\nabla E(\mathbf{w}) = \begin{pmatrix} \sum_{n=1}^{N}(t_n - \mathbf{w}^T\mathbf{x}_n)\cdot(-\mathbf{x}_{n1}) \\ \vdots \\ \sum_{n=1}^{N}(t_n - \mathbf{w}^T\mathbf{x}_n)\cdot(-\mathbf{x}_{nd}) \end{pmatrix} \tag{4.38}$$

we denote $\mathbf{x}_{ni} = x_{ni}$ to emphasize that is a real number (i-th component of vector $\mathbf{x}_n$). We can consider the thresholded of the step function:

$$\nabla E(\mathbf{w}) = \begin{pmatrix} \sum_{n=1}^{N}(t_n - H(\mathbf{w}^T\mathbf{x}_n))(-x_{n1}) \\ \vdots \\ \sum_{n=1}^{N}(t_n - H(\mathbf{w}^T\mathbf{x}_n))(-x_{nd}) \end{pmatrix} \tag{4.39}$$

$\mathbf{w}$ can be found by an iterative method described as follows:

1. initialize $\mathbf{w}$ with small random values
2. repeat the following until a termination condition:
   - $\mathbf{w} \leftarrow \mathbf{w} - \eta\nabla E(\mathbf{w})$
3. return $\mathbf{w}$

In the step

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\nabla E(\mathbf{w})$$

we are moving towards a direction where the error decrease.

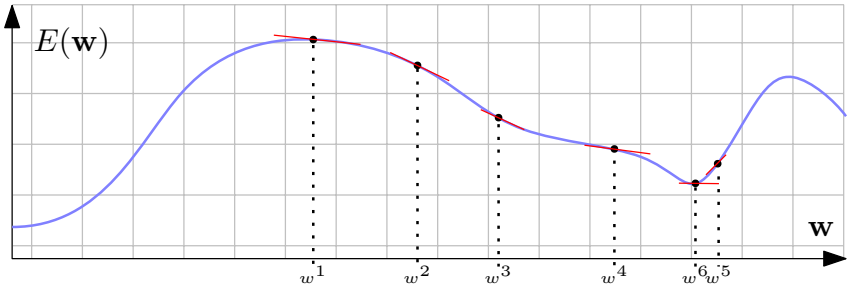


Figure 4.3: In this case $\mathbf{w} \in \mathbb{R}$, with w^k is denoted the value of $\mathbf{w}$ at the iteration k.

η is a real positive value, is an hyperparameter that tells "how strongly" the descent follows the direction of the gradient. The more η is small, the more the precision are high and the slow the algorithm will converge, linearly separable data and sufficient small η ar sufficient conditions to make the algorithm converge. Since at each step we have to compute the gradient by iterating on all the samples, we can consider a variant of the methods called *stochastic gradient descent* where the error function is calculated not on D but on a subset $S \subset D$ extracted randomly.

$$\nabla E_S(\mathbf{w}) = \begin{pmatrix} \sum_{\mathbf{x}_n \in S}(t_n - H(\mathbf{w}^T\mathbf{x}_n))(-x_{n1}) \\ \vdots \\ \sum_{\mathbf{x}_n \in S}(t_n - H(\mathbf{w}^T\mathbf{x}_n))(-x_{nd}) \end{pmatrix} \tag{4.40}$$

Suitable solution for the termination condition may be

- the error E reached a sufficient small value
- after a predefined number of iteration.

After we learned a model $\mathbf{w}^*$, a new sample $\mathbf{x}$ is predicted with $H(\mathbf{w}^{*T}\mathbf{x})$.

4.1.3 Fisher's Linear Discriminant

This method's consider a linear discriminant $\mathbf{w}$, and it adjust the direction of that linear function to maximize the separation of classes in the dataset, consider a dataset D with points in two classes, where N_1 are the number of points in class C_1 and N_2 are the number of points in class C_2. Two vectors $\mathbf{m}_1, \mathbf{m}_2$ are calculated in the following way

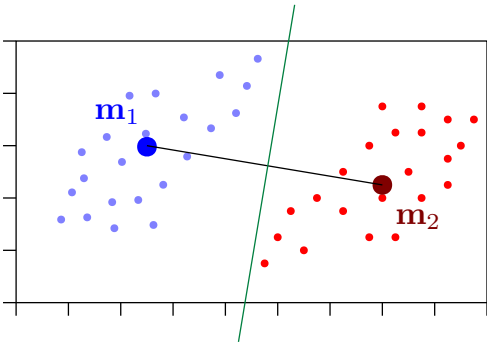
$$\mathbf{m}_1 = \frac{1}{N_1}\sum_{\mathbf{x}_n \in C_1}\mathbf{x}_n \tag{4.41}$$

$$\mathbf{m}_2 = \frac{1}{N_2}\sum_{\mathbf{x}_n \in C_2}\mathbf{x}_n \tag{4.42}$$

We solve the following optimization problem:

$$\begin{aligned} & \max J(\mathbf{w}) = \mathbf{w}^T(\mathbf{m}_1 - \mathbf{m}_2) \\ \text{subject to} \quad & \|\mathbf{w}\| = 1 \end{aligned}$$

$\mathbf{m}_1$ is the center of mass of the samples of class C_1 (analogous interpretation for $\mathbf{m}_2$ and C_2). The value of $\mathbf{w}$ that maximize $\mathbf{w}^T(\mathbf{m}_1 - \mathbf{m}_2)$ will be aligned to the vector $(\mathbf{m}_1 - \mathbf{m}_2)$.



TODO

4.1.4 Support Vector Machines

TODO

4.2 Linear Regression

We want to use a linear model for regression, the target function is

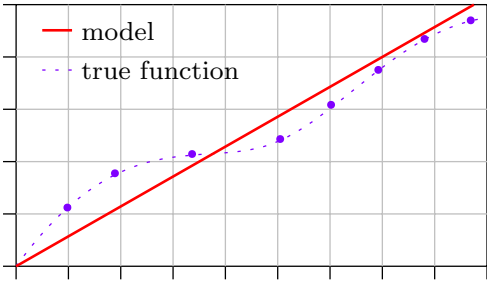
$$f : X \to Y \tag{4.43}$$
$$X \subseteq \mathbb{R}^d \tag{4.44}$$
$$Y \subseteq \mathbb{R} \tag{4.45}$$

The d parameters are $\mathbf{w}$ and the model is

$$y(\mathbf{x}) = y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x} \tag{4.46}$$

with

$$\mathbf{x}^T = \begin{pmatrix} 1 & x_1 & \dots & x_d \end{pmatrix}$$
$$\mathbf{w}^T = \begin{pmatrix} w_0 & w_1 & \dots & w_d \end{pmatrix}$$



It is possible to consider a non linear transformation of the input $\boldsymbol{\phi}(\mathbf{x})$

$$y(\mathbf{x}) = \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}) \tag{4.47}$$

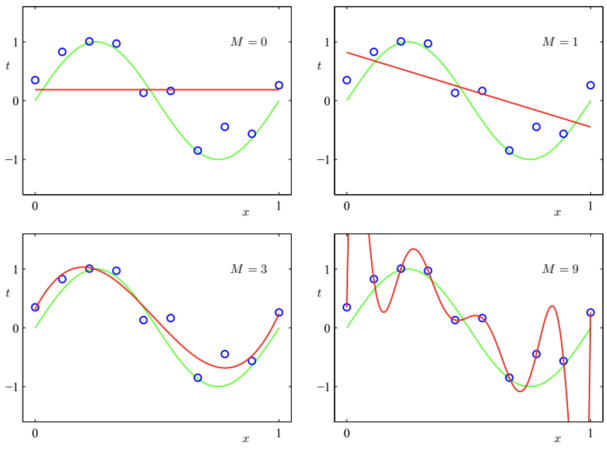
where

$$\boldsymbol{\phi}(\mathbf{x}) = \begin{pmatrix} \phi_0(\mathbf{x}) \\ \phi_1(\mathbf{x}) \\ \vdots \\ \phi_M(\mathbf{x}) \end{pmatrix} \tag{4.48}$$

An example, with $X = \mathbb{R}$, is $\phi_j(x) = x^j$:

$$y(x, \mathbf{w}) = \sum_{j=1}^{M} w_j x^j \tag{4.49}$$

M is an hyper parameter of the machine learning problem.



Let's now consider a noisy dataset

$$D = \{(\mathbf{x}_n, t_n)_{n=1}^N\} \tag{4.50}$$
$$f \text{ is the real target function} \tag{4.51}$$
$$t_n = f(\mathbf{x}_n) + \epsilon \tag{4.52}$$
$$\epsilon \in \mathbb{R}, \quad \epsilon \sim 0 \tag{4.53}$$

We assume that the noise on the samples have a Gaussian distribution

$$\mathbb{P}(\epsilon \text{ noise on sample } t_n) = \mathcal{N}(\epsilon, 0, \beta^{-1}) \tag{4.54}$$

where β^{-1} is the variance, we call β the *information coefficient*. Given such assumption:

$$\mathbb{P}(t_n|\mathbf{x}_n, \beta) = \mathcal{N}(t_n, f(\mathbf{x}_n), \beta^{-1}) = \text{ probability that } t_n \text{ is the real value of } f(\mathbf{x}_n) \tag{4.55}$$

Given a trained model y with parameters $\mathbf{w}$, under this assumption we have

$$\mathbb{P}(t|\mathbf{x}, \mathbf{w}, \beta) \tag{4.56}$$

the probability that t is the real outcome of $f(\mathbf{x})$ given the model $\mathbf{w}$ and the information coefficient β, assuming that the samples are independent from each other, we have the maximum likelihood function:

$$\mathbb{P}(\{t_1 \dots t_N\}|\{\mathbf{x}_1 \dots \mathbf{x}_N\}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathbb{P}(t_n|\mathbf{x}_n, \mathbf{w}, \beta) \tag{4.57}$$

given the assumptions:

$$\mathbb{P}(t_n|\mathbf{x}_n, \mathbf{w}, \beta) = \mathcal{N}(t_n|\mathbf{w}^T, \boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}) \tag{4.58}$$

we consider the log likelihood function:

$$\ln \mathbb{P}(\{t_1 \dots t_N\}|\{\mathbf{x}_1 \dots \mathbf{x}_N\}, \mathbf{w}, \beta) = \sum_{n=1}^{N} \ln \mathbb{P}(t_n|\mathbf{x}_n, \mathbf{w}, \beta)$$

Without entering in the details of the proof, it is shown that the log likelihood is proportional to the least squared error:

$$\ln \mathbb{P}(\{t_1 \dots t_N\}|\{\mathbf{x}_1 \dots \mathbf{x}_N\}, \mathbf{w}, \beta) \simeq \frac{1}{2} \sum_{n=1}^{N} (t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n))^2 \tag{4.59}$$

We can solve analytically the following problem:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{2}\sum_{n=1}^{N}(t_n - \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n))^2 \tag{4.60}$$

If we consider the vector of labels $\mathbf{t}$ and the matrix

$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \dots & \phi_M(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_2) & \dots & \phi_M(\mathbf{x}_2) \\ \vdots & \ddots & \vdots & \\ \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_N) & \dots & \phi_M(\mathbf{x}_N) \end{pmatrix} = \begin{pmatrix} \boldsymbol{\phi}(\mathbf{x}_1)^T \\ \boldsymbol{\phi}(\mathbf{x}_2)^T \\ \vdots \\ \boldsymbol{\phi}(\mathbf{x}_N)^T \end{pmatrix} \tag{4.61}$$

the least squared error can be rewritten in matrix form:

$$\frac{1}{2}\sum_{n=1}^{N}(t_n - \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n))^2 = \frac{1}{2}(\mathbf{t} - \Phi\mathbf{w})^T(\mathbf{t} - \Phi\mathbf{w}) \tag{4.62}$$

the optimal value for $\mathbf{w}$ is

$$\mathbf{w}_{ML} = (\Phi^T\Phi)^{-1}\Phi^T\mathbf{t} = \Phi^\dagger\mathbf{t} \tag{4.63}$$

computing $\Phi^\dagger\mathbf{t}$ is computationally expensive, we may consider a sequential algorithm, like the stochastic gradient descent.

$$\hat{\mathbf{w}} \leftarrow \hat{\mathbf{w}} - \eta\nabla E_S$$

with η a learning parameter, $S \subset D$ and E_S the least squared error evaluated on the samples in S.

4.2.1 Regularization

The *regularization* of a model is a technique applied to find a solution $\mathbf{w}$ with a smaller norm $\|\mathbf{w}\|$ respect the one found by considering a standard linear regression algorithm. A common choice is the following: We consider a regularization parameter

$$\lambda \in \mathbb{R}$$

We have $E_D(\mathbf{w})$ as the least squared error on the dataset D, we define a new error function

$$E_W(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} \tag{4.64}$$

Instead of solving the problem in equation (4.60), we solve

$$\arg\min_{\mathbf{w}} (E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})) = \tag{4.65}$$

$$\arg\min_{\mathbf{w}} \left(\frac{1}{2}\sum_{n=1}^{N}(t_n - \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n))^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}\right) \tag{4.66}$$

The graph in figure 4.4 show (in red) the solution of a linear regression with a 9 degree polynomial without regularization.

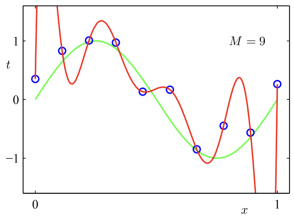


Figure 4.4: in green, the true function, in red, the learned function.

The graph in figure 4.5 show the solutions of a linear regression with a 9 degree polynomial with regularization and different choices for λ.

A wrong choice for λ may result in underfitting or overfitting model.In case of multiple classes (let's

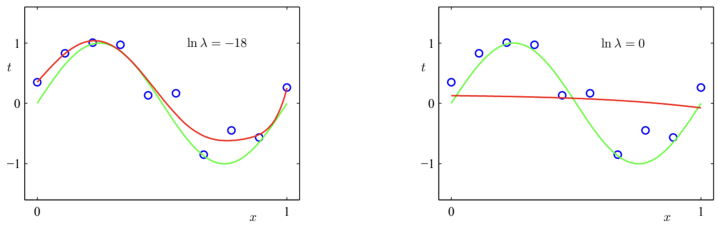


Figure 4.5: in green, the true function, in red, the learned function.

say K), instead of a single vector of $d+1$ parameters $\mathbf{w}$, we have K vectors, the parameters are contained in a $d+1 \times K$ matrix

$$W = \begin{pmatrix} w_{1,0} & w_{1,1} & \dots & w_{1,d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K,0} & w_{K,1} & \dots & w_{K,d} \end{pmatrix} \tag{4.67}$$

the model y is the following:

$$y(\mathbf{x}, W) = W^T \boldsymbol{\phi}(\mathbf{x}) \tag{4.68}$$

Instead of a label vector $\mathbf{t}$ in the dataset, we have a matrix T, the maximum likelihood solution (similarly as before) is the following matrix:

$$W_{ML} = (\Phi^T \Phi)^{-1} \Phi^T T = \Phi^\dagger T. \tag{4.69}$$

4.3 Kernel Methods

We have seen that, in linear regression, we have the following error function

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} (t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n))^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w} \tag{4.70}$$

written also as:

$$E(\mathbf{w}) = (\mathbf{t} - X\mathbf{w})^T (\mathbf{t} - X\mathbf{w}) + \lambda \|\mathbf{w}\| \tag{4.71}$$

the optimal solution is given by $\mathbf{w}^*$ such that

$$\nabla E(\mathbf{w}^*) = 0 \tag{4.72}$$

we saw that this problem can be solved analytically

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (\mathbf{w}^T \mathbf{x}_n - t_n)\mathbf{x}_n + \lambda \mathbf{w} = 0 \implies \tag{4.73}$$

$$\mathbf{w}^* = X^T (X X^T + \lambda I_N)^{-1} \mathbf{t} \tag{4.74}$$

let $\boldsymbol{\alpha} = (X X^T + \lambda I_N)^{-1} \mathbf{t}$, then

$$\mathbf{w}^* = X^T \boldsymbol{\alpha} \tag{4.75}$$

the model is the following

$$y(\mathbf{x}, \mathbf{w}^*) = \mathbf{w}^{*T} \mathbf{x} = \sum_{n=1}^{N} \alpha_n \mathbf{x}_n^T \mathbf{x} \tag{4.76}$$

note how in the model, it appears the dot product $\mathbf{x}_n^T \mathbf{x}$, this is a similarity measure between $\mathbf{x}_n$ and $\mathbf{x}$. The model y can be expressed in $\mathbf{w}$ and in $\boldsymbol{\alpha}$, the solution is:

$$\boldsymbol{\alpha} = (X X^T + \lambda I_N)^{-1} \mathbf{t} = (K + \lambda I_N)^{-1} \mathbf{t} \tag{4.77}$$

we set $K = X X^T$, this is called **Gram matrix** and the entries are the dot products between the samples $\mathbf{x}_n$, we recall that these entries is a measure of how "similar" two samples are.

$$K = \begin{pmatrix} \mathbf{x}_1^T \mathbf{x}_1 & \dots & \mathbf{x}_1^T \mathbf{x}_N \\ \vdots & \ddots & \vdots \\ \mathbf{x}_N^T \mathbf{x}_1 & \dots & \mathbf{x}_N^T \mathbf{x}_N \end{pmatrix} \tag{4.78}$$

Instead of using the dot product, we may consider a general **similarity function** that we call **kernel**

$$k : X \times X \to \mathbb{R} \tag{4.79}$$

the function $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$ is called *linear kernel*.

Definition 15 *A **kernel function** is a real valued function $k : X \times X \to \mathbb{R}$. Typically:*

- $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$
- $k(\mathbf{x}, \mathbf{x}') \geq 0$

The generalized model became

$$y(\mathbf{x}, \boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n k(\mathbf{x}_n, \mathbf{x}) \tag{4.80}$$

the kernel function k is arbitrary, the trainable parameters are $\boldsymbol{\alpha}$. The solution is

$$\boldsymbol{\alpha} = (K + \lambda I_N)^{-1}\mathbf{t} \tag{4.81}$$

where K is the generalized Gram matrix

$$K = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_N) \\ k(\mathbf{x}_2, \mathbf{x}_1) & \dots & k(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & \dots & k(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}. \tag{4.82}$$

Before entering in the details of kernel functions, we introduce a procedure called **normalization**, that aims to transform the input dataset D making feasible to define a similarity measure. Let $D = \{(\mathbf{x}_n, t_n)_{n=1}^N\}$, we denote X the samples $\mathbf{x}_n$ in D. Given

- min_j is the smallest value that appears as the j-th component in all the samples in X

$$min_j = \min_{n=1,\dots N} x_{n,j}. \tag{4.83}$$

- max_j is the largest value that appears as the j-th component in all the samples in X

$$max_j = \max_{n=1,\dots N} x_{n,j}. \tag{4.84}$$

We can normalize each sample $\mathbf{x}_n$, the transformed sample $\bar{\mathbf{x}}_n$ is the vector such that:

$$\bar{x}_{n,j} = \frac{x_{n,j} - min_j}{max_j - min_j} \tag{4.85}$$

this procedure is called **min max** normalization and ensures that

$$\bar{x}_{n,j} \in [0, 1], \quad \forall n \forall j \tag{4.86}$$

Another type of normalization is to resize all the vectors such that they have unitary length:

$$\bar{\mathbf{x}} = \frac{\mathbf{x}}{\|\mathbf{x}\|} \tag{4.87}$$

The last one is called **standardization**, let $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ to be the mean and the standard deviation in X, we transform vectors in the following way:

$$\bar{\mathbf{x}} = \frac{\mathbf{x} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \tag{4.88}$$

The division by $\boldsymbol{\sigma}$ is applied element wise

$$\bar{x}_j = \frac{x_j - \mu_j}{\sigma_j}. \tag{4.89}$$

In the following discussion, we assume that the input dataset is normalized. There are different families of kernel functions:

- **linear**:
$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$
- **polynomial**:
$$k(\mathbf{x}, \mathbf{x}') = (\beta \mathbf{x}^T \mathbf{x}' + \gamma)^d, \quad d \in \{2, 3, \dots\}$$
- **radial basis function**:
$$k(\mathbf{x}, \mathbf{x}') = \exp(-\beta \|\mathbf{x} - \mathbf{x}'\|^2)$$
- **sigmoid**:
$$k(\mathbf{x}, \mathbf{x}') = \tanh(\beta \mathbf{x}^T \mathbf{x}' + \gamma)$$

4.3.1 Kernelized Support Vector Machines

We saw that, for the support vector machines, the solution $\mathbf{w}^*$ has the form:

$$\mathbf{w}^* = \sum_{n=1}^{N} \alpha_n \mathbf{x}_n \tag{4.90}$$

it is composed by a weighted sum of the samples. The linear model for binary classification is

$$y(\mathbf{x}, \boldsymbol{\alpha}) = \text{sign}\left(w_0 + \sum_{n=1}^{N} \alpha_n \mathbf{x}_n^T \mathbf{x} \right) \tag{4.91}$$

Since it appears the dot product $\mathbf{x}_n^T \mathbf{x}$, we can apply the *kernel trick* and substitute $\mathbf{x}_n^T \mathbf{x}$ with a general kernel function:

$$y(\mathbf{x}, \boldsymbol{\alpha}) = \text{sign}\left(w_0 + \sum_{n=1}^{N} \alpha_n k(\mathbf{x}_n, \mathbf{x}) \right) \tag{4.92}$$

The Lagrangian function for the kernelized SVM classification became

$$\mathcal{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \tag{4.93}$$

we know that the solution $\mathbf{w}^*$ expressed in terms of the Lagrangian multipliers $\mathbf{a}^*$ is

$$\mathbf{w}^* = \sum_{n=1}^{N} a_n^* t_n \mathbf{x}_n \tag{4.94}$$

and $\mathbf{a}^*$ is the solution to the optimization problem

$$\mathbf{a}^* = \arg\max_{\mathbf{a}} \mathcal{L}(\mathbf{a}) \tag{4.95}$$

$$\text{subject to} \tag{4.96}$$

$$a_n \geq 0 \tag{4.97}$$

$$\sum_{n=1}^{N} a_n t_n = 0 \tag{4.98}$$