Notes on probability theory and probabilistic Machine Learning

Alessio Benavoli

August 10, 2020

Contents

1 Discrete Probability Theory **5**

1.1 What is a probability? 5
1.2 Finite possibility space 6
1.3 Frequentist interpretation 7
1.4 Rules of probability 8
1.4.1 Probability mass function 11
1.5 Conditional probability 11
1.6 Two or more variables 14
1.6.1 More about Bayes' rule 17
1.7 Inference 19
1.7.1 Case study: spam filters 20
1.7.2 Learning 27
1.8 Exercises with solutions 29
1.8.1 Solutions 30
1.9 Exercises without solutions 34

2 Learning **37**

2.1 Maximum Likelihood Estimator 37
2.1.1 Reguralisation 41
2.2 Application: spam filter continues 42
2.2.1 Reguralisation 46
2.2.2 Naive hypothesis 46
2.3 Language Modelling 49
2.3.1 Unigram Language Model 50
2.3.2 N-gram Language Models 53
2.4 Application: spam filter continues 55
2.5 Further material 56
2.6 Exercises with solutions 56
2.6.1 Solutions 57
2.7 Exercises without solutions 59

3 Continuous Probability **61**
3.1 Continuous variables 61
3.2 Probability Density Function 64
3.3 Rules of probability for PDFs 65
3.4 Gaussian distribution 66
3.4.1 Conditional probability 71
3.4.2 More than one variable 78
3.5 Exercises without solutions 82

4 Learning II **85**
4.1 MLE of the parameters of a Gaussian PDF 85
4.2 Reguralisation 86
4.3 Another way of learning 89
4.4 Application: sales forecast 92
4.4.1 MLE derivation of least squares 94
4.4.2 A better way to learn α, β 95
4.5 Exercises with solutions 101
4.5.1 Solutions 102
4.6 Exercises without solutions 104

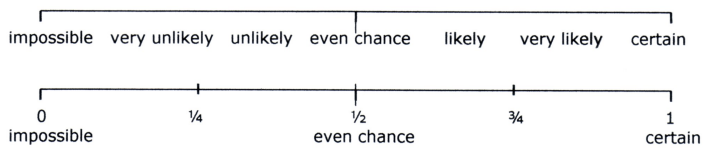
Chapter 1

Discrete Probability Theory

This chapter introduces probability theory, focusing on discrete variables.

1.1 What is a probability?

A probability is a numerical measure of the possibility of happening of a certain event (fact). A probability is a number between 0 and 1. If we assign 0 to the probability of an event, this indicates that the event cannot happen. If we assign 1 to the probability of an event, this indicates that the event will happen with certainty. So what does 0.5 mean? It means that the occurrence or non-occurrence of a certain event have equal possibility.



There are two interpretations or ways of thinking about the meaning of probability:

1. *frequentist*;

2. *subjective* (also called Bayesian).

Figure 1.1: Thomas Bayes

1.2 Finite possibility space

We always talk about the probability of *something*, for instance the probability that a *coin toss* results in heads or the probability that a dice roll returns 2. We generically call "tossing a coin" or "rolling a dice" an *experiment*.

Definition: The **space of possibility** is the set of possible results of a certain experiment. We denote it with Ω.

For a coin toss, $\Omega = \{head, tail\}$. For a dice, $\Omega = \{1, 2, 3, 4, 5, 6\}$.[1] For the weather forecast, $\Omega = \{sunny, cloudy, rainy\}$. For a spam filter, $\Omega = \{spam, ham\}$. In this Chapter, we will only consider the cases where Ω is a finite set (that it has a finite number of elements). In Chapter 3 we will consider infinite sets.

[1] The objects in the set are called its elements. The set is denoted by enclosing the list of objects in curly brackets.

Definition: An **event** is any subset of Ω or, said it differently, is any result of an experiment. An event that is verified by a single result (a single element of Ω) is called **elementary event**.

For instance, the face 2 is an elementary event in a dice roll, while the event "2 or 3" (that is the subset $\{2, 3\}$) is not an elementary event. Since events are subsets of Ω, we refer to events using the language of set theory.[2] Events are usually indicated with capital letters.

[2] You can find here a review of basic concepts and examples in set theory https://en.wikipedia.org/wiki/Set_theory#Basic_concepts_and_notation

Definition:

- Given two events A and B, $A \cup B$ indicates the **union**, or the event consisting of the occurrence of event A or event B.
- Given two events A and B, the **intersection** is indicated by $A \cap B$, that is, the event consisting of the occurrence of both event A and event B.
- If $A \cap B = \emptyset$ the two events A and B are called **mutually exclusive** (that is they cannot occur simultaneously).
- The **complement** of an event A with respect to Ω, denoted as $\Omega \backslash A$, is called a complementary event or negation of A and indicates its non-occurrence (i.e. the occurrence of the complementary event). We will often denote it as A^c.

1.3 Frequentist interpretation

For many phenomena it is intuitive to define the frequency of an event A as:

$$Fr(A) = \frac{\text{number of cases favorable to the event}}{\text{number of possible cases}}$$

Let us consider a coin experiment: say that the probability of heads is 0.5, it means that if we flip the coin many times we expect it to be around 50% of the time the result of the throw will be head.

More in general, by indicating with A an event and with n_A the number of cases favorable to A, the relative frequency of A is equal to:

$$Fr(A) = \frac{n_A}{n}.$$

It then becomes natural to define *probability* as the limit to which the relative frequency of the event tends as the number of experiments increases:

$$p(A) = \lim_{n\to\infty} \frac{n_A}{n}.$$

The definition of relative frequency with finite n is useful to understand the general rules of probability theory. Consider a dice, the space of possibility is $\Omega = \{1, 2, 3, 4, 5, 6\}$, and assume we made $n = 100$ throws and it turns out:

event	1	2	3	4	5	6
freq.	15	12	18	21	16	18

that means that the dice landed on a one 15 times, on a two 12 times, on a three 18 times and so on.

- What is the relative frequency of the event $\{1\}$? By definition it is $n_1/n = 15/100 = 0.15$.
- What is the relative frequency of the event $\{2\}$? It is $n_2/n = 12/100 = 0.12$.
- What is the relative frequency of the event $\{1, 2\} = \{1\} \cup \{2\}$? We can simply sum the frequencies we computed before $n_{\{1,2\}}/n = 0.15 + 0.12 = 0.27$.
- What is the relative frequencies of the event $\{3, 4\}$? We can simply sum the frequencies $n_3/n = 18/100 = 0.18$ and $n_4/n = 21/100 = 0.21$ and so $n_{\{3,4\}}/n = 0.18 + 0.21 = 0.39$.
- What is the relative frequency of the event $\{1, 2, 3, 4\} = \{1, 2\} \cup \{3, 4\}$? We can again sum $n_{\{1,2\}}/n$ and $n_{\{3,4\}}/n$ to obtain 0.66.

- What is the relative frequency of the event $\{even\ number\}$? It is $n_{\{2,4,6\}}/n = 0.52$
- What is the relative frequency of the *certain event* $\{\Omega\}$? It is $n/n = 100/100 = 1$.
- What is the relative frequency of the event $\{1,2\} \cup \{2,3\}$? The union of the two sets is $\{1,2\} \cup \{2,3\} = \{1,2,3\}$ and so $n_{\{1,2,3\}}/n = 0.45$. Note that we do not obtain the same result if we sum $n_{\{1,2\}}/n = 0.27$ and $n_{\{2,3\}}/n = 0.30$, because in this way we count the event 2 twice (that is, the events $\{1,2\}$ and $\{2,3\}$ are not mutually exclusive), we can recover the correct frequency as $n_{\{1,2\}}/n + n_{\{2,3\}}/n - n_{\{2\}}/n = 0.57 - 0.12 = 0.45$.

By answering the above questions, we have discovered the basic rules of probability calculus:

1. The probability of an event is a number between 0 and 1;
2. The probability of the certain event Ω is equal to 1;
3. The probability of two *mutually exclusive* events, that is two events that cannot occur simultaneously, is equal to the sum of the probabilities of the two events.

1.4 Rules of probability

We now introduce a notation to express these rules more compactly. We will use

- lower case letters, for example $x, y, z, \ldots$, to denote experiments;
- $\Omega_x, \Omega_y, \Omega_z$ to denote respectively the possibility space of the experiment $x, y, z, \ldots$;
- $\mathsf{x}, \mathsf{y}, \mathsf{z}, \ldots$ to denote respectively the outcome of the experiment $x, y, z, \ldots$.

In this way, the sentence "the probability that the result of a dice roll is face 2 is equal to 0.2" can be expressed mathematically as:

$$p(x = 2) = 0.2,$$

where we have used x to denote the dice roll experiment. More abstractly we can talk about "the probability that the result of a dice roll is face x" as

$$p(x = \mathsf{x}),$$

where x can be either 1 or 2 or 3 etc., that is $\mathsf{x} \in \Omega_x = \{1, 2, 3, 4, 5, 6\}$ (the symbol $\in$ means "belongs to"). We can also consider non-elementary events, for instance

$$p(x \in \{2, 4, 6\}),$$

that is the probability that the outcome is "even", that we can indicate more abstractly as:

$$p(x \in A),$$

where A is any subset of $\Omega_x = \{1, 2, 3, 4, 5, 6\}$.

Using this notation, we can define the rules of probability theory we derived previously:[3]

Definition: Rules of probability theory:

(A.1) $p : \Omega \to [0, 1]$;

(A.2) $p(\Omega) = 1$.

(A.3) Given m events $A_1, \ldots, A_m$ in Ω such that $A_i \cap A_j = \emptyset$ for all $i \neq j = 1, \ldots, m$ one has that:

$$p\left(x \in \bigcup_{i=1}^{m} A_i\right) = \sum_{i=1}^{m} p\left(x \in A_i\right)$$

[3] In (A.3), the symbol $\sum_{i=1}^{m}$ denotes a sum indexed by i, for instance $\sum_{i=1}^{3} i = 1 + 2 + 3 = 6$ and $\sum_{i=1}^{3} i^2 = 1 + 4 + 9 = 14$. In particular, $\sum_{i=1}^{m} p\left(x \in A_i\right) = p(x \in A_1) + p(x \in A_2) + \cdots + p(x \in A_m)$.

A.1 says that probability p is a function from Ω to the interval $[0, 1]$. A.2 says that the probability of the certain event Ω is one and A.3 means that, provided that the events are mutually exclusive,

$$p(x \in A_1 \cup A_2 \cup \cdots \cup A_m) = p(x \in A_1) + p(x \in A_2) + \cdots + p(x \in A_m).$$

Example: Consider again a dice roll and assume that the probability of the six elementary events are

event	1	2	3	4	5	6
prob.	0.15	0.12	0.18	0.21	0.16	0.18

What is the probability that the result of a dice roll is an "even number"? Answer:

$$\begin{aligned} p(x \in \{2, 4, 6\}) &= \sum_{\mathsf{x} \in \{2,4,6\}} p(x = \mathsf{x}) \\ &= p(x = 2) + p(x = 4) + p(x = 6) = 0.48, \end{aligned}$$

we have used rule A.3 to derive it.

From the rules A.1–A.3, we can derive these rules.[4]

Result:

1. $p(x \in \emptyset) = 0$, where $\emptyset$ denotes the empty-set;
2. $p(x \in A^c) = 1 - p(x \in A)$, where $A^c = \Omega \backslash A$ denotes the complementary event of A.
3. Given two events that are not mutually exclusive (that is $A \cap B \neq \emptyset$) we have that

$$p(x \in A \cup B) = p(x \in A) + p(x \in B) - p(x \in A \cap B). \quad (1.1)$$

Example: Consider again the dice roll example:

event	1	2	3	4	5	6
prob.	0.15	0.12	0.18	0.21	0.16	0.18

What is the probability that the result of a dice roll is an "odd number"?

$$p(x \in \{1,3,5\}) = 1 - p(x \in \{2,4,6\}) = 0.52,$$

We can also compute:

$$p(x \in \{1,2\}) = p(x = 1) + p(x = 2) = 0.27,$$

and

$$p(x \in \{2,3\}) = p(x = 2) + p(x = 3) = 0.30.$$

However, note that

$$p(x \in \{1,2,3\}) \neq p(x \in \{1,2\}) + p(x \in \{2,3\}).$$

In fact in this case we cannot apply A.3 because the events $\{1,2\}$ and $\{2,3\}$ are not mutually exclusive: their intersection $\{1,2\} \cap \{2,3\} = \{2\}$ is not the empty-set. To obtain the correct probability we must apply equation (1.1):

$$\begin{aligned} p(x \in \{1,2,3\}) &= p(x \in \{1,2\}) + p(x \in \{2,3\}) - p(x \in \{1,2\} \cap \{2,3\}), \\ &= p(x \in \{1,2\}) + p(x \in \{2,3\}) - p(x = 2), \end{aligned}$$

in this way we count the probability of 2 only once.

Note that

$$x \in A, \quad x \in B, \quad x \in A^c$$

are events: they can be **true** or **false**.

[4] For instance, to prove that $p(\emptyset) = 0$, first note that $\Omega \cup \emptyset = \Omega$ and $\Omega \cap \emptyset = \emptyset$, therefore the events are mutually exclusive and from rule (A.3) follows that $1 = 1 + p(\emptyset)$ and so $p(\emptyset) = 0$.

Hence, we may want to compute

$$p(x \in A \text{ and } x \in B),\ p(x \in A \text{ or } x \in B),\ p(x \text{ not in } A)$$

From elementary Boolean logic and set theory, we can derive that

$$\begin{aligned} p(x \in A \text{ and } x \in B) &= p(x \in A \cap B) \\ p(x \in A \text{ or } x \in B) &= p(x \in A \cup B) \\ p(x \notin A) &= p(x \in A^c) \end{aligned}$$

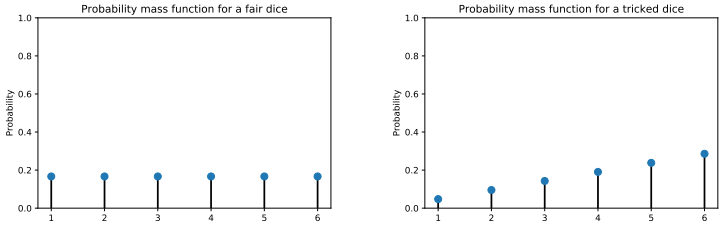
Note that, $p(x \in A \text{ and } x \in B)$ is often denoted as $p(x \in A, x \in B)$.

1.4.1 Probability mass function

A probability function can be fully specified by a discrete list of the probabilities of the elementary events, called *Probability Mass Function* (PMF).

A PMF is the list of all probabilities $p(x = \mathsf{x})$ for each element $\mathsf{x} \in \Omega$.

We can plot it, as shown in the example:



where the left plot is the PMF of a fair dice ($p(x = \mathsf{x}) = 1/6$ for all $\mathsf{x} \in \{1, 2, 3, 4, 5, 6\}$) and the right plot is the PMF of some unfair dice.

Given the PMF, we can compute the probability of any other (non-elementary) event, for instance

$$p(x \in \{2, 4, 6\}) = p(x = 2) + p(x = 4) + p(x = 6),$$

that is equal to $1/6 + 1/6 + 1/6 = 1/2$ for a fair dice.

1.5 Conditional probability

If we know that a certain event has occurred, does this change the probability we assign to another event? Suppose we have been

told that A occurred, so any thing outside of A is not possible. We just have to consider the event A. The space of possibilities Ω has been reduced to A, only parts of any other event B which are now relevant are those contained in A. Since A occurred, the total probability in the possibility space reduced to A must be 1. The conditional probability of B given A is defined as the probability of the parts of B which are also in A, multiplied by the scale factor $1/p(A)$.

Definition: The probability of event B conditioned on knowing event A (or more shortly, the probability of B given A) is defined as

$$p(x \in B | x \in A) = \frac{p(x \in A \cap B)}{p(x \in A)} \tag{1.2}$$

this equation is also called **Bayes' Rule**.

We read it as "the probability of B given A" or " the probability of B conditioned to A". Note: $p(x \in B | x \in A)$ is proportional to the joint probability of the two events, $p(A \cap B)$, but rescaled by the probability of A which takes into account that the possibility space is now reduced to A.

Example: Consider a fair dice:

1. What is the probability that 1 came out given the result is an odd number?
2. What is the probability that 1 or 3 came out given the result is an odd number?
3. What is the probability that 1 or 3 or 5 came out given the result is an odd number?

We can apply Bayes' rule. The answers are:

$$\begin{aligned} p(x = 1 | x \text{ is odd}) &= \frac{p(x \in \{1\} \cap \{1,3,5\})}{p(x \in \{1,3,5\})} \\ &= \frac{p(x = 1)}{p(x \in \{1,3,5\})} = \frac{1/6}{1/2} = 1/3 \end{aligned}$$

$$\begin{aligned} p(x \in \{1,3\} | x \text{ is odd}) &= \frac{p(x \in \{1,3\} \cap \{1,3,5\})}{p(x \in \{1,3,5\})} \\ &= \frac{p(x \in \{1,3\})}{p(x \in \{1,3,5\})} = \frac{2/6}{1/2} = 2/3 \end{aligned}$$

and

$$p(x \in \{1,3,5\} | x \text{ is odd}) = \frac{p(x \in \{1,3,5\} \cap \{1,3,5\})}{p(x \in \{1,3,5\})} = \frac{p(x \in \{1,3,5\})}{p(x \in \{1,3,5\})} = \frac{1/2}{1/2} = 1$$

This shows that conditional probability satisfies the rules of probability:

Definition:

1. $p(\cdot | x \in B) \in [0,1]$;

2. $p(x \in B | x \in B) = 1$.

3. Given m events $A_1, \ldots, A_m$ in Ω such that $A_i \cap A_j = \emptyset$ for all $i \neq j = 1, \ldots, m$ one has that:

$$p\left(x \in \bigcup_{i=1}^{m} A_i | x \in B\right) = \sum_{i=1}^{m} p\left(x \in A_i | x \in B\right)$$

Example: Consider a 52 cards deck. It consist of 4 suits: hearts, diamonds, spades and clubs. Each suit further contains 13 cards: 10 ace cards (A to 10) and 3 picture cards: Jack, Queen, and King.

Note that

$$p(hearts | queen) = \frac{1}{4}$$

while

$$p(queen | hearts) = \frac{1}{13}$$

In general, $p(A|B) \neq p(B|A)$. However, $p(A|B)$ and $p(B|A)$ seem to be related in some way. Bayes' rule tells us how as we will see in Section 1.6.1.

Note that, $p(x \in A | x \in B)$ should not be interpreted as 'Given the event $x \in B$ has occurred, $p(x \in A | x \in B)$ is the probability of the event $x \in A$ occurring'. The correct interpretation should be '$p(x \in A | x \in B)$' is the probability of x being in A under the **constraint** that x is in B.

1.6 Two or more variables

Now we will consider the case we have multivariate experiments, that is:

- we roll two dice
- we roll three coins
- we roll a dice and a coin...

For instance consider the roll of a red and a blue fair dice and denote with z the possible outcome:

- the space of possibility is $\Omega_z = \{(1,1),(1,2),(1,3),(1,4),(1,5),\ldots,(6,4),(6,5),(6,6)\}$ (it has 36 elements);
- $p(z=(1,1)) = p(z=(1,2)) = \cdots = p(z=(6,6)) = \frac{1}{36}$
- $\sum_{\mathsf{z}\in\Omega_z} p(z=\mathsf{z}) = 1$

Note that, in this case, we cannot talk about "the outcome of the red dice" because for instance in the pair $(1,2)$ we do not know if 1 is for red or blue.

To solve this problem, we can introduce two variables

- x denotes the red dice and so $\Omega_x = \{1,2,3,4,5,6\}$;
- y denotes the blue dice and so $\Omega_y = \{1,2,3,4,5,6\}$.

We can then define $z = (x,y)$ (all possible pairs, but now we know that $(1,2)$ means 1 on red dice and 2 on blue dice) and define Ω_z as $\Omega_x \times \Omega_y$, where $\times$ denotes the Cartesian product.[5]

Therefore, we can equivalently denote

$$p(z=(1,2)) \text{ as } p(x=1,y=2)$$

Definition: Given two (or more) variables $p(x=\mathsf{x},y=\mathsf{y})$ is called the **joint probability** of x and y

Definition: Given a joint probability $p(x,y)$ the **marginal probability** of x is defined by

$$p(x=\mathsf{x}) = \sum_{\mathsf{y}\in\Omega_y} p(x=\mathsf{x},y=\mathsf{y}) \tag{1.3}$$

and the **marginal probability** of y is defined by

$$p(y=\mathsf{y}) = \sum_{\mathsf{x}\in\Omega_x} p(x=\mathsf{x},y=\mathsf{y})/ \tag{1.4}$$

[5] The Cartesian product of two sets A and B, denoted $A \times B$, is the set of all possible pairs where the elements of A are first and the elements of B are second. That is, $A \times B = \{(a,b) : a \in A \text{ and } b \in B\}$.

Example: For two fair dice, the joint probability is

$$p(x=1,y=1)=p(x=1,y=2)=\cdots=p(x=6,y=6)=\frac{1}{36}$$

How can we compute $p(x=1)$? $p(x=1)$ is called *marginal probability* and can be obtained as

$$\begin{aligned}p(x=1)&=\sum_{\mathsf{y}\in\{1,2,3,4,5,6\}}p(x=1,y=\mathsf{y})\\&=p(x=1,y=1)+p(x=1,y=2)\\&+p(x=1,y=3)+p(x=1,y=4)\\&+p(x=1,y=5)+p(x=1,y=6)=\frac{1}{6}.\end{aligned}$$

Example: Now consider an experiment consisting on tossing a coin x and a dice y, the space of possibility is

$$\begin{aligned}\Omega&=\{H,T\}\times\{1,2,3,4,5,6\}\\&=\{(H,1),(H,2),(H,3),(H,4),(H,5),(H,6)\\&(T,1),(T,2),(T,3),(T,4),(T,5),(T,6)\}.\end{aligned}$$

We can assign probabilities to these events, for instance:

$$\begin{gathered}p(H,1)=0.1,\ p(H,2)=0.05,\ p(H,3)=0.1,\ p(H,4)=0.05,\\p(H,5)=0.1,\ p(H,6)=0.2,\ p(T,1)=0.1,\ p(T,2)=0.05,\\p(T,3)=0.1,\ p(T,4)=0.05,\ p(T,5)=0.1,\ p(T,6)=0\end{gathered}$$

This is the joint probability. What is $p(x=H)$? $p(x=H)$ is called *marginal probability* and can be obtained as

$$\begin{aligned}p(x=H)&=\sum_{\mathsf{y}\in\{1,2,3,4,5,6\}}p(x=H,y=\mathsf{y})\\&=p(x=H,y=1)+p(x=H,y=2)\\&+p(x=H,y=3)+p(x=H,y=4)\\&+p(x=H,y=5)+p(x=H,y=6)=0.6.\end{aligned}$$

and so $p(x=T)=0.4$. What is $p(y=1)$?

$$\begin{aligned}p(y=1)&=\sum_{\mathsf{x}\in\{H,T\}}p(x=\mathsf{x},y=1)\\&=p(x=H,y=1)+p(x=H,y=1)=0.2.\end{aligned}$$

Definition: Two variables x, y are said to be **independent** if

$$p(x = \mathsf{x}, y = \mathsf{y}) = p(x = \mathsf{x})p(y = \mathsf{y}) \tag{1.5}$$

If this is not true, they are called **dependent**.

Sometimes there is confusion with the concept of independence also because the word independence has many meanings. The primary meaning of independence is that something does not depend on something else. This is what we mean when we talk about independent variables. Indeed, variables x and y are said to be independent if knowing the value of one variable gives no extra information about the other. This definition can be better expressed using conditioning.

Definition: Variables x and y are said to be independent if

$$p(y = \mathsf{y}|x = \mathsf{x}) = p(y = \mathsf{y}) \tag{1.6}$$

or, equivalently, if

$$p(y = \mathsf{y}|x = \mathsf{x}) = p(y = \mathsf{y}) \tag{1.7}$$

The three conditions for independence in Equations (3.1), (1.6) and (1.7) are equivalent. That means we can use any of them to prove that two variables are independent.

Example: Let us go back to the experiment consisting on tossing a coin x and a dice y, the joint probability was:

$$p(H,1) = 0.1,\ p(H,2) = 0.05,\ p(H,3) = 0.1,\ p(H,4) = 0.05,$$
$$p(H,5) = 0.1,\ p(H,6) = 0.2,\ p(T,1) = 0.1,\ p(T,2) = 0.05,$$
$$p(T,3) = 0.1,\ p(T,4) = 0.05,\ p(T,5) = 0.1,\ p(T,6) = 0$$

Are the variables x, 'coin toss', and y 'dice toll' independent? To verify that, we can use any of the definitions in Equation (3.1), (1.6) and (1.7). Let us use (3.1), first we need to compute the marginal probabilities $p(x = H), p(x = T)$ and $p(y = 1), p(y = 2), \ldots, p(y = 6)$. We have already seen that

$$p(x = H) = 0.6, \quad p(x = T) = 0.4, \quad p(y = 1) = 0.2.$$

Note that

$$\begin{aligned}
p(y=2) &= \sum_{\mathsf{x}\in\{H,T\}} p(x=\mathsf{x}, y=2) \\
&= p(x=H, y=2) + p(x=H, y=2) = 0.1 \\
p(y=3) &= \sum_{\mathsf{x}\in\{H,T\}} p(x=\mathsf{x}, y=3) \\
&= p(x=H, y=3) + p(x=H, y=3) = 0.2 \\
p(y=4) &= \sum_{\mathsf{x}\in\{H,T\}} p(x=\mathsf{x}, y=4) \\
&= p(x=H, y=4) + p(x=H, y=4) = 0.1. \\
p(y=5) &= \sum_{\mathsf{x}\in\{H,T\}} p(x=\mathsf{x}, y=3) \\
&= p(x=H, y=5) + p(x=H, y=5) = 0.2. \\
p(y=6) &= \sum_{\mathsf{x}\in\{H,T\}} p(x=\mathsf{x}, y=3) \\
&= p(x=H, y=6) + p(x=H, y=6) = 0.2.
\end{aligned}$$

We need to verify if

$$p(x=\mathsf{x}, y=\mathsf{y}) \stackrel{?}{=} p(x=\mathsf{x})p(y=\mathsf{y})$$

for all possible cases. Note that:

$$p(x=H, y=1) = 0.1 \neq p(x=H)p(y=1) = 0.6 \cdot 0.2 = 0.12$$

and, therefore, the variables x, y are dependent. That means that the result of the dice roll in somehow depends on the result of the coin toss (and vice versa).

1.6.1 More about Bayes' rule

In Equation (1.2), we have defined Bayes' rule. We can formally change the role of A and B. The probability of A conditional on B:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Definition: From the conditioning rule it follows that:

$$P(A \cap B) = P(A|B)P(B)$$

which is known as the **probability multiplication rule**.

We can define conditioning also if the conditioning event is B^c (complementary to B):

$$P(A|B^c) = \frac{P(A \cap B^c)}{P(B^c)}$$

and so

$$P(A \cap B^c) = P(A|B^c)P(B^c)$$

From the definition

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

Since $A = (A \cap B) \cup (A \cap B^c)$ follows that

$$P(A) = P(A \cap B) + P(A \cap B^c)$$

and therefore replacing the second formula in the first we have:

$$P(B|A) = \frac{P(A \cap B)}{P(A \cap B) + P(A \cap B^c)}$$

Finally applying the multiplication rule we have that

$$P(B|A) = \frac{P(A|B)P(B)}{P(A|B)P(B) + P(A|B^c)P(B^c)}$$

Result: Given two events A and B the probability of B given A is equal to:

$$p(B|A) = \frac{p(A \cap B)}{p(A)} = \frac{p(A|B)p(B)}{p(A)} = \frac{p(A|B)p(B)}{p(A|B)p(B) + p(A|B^c)p(B^c)} \tag{1.8}$$

Summarizing: Equation (1.8) is a redefinition of the conditional probability:

1. the probability of A is found as the sum of the probability of the two disjoint events $A \cap B$ and $A \cap B^c$.
2. Each of the joint probabilities is found by the multiplication rule.

Note that the *union* of B and B^c is the whole space of possibilities. We say that B and B^c constitute a *partition* of the space of possibilities.

Example: Let us go back to the 52 card deck. Note that $p(hearts|queen) = \frac{1}{4}$ and $p(queen|hearts) = \frac{1}{13}$. We can exploit Equation (1.8) to understand the relationship between these two probabilities.

$$\begin{aligned} p(queen|hearts) &= \frac{p(hearts|queen)p(queen)}{p(hearts)} \\ &= \frac{p(hearts|queen)p(queen)}{p(hearts|queen)p(queen) + p(hearts|not\ queen)p(not\ queen)} \\ &= \frac{\frac{1}{4}\frac{4}{52}}{\frac{1}{4}\frac{4}{52} + \frac{12}{48}\frac{48}{52}} = \frac{1}{13}. \end{aligned}$$

1.7 Inference

Equation (1.8) is very important. Much of science (and machine learning) deals with problems of the form: tell me something about the variable θ given that I have observed some *data* and have some knowledge of the underlying data generating mechanism. Our interest is then the quantity

$$p(\theta|\text{data}) = \frac{p(\text{data}|\theta)p(\theta)}{p(\text{data})}$$

This shows how from a forward or *generative model* $p(\text{data}|\theta)$ of the dataset, and coupled with a *prior* belief $p(\theta)$ about which variable values are appropriate, we can infer the *posterior* distribution $p(\theta|\text{data})$ of the variable in light of the observed data.

Bayes' rule tells us how to update our prior beliefs about the variable θ in light of the data to a posterior belief:

$$\underbrace{p(\theta|\text{data})}_{\text{posterior}} = \frac{\overbrace{p(\text{data}|\theta)}^{\text{likelihood}}\overbrace{p(\theta)}^{\text{prior}}}{\underbrace{p(\text{data})}_{\text{evidence}}}$$

We consider the following example.[6]

Example: A particular test for whether someone has been using cannabis is 90% sensitive and 80% specific, meaning it leads to 90% true positive results (correct identification of drug use) for cannabis users and 80% true negative results (correct identification of non-use) for non-users, but also generates 20% false positives for non-

[6] This example is from https://en.wikipedia.org/wiki/Bayes%27_theorem#Drug_testing.

users. Assuming 5% of people use cannabis, what is the probability that a random person who tests positive is really a cannabis user?

Let $p(\text{User} \mid \text{Positive})$ mean *the probability that someone is a cannabis user given that they test positive (the result of the test is our data)*. Then we can write:

$$p(\text{User} \mid \text{Positive}) = \frac{p(\text{Positive} \mid \text{User})p(\text{User})}{p(\text{Positive})}$$
$$= \frac{p(\text{Positive} \mid \text{User})p(\text{User})}{p(\text{Positive} \mid \text{User})p(\text{User}) + p(\text{Positive} \mid \text{Non-user})p(\text{Non-user})}$$
$$= \frac{0.90 \times 0.05}{0.90 \times 0.05 + 0.20 \times 0.95} \approx 19\%$$

Even if someone tests positive, the probability they are a cannabis user is only 19%, because in this group only 5% of people are users, most positives are false positives coming from the remaining 95%.

$p(\text{User})$ is the prior probability (that is before seeing the test result) that a random person is a cannabis user. $p(\text{Positive} \mid \text{User})$ is the likelihood, that is the probability that a test is positive given that the person is an User. $p(\text{Positive})$ is the evidence, the probability of observing a positive test (not matter if the person is a User or not).

1.7.1 Case study: spam filters

Building a simple spam filter for emails is a good way to understand probability theory and also understand the basic probability concepts used in Natural Language Processing (NLP).

The question we aim to answer: is an email that includes the words "Bye" and "Won" Spam? We consider a simplified spam-filter. We will only use these two words, "Bye", "Won", to assess if an email is spam (the email can include other words).

What is the possibility space? Any email can be spam or not, can include "Bye" or not, can include "Won" or not. Therefore, we have three variables s, b, w whose possibility space are:

- $\Omega_s \in \{0, 1\}$ where 0 means not-spam and 1 means spam.
- $\Omega_b \in \{0, 1\}$ where 0 means that the email does not include "Bye" and 1 means it includes "Bye".
- $\Omega_w \in \{0, 1\}$ where 0 means that the email does not include "Won" and 1 means it includes "Won".

The joint space of possibility is therefore

$$\Omega = \Omega_s \times \Omega_b \times \Omega_w,$$

that has $2^3 = 8$ elements accounting for all possible cases: $(0, 0, 0)$ the email is not spam and does not include "Bye" or "Won"; $(0, 0, 1)$ the email is not spam, it does not include "Bye" but includes "Won" and so on.

Assume we know the following **joint probability**:

s	b	w	$Prob$
0	0	0	0.168
1	0	0	0.032
0	1	0	0.252
1	1	0	0.048
0	0	1	0.072
1	0	1	0.128
0	1	1	0.108
1	1	1	0.192

where the first row means

$$p(s = 0, b = 0, s = 0) = 0.168$$

and similarly for the other 7 rows. This table defines the joint probability (we also call it *joint probability distribution*) of the three variables. We will see later on how to derive this table from data (a dataset of emails), but for the moment we assume that we know these probabilities.

Why do we need probability theory? Because we want to answer questions like:

1. What is the probability that an email is Spam? (no matter which words it includes)
2. What is the probability that an email is Spam and does not include "Won" ?
3. What is a probability that the email includes "Bye" given that the email is Spam?
4. What is the probability that an email is Spam given it includes "Won" but not "Bye"?
5. ...

These are all important questions when we design a spam filter. We will use these questions to review the probability rules learned in this chapter.

Let us start with the first question: what is the probability that an email is spam? We want to know $p(s = 1)$, we do not have it but we can compute it from the joint probability distribution by marginalising out the other variables:

$$\begin{aligned} p(s = 1) &= \sum_{\mathsf{b},\mathsf{w} \in \{0,1\}} p(s = 1, b = \mathsf{b}, w = \mathsf{w}) \\ &= p(s = 1, b = 0, w = 0) + p(s = 1, b = 0, w = 1) \\ &+ p(s = 1, b = 1, w = 0) + p(s = 1, b = 1, w = 1) \\ &= 0.032 + 0.128 + 0.048 + 0.192 = 0.4 \end{aligned}$$

In practice, we sum all the rows in the table that include the instance $s = 0$ (colored rows in the following table):

s	b	w	$Prob$
0	0	0	0.168
1	0	0	0.032
0	1	0	0.252
1	1	0	0.048
0	0	1	0.072
1	0	1	0.128
0	1	1	0.108
1	1	1	0.192

The next question is: what is the probability that an email is spam and does not include "Won" ? We want $p(s = 1 \text{ and } w = 0) = p(s = 1, w = 0)$, we do not have it but we can compute it by marginalising out the other variable:

$$\begin{aligned} p(s = 1, w = 0) &= \sum_{b \in \{0,1\}} p(s = 1, b = \mathsf{b}, w = 0) \\ &= p(s = 1, b = 0, w = 0) + p(s = 1, b = 1, w = 0) \\ &= 0.032 + 0.048 = 0.08 \end{aligned}$$

Again we need to sum all the rows that include the instance $s = 1, w = 0$ (colored rows):

s	b	w	$Prob$
0	0	0	0.168
1	0	0	0.032
0	1	0	0.252
1	1	0	0.048
0	0	1	0.072
1	0	1	0.128
0	1	1	0.108
1	1	1	0.192

Next question: what is the probability that an email includes the word *Bye* and *Won* given that is spam? We want to know $p(b = 1, w = 1|s = 1)$, we do not have it but we can compute it by applying Bayes' rule (conditional probability):

$$p(b = 1, w = 1|s = 1) = \frac{p(b = 1, w = 1, s = 1)}{p(s = 1)} = \frac{0.192}{0.4} = 0.48$$

We saw in Section 1.5 that the conditional probability of $b = 1, w = 1$ given $s = 1$ is defined as the probability of the parts of $b = 1, w = 1$ which are also in $s = 1$, multiplied by the scale factor $1/p(s = 1)$. In the following table, "the parts of $b = 1, w = 1$ which are also in $s = 1$" is the last row (red) and, therefore, we divide $p(b = 1, w = 1, s = 1)$ by $1/p(s = 1)$ (we can obtain the marginal probability $p(s = 1)$ by summing all colored rows in the below table).

s	b	w	$Prob$
0	0	0	0.168
1	0	0	0.032
0	1	0	0.252
1	1	0	0.048
0	0	1	0.072
1	0	1	0.128
0	1	1	0.108
1	1	1	0.192

There are alternative formulas we could use to obtain the same result. They are equivalent. We can choose the one that is more convenient to use given the question we aim to answer to. Note that,

$$\begin{aligned} p(b = 1, w = 1|s = 1) &= \frac{p(b = 1, w = 1, s = 1)}{p(s = 1)} \\ &= \frac{p(b = 1|w = 1, s = 1)p(w = 1, s = 1)}{p(s = 1)} \\ &= \frac{p(b = 1|w = 1, s = 1)p(w = 1|s = 1)p(s = 1)}{p(s = 1)} \\ &= p(b = 1|w = 1, s = 1)p(w = 1|s = 1) \end{aligned}$$

or, other equivalent formulas:

$$\begin{aligned} p(b=1,w=1|s=1) &= \frac{p(b=1,w=1,s=1)}{p(s=1)} \\ &= \frac{p(w=1|b=1,s=1)p(b=1,s=1)}{p(s=1)} \\ &= \frac{p(w=1|b=1,s=1)p(b=1|s=1)p(s=1)}{p(s=1)} \\ &= p(w=1|b=1,s=1)p(b=1|s=1) \end{aligned}$$

These formulas can be obtained by simply applying the laws of probability you learned in this Chapter.

Next questions: what is $p(b=1|s=1)$: probability that there is "Bye" given the email is spam?

$$\begin{aligned} p(b=1|s=1) &= \sum_{w\in\{0,1\}} p(b=1,w=w|s=1) \\ &= \sum_{w\in\{0,1\}} \tfrac{p(b=1,w=w,s=1)}{p(s=1)} \\ &= \tfrac{p(s=1,b=1,w=0)}{p(s=1)} + \tfrac{p(s=1,b=1,w=1)}{p(s=1)} \\ &= \tfrac{0.048}{0.4} + \tfrac{0.192}{0.4} = \tfrac{0.24}{0.4} = 0.6 \end{aligned}$$

and so $p(b=0|s=1) = 1 - p(b=1|s=1) = 0.4$.

s	b	w	$Prob$
0	0	0	0.168
1	0	0	0.032
0	1	0	0.252
1	1	0	0.048
0	0	1	0.072
1	0	1	0.128
0	1	1	0.108
1	1	1	0.192

What is $p(w=1|s=1)$: probability that there is the word "Won" given the email is spam?

$$\begin{aligned} p(w=1|s=1) &= \sum_{b\in\{0,1\}} p(b=b,w=1|s=1) = \sum_{b\in\{0,1\}} \tfrac{p(b=b,w=1,s=1)}{p(s=1)} \\ &= \tfrac{p(s=1,b=0,w=1)}{p(s=1)} + \tfrac{p(s=1,b=1,w=1)}{p(s=1)} \\ &= \tfrac{0.128}{0.4} + \tfrac{0.192}{0.4} = \tfrac{0.32}{0.4} = 0.8 \end{aligned}$$

and so $p(w=0|s=1) = 1 - p(w=1|s=1) = 0.2$.

s	b	w	$Prob$
0	0	0	0.168
1	0	0	0.032
0	1	0	0.252
1	1	0	0.048
0	0	1	0.072
1	0	1	0.128
0	1	1	0.108
1	1	1	0.192

What is $p(w=1|s=0)$: probability that there is the word "Won" given the email is not spam?

$$\begin{aligned} p(w=1|s=0) &= \sum_{b\in\{0,1\}} p(b=b, w=1|s=0) = \sum_{b\in\{0,1\}} \tfrac{p(b=b,w=1,s=0)}{p(s=0)} \\ &= \tfrac{p(s=0,b=0,w=1)}{p(s=0)} + \tfrac{p(s=0,b=1,w=1)}{p(s=0)} \\ &= \tfrac{0.072}{0.6} + \tfrac{0.108}{0.6} = \tfrac{0.18}{0.6} = 0.3 \end{aligned}$$

and so $p(w=0|s=0) = 1 - p(w=1|s=0) = 0.7$.

s	b	w	$Prob$
0	0	0	0.168
1	0	0	0.032
0	1	0	0.252
1	1	0	0.048
0	0	1	0.072
1	0	1	0.128
0	1	1	0.108
1	1	1	0.192

What is the probability that there is "Won" given the email is spam and includes "Bye"?

$$\begin{aligned} p(w=1|b=1,s=1) &= \tfrac{p(w=1,b=1,s=1)}{p(b=1,s=1)} \\ &= \tfrac{p(w=1,b=1,s=1)}{p(b=1|s=1)p(s=1)} = \tfrac{0.192}{0.6\cdot 0.4} = 0.8 \end{aligned}$$

and so $p(w=0|b=1,s=1) = 1 - p(w=1|b=1,s=1) = 0.2$.

What is the probability that there is "Won" given the email is spam and does not include "Bye"?

$$\begin{aligned} p(w=1|b=0,s=1) &= \tfrac{p(w=1,b=0,s=1)}{p(b=0,s=1)} \\ &= \tfrac{p(w=1,b=0,s=1)}{p(b=0|s=1)p(s=1)} = \tfrac{0.128}{0.4\cdot 0.4} = 0.8 \end{aligned}$$

and so $p(w = 0|b = 0, s = 1) = 1 - p(w = 1|b = 0, s = 1) = 0.2$.

We now check conditional independence. Are the variables b and w conditional independent given s?

Definition: Two variables x, y are said to be **conditionally independent** given z if

$$p(x = \mathsf{x}, y = \mathsf{y} \mid z = \mathsf{z}) = p(x = \mathsf{x} \mid z = \mathsf{z})p(y = \mathsf{y} \mid z = \mathsf{z}) \quad (1.9)$$

for all values $\mathsf{x}, \mathsf{y}, \mathsf{z}$. Equivalently, if

$$p(x = \mathsf{x} \mid y = \mathsf{y}, z = \mathsf{z}) = p(x = \mathsf{x} \mid z = \mathsf{z}) \quad (1.10)$$

or

$$p(y = \mathsf{y} \mid x = \mathsf{x}z = \mathsf{z}) = p(y = \mathsf{y} \mid z = \mathsf{z}) \quad (1.11)$$

These three definitions are all equivalent. If this is not true, they are called **conditionally dependent**.

We have reported the joint probability distribution hereafter for convenience:

s	b	w	$Prob$
0	0	0	0.168
1	0	0	0.032
0	1	0	0.252
1	1	0	0.048
0	0	1	0.072
1	0	1	0.128
0	1	1	0.108
1	1	1	0.192

The two variables are conditional independent given s if

$$p(w = w|b = b, s = s) = p(w = w|s = s) \text{ or}$$
$$p(b = b|w = w, s = s) = p(b = b|s = s)$$

for all $w, b, s \in \{0, 1\}$. If we choose the definition at the top, we must verify that the probability in the following two columns are

equal:

$$
\begin{aligned}
&p(w=1|b=1,s=1)=0.8, \quad p(w=1|s=1)=0.8,\\
&p(w=0|b=1,s=1)=0.2, \quad p(w=0|s=1)=0.2,\\
&p(w=1|b=0,s=1)=0.8, \quad p(w=1|s=1)=0.8,\\
&p(w=0|b=0,s=1)=0.2, \quad p(w=0|s=1)=0.2,\\
&p(w=1|b=1,s=0)=0.3, \quad p(w=1|s=0)=0.3,\\
&p(w=0|b=1,s=0)=0.7, \quad p(w=0|s=0)=0.7,\\
&p(w=1|b=0,s=0)=0.3, \quad p(w=1|s=0)=0.3,\\
&p(w=0|b=0,s=0)=0.7, \quad p(w=0|s=0)=0.7
\end{aligned}
$$

Since this is the case, b, w are conditional independent given s.

Are the variables b and s (in)dependent? That is, does the knowledge that an email is spam changes our belief that the email includes or doesn't include the word *Bye*? The joint probability is:

s	b	w	$Prob$
0	0	0	0.168
1	0	0	0.032
0	1	0	0.252
1	1	0	0.048
0	0	1	0.072
1	0	1	0.128
0	1	1	0.108
1	1	1	0.192

we must verify that the probability in the following two columns are equal:

$$
\begin{aligned}
&p(b=1|s=1)=0.6 \quad p(b=1)=0.6\\
&p(b=0|s=1)=0.4 \quad p(b=0)=0.4\\
&p(b=1|s=0)=0.6 \quad p(b=1)=0.6\\
&p(b=0|s=0)=0.4 \quad p(b=0)=0.4
\end{aligned}
$$

Since this is the case, then b, s are independent.

1.7.2 Learning

What is the probability that an email that includes the word "Bye" and "Won" is spam? We have seen that if we know this joint

probability

s	b	w	$Prob$
0	0	0	0.168
1	0	0	0.032
0	1	0	0.252
1	1	0	0.048
0	0	1	0.072
1	0	1	0.128
0	1	1	0.108
1	1	1	0.192

we can compute

$$p(s = s|b = b, w = w)$$

which is what a spam filter computes to make decisions. This probability helps us to filter all the mails which are marked as Spam and then store them in a Spam folder.[7]

[7] For instance, if $p(s = 1|b = b, w = w) > 0.5$ we can conclude that the email is spam and move it to the Spam folder.

The problem is that we do not know the joint distribution:

s	b	w	$Prob$
0	0	0	?
1	0	0	?
0	1	0	?
1	1	0	?
0	0	1	?
1	0	1	?
0	1	1	?
1	1	1	?

but we have past data, that is user-annotated emails as spam or not-spam (the class column in the Table):

Id	Text	Spam
0	Dear John, You won 100 euros!	1
1	Dear Ashley, You won a new car!	1
2	Hi Helen, We won the game yesterday! Bye	0
3	Huge discount for you buy our new clothes. Bye	0
4	Thank you for your offer	1
5	You won the lottery	1
6	Buy our cruise	1
7	Dear Charlie, Your documents are ready. Bye	0

A ML algorithm can utilize such data to learn the "?" probabilities and use them to predict the correct label (spam or not-spam) of unseen new emails.

1.8 Exercises with solutions

Exercise 1) We toss a fair coin 3 times.

a Write down the possibility space, if we record the exact sequences of Heads(= 1) and Tails (= 0)

b Write down the possibility space, if we record only the total number of Heads.

Exercise 2 Assign a probability to all elements (elementary events) of the possibility spaces of Exercise 1.(a) and 1.(b).

Exercise 3 We randomly extract 3 balls from an urn with 365 balls, numbered $1, \ldots, 365$.

a If we put each ball back into the urn before we draw the next, how many possible outcomes of the experiment are there?

b Answer the same question as above, but now if we do not put the balls back.

c Calculate the probability that we draw 3 times the same ball in case (a).

Exercise 4 Consider the experiment of throwing 2 fair dice.

a Find the probability that both dice show the same face.

b Find the same probability, given you know that the sum of the dice is not greater than 4.

Exercise 5 Consider the experiment of throwing 2 dice. Denote with x the outcome for the first dice and y the ourcome for the second dice. Assume that the joint probability mass function of x, y is:

$$p(x = i, y = i) = 0 \text{ and } p(x = i, y = j) = \frac{1}{30}$$

for $i, j = 1, 2, 3, 4, 5, 6$ and $i \neq j$.

a Compute the marginal probability mass functions $p(x)$ and $p(y)$.

b Are the two variables x and y independent?

c By applying Bayes' rule, compute the probability that $x+y \geq 11$ given that you know that x is even.

Exercise 6 In a binary transmission channel, the bit 1 is transmitted with probability 2/3 and the bit 0 with probability 1/3. The channel is noisy so the conditional probability of receiving a 1 when a 1 was sent is 0.95, the conditional probability of receiving a 0 when a 0 was sent is 0.90. Given that a 1 is received, what is the probability that a 1 was transmitted?

Exercise 7 Suppose the probability that we are flu is $p(\text{Flu}) = 0.05$, the probability that we have High-temperature when we are flu is $p(\text{High-temperature}|\text{Flu}) = 0.9$ and the probability that we have a high temperature when we are not flu (false alarm) is $p(\text{High-temperature}|\neg\text{Flu}) = 0.2$, where $\neg$ denotes not.

Assume we have evidence that a person has a high-temperature, what is the probability that this person is Flu?

Exercise 8 Consider the last exercise. We have now a thermometer whose rate of false negative reading is 5% and false positive reading is 14%, that is,

$$p(\text{HighTherm} = True|\text{HighTemp} = True) = 0.95$$
$$p(\text{HighTherm} = True|\text{HighTemp} = False) = 0.15$$

Assume that Flu and HighTherm are independent given that the person has HighTemp.

a The person is Flu and Thermometer suggests HighTemp, what is the probability that the person has a High temperature?

b Thermometer suggests HighTemp, what is the probability of the person being Flu?

1.8.1 Solutions

Exercise 1

(a) $\Omega = \{(0,0,0), ..., (1,1,1)\}$.

(b) $\Omega = \{0,1,2,3\}$.

Exercise 2

(a) $p(x,y,z) = 1/8$ for all $x, y, z \in \{0,1\}$.

(b) $p(0) = 1/8$, $p(1) = 3/8$, $p(2) = 3/8$ and $p(3) = 1/8$

Exercise 3

(a) the number of possible outputs is 365^3.

(b) the number of possible outputs is $365 \cdot 364 \cdot 363$

(c) the probability is $365/365^3 = 1/365^2$

Exercise 4

(a) The probability that both dices show the same face is

$$p(x = 1, y = 1) + p(x = 2, y = 2) + \cdots + p(x = 6, y = 6) = \frac{1}{6}$$

(b) Let A be the event that the dice show the same face, and B the event that the sum is not greater than 4. Then B = {(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (3, 1)}, and $A \cap B$ = {(1, 1), (2, 2)}. Hence, $p(A|B) = 2/6 = 1/3$. We can also solve it by applying Bayes' rule:

$$p(x = i, y = i|x+y \leq 4) = \frac{p(x + y \leq 4|x = i, y = i)p(x = i, y = i)}{p(x + y \leq 4)} = \frac{\frac{2}{6}\frac{1}{36}}{\frac{1}{6}} = \frac{2}{36}$$

Note then

$$p(x + y \leq 4|x = i, y = i) = 1$$

for $i = 1, 2$ and zero otherwise. Therefore

$$p(x = 1, y = 1|x+y \leq 4) = \frac{p(x + y \leq 4|x = 1, y = 1)p(x = 1, y = 1)}{p(x + y \leq 4)} = \frac{\frac{1}{36}}{\frac{1}{6}} = \frac{1}{6}$$

and

$$p(x = 2, y = 2|x+y \leq 4) = \frac{p(x + y \leq 4|x = 1, y = 1)p(x = 1, y = 1)}{p(x + y \leq 4)} = \frac{\frac{1}{36}}{\frac{1}{6}} = \frac{1}{6}$$

and

$$p(x = i, y = i|x + y \leq 4) = 0 \text{ for } i \geq 3$$

Hence,

$$p(x = 1, y = 1|x + y \leq 4) + p(x = 2, y = 2|x + y \leq 4) = \frac{1}{3}$$

Exercise 5

a $p(x = 1) = \sum_{y=1}^{6} p(x = 1, y = y) = \frac{5}{30} = \frac{1}{6}$, similarly $p(x = 2) = p(x = 3) = \cdots = p(x = 6) = 1/6$. Same for $p(y)$.

b By definition, two variables are independent if

$$p(x = \mathsf{x}, y = \mathsf{y}) = p(x = \mathsf{x})p(y = \mathsf{y})$$

for all $\mathsf{x}, \mathsf{y} = 1, 2, \ldots, 6$. In this case,

$$p(x = 1, y = 1) = 0 \neq p(x = 1)p(y = 1) = \frac{1}{36}$$

and, therefore, the variables are dependent. Note in fact that, when $x = i$ then the probability that $y = i$ is zero for $i = 1, 2, \ldots, 6$.

c We apply Bayes' rule:

$$p(x+y \geq 11|x \text{ is even}) = \frac{p(x \text{ is even}|x + y \geq 11)p(x + y \geq 11)}{p(x \text{ is even})}$$

where

$$p(x \text{ is even}) = \frac{1}{2} \text{ we know it from the marginal}$$

and

$$p(x \text{ is even}|x + y \geq 11) = \frac{1}{2}$$

and

$$p(x + y \geq 11) = \frac{2}{30}$$

Therefore, we have that

$$p(x + y \geq 11|x \text{ is even}) = \frac{\frac{2}{30}\frac{1}{2}}{\frac{1}{2}} = \frac{1}{15}$$

Exercise 6 Let B be the event that a 1 was sent, and A the event that a 1 is received. Then, $p(A = 1|b = 1) = 0.95$, and $p(A = 0|b = 0) = 0.90$. Thus, $p(A = 0|b = 1) = 0.05$ and $p(A = 1|B = 0) = 0.10$. Moreover, $p(B) = 2/3$ and $p(b = 0) = 1/3$. By Bayes' rule:

$$\begin{aligned} p(b = 1|A = 1) &= \frac{p(A = 1|b = 1)p(b = 1)}{p(A = 1|b = 0)p(b = 0) + p(A = 1|b = 1)p(b = 1)} \\ &= \frac{0.95 \cdot \frac{2}{3}}{0.10 \cdot \frac{1}{3} + 0.95 \cdot \frac{2}{3}} = 0.95 \end{aligned}$$

Exercise 7 We want to compute $p(\text{Flu}|\text{High-temperature})$ by Bayes' rule

$$p(\text{Flu}|\text{High-temperature}) = \frac{p(\text{High-temperature}|\text{Flu})p(\text{Flu})}{p(\text{High-temperature})}$$

Note that

$$p(\text{High-temperature}) = p(\text{High-temperature}|\text{Flu})p(\text{Flu}) + p(\text{High-temperature}|\neg\text{Flu})p(\neg\text{Flu})$$

Hence

$$p(\text{Flu}|\text{High-temperature}) = \frac{0.9 \cdot 0.05}{0.9 \cdot 0.05 + 0.2 \cdot 0.95} = 0.192$$

Exercise 8

a The person is Flu and Thermometer suggests HighTemp, what is the probability that the person has a High temperature? We aim to compute

$$p(\text{HighTemp}|\text{Flu}, \text{HighTherm})$$

By Bayes' rule:

$$\begin{aligned} p(\text{HighTemp}|\text{Flu}, \text{HighTherm}) &= \frac{p(\text{HighTemp}|\text{Flu})p(\text{HighTherm}|\text{HighTemp})p(\text{Flu})}{p(\text{Flu}, \text{HighTherm})} \\ &= \frac{0.9 \cdot 0.95 \cdot 0.05}{p(\text{Flu}, \text{HighTherm})} \end{aligned}$$

where

$$\begin{aligned} p(\text{Flu}, \text{HighTherm}) &= p(\text{HighTemp}|\text{Flu})p(\text{HighTherm}|\text{HighTemp})p(\text{Flu}) \\ &\quad + p(\neg\text{HighTemp}|\text{Flu})p(\text{HighTherm}|\neg\text{HighTemp})p(\text{Flu}) \\ &= 0.9 \cdot 0.95 \cdot 0.05 + 0.1 \cdot 0.15 \cdot 0.05 \end{aligned}$$

and therefore

$$p(\text{HighTemp}|\text{Flu}, \text{HighTherm}) = \frac{0.9 \cdot 0.95 \cdot 0.05}{0.9 \cdot 0.95 \cdot 0.05 + 0.1 \cdot 0.15 \cdot 0.05} = 0.982$$

b Thermometer suggests HighTmp, what is the probability of the person being Flu?

$$p(\text{Flu}|\text{HighTherm}) = \frac{p(\text{HighTherm}|\text{Flu})p(\text{Flu})}{p(\text{HighTherm})}$$

where

$$\begin{aligned} p(\text{HighTherm}|\text{Flu}) &= p(\text{HighTherm}, \text{HighTemp}|\text{Flu}) + p(\text{HighTherm}, \neg\text{HighTemp}|\text{Flu}) \\ &= p(\text{HighTherm}|\text{HighTemp})p(\text{HighTemp}|\text{Flu}) \\ &\quad + p(\text{HighTherm}|\neg\text{HighTemp})p(\neg\text{HighTemp}|\text{Flu}) \\ &= (0.95 \cdot 0.9 + 0.15 \cdot 0.1) \cdot 0.05 = 0.87 \cdot 0.05 \end{aligned}$$

and

$$\begin{aligned} p(\text{HighTherm}) &= p(\text{HighTherm}|\text{HighTemp})p(\text{HighTemp}|\text{Flu}) \\ &+ p(\text{HighTherm}|\neg\text{HighTemp})p(\neg\text{HighTemp}|\text{Flu}) \\ &+ p(\text{HighTherm}|\text{HighTemp})p(\text{HighTemp}|\neg\text{Flu}) \\ &+ p(\text{HighTherm}|\neg\text{HighTemp})p(\neg\text{HighTemp}|\text{Flu}) \\ &= 0.87 \cdot 0.05 + 0.31 \cdot 0.95 \end{aligned}$$

and so

$$p(\text{Flu}|\text{HighTherm}) = \frac{0.87 \cdot 0.05}{0.87 \cdot 0.05 + 0.31 \cdot 0.95} = 0.128$$

1.9 Exercises without solutions

Exercise 1 Consider an experiment with three coins and denotes the result of the coin toss of the first coin with a, the result of the coin toss of the second coin with b and the result of the coin toss of the third coin with c. Assume you know the following joint distribution for three binary variables a, b, c:

a	b	c	$Prob$
0	0	0	0.192
1	0	0	0.032
0	1	0	0.252
1	1	0	0.048
0	0	1	0.072
1	0	1	0.128
0	1	1	0.108
1	1	1	0.168

a What is the probability $p(b = 0)$?

b What is the probability $p(b = 0, c = 0|a = 0)$?

Exercise 2 Consider two binary variables A and B and the conditional probability

$$p(A = 1|B = 0) = 0.8$$

and the following possible values of $p(A = 1|B = 1)$. In what case we can say that the two variables are independent.

a

$$p(A = 1|B = 1) = 0.8$$

b

$$p(A = 1|B = 1) = 0.2$$

c

$$p(A = 1|B = 1) = 0.5$$

d None of the three cases

Exercise 3 Assume you know the following joint distribution for three binary variables $Money, Rise, Spam$:

$Money$	$Rise$	$Spam$	$Prob$
0	0	0	0.208
1	0	0	0.056
0	1	0	0.312
1	1	0	0.084
0	0	1	0.052
1	0	1	0.084
0	1	1	0.078
1	1	1	0.126

Compute the posterior probability

$$p(Spam = 1|Money = 1, Rise = 0).$$

Compute the marginal probability

$$p(Rise = 0).$$

Exercise 4 Inspector Clouseau arrives at the scene of a crime. The Butler (b) and Maid (m) are his main suspects. The inspector has a prior belief of 0.6 that the Butler is the murderer, and a prior belief of 0.2 that the Maid is the murderer. These probabilities are independent in the sense that $p(b, m) = p(b)p(m)$. (It is possible that both the Butler and the Maid murdered the victim or neither). The inspector's *prior* criminal knowledge can be formulated mathematically as follows:

$$\Omega_b = \Omega_m = \{\mathsf{murderer}, \mathsf{not\ murderer}\}$$

$$\Omega_k = \{\mathsf{knife\ used}, \mathsf{knife\ not\ used}\}$$

$$p(b = \mathsf{murderer}) = 0.6, \quad p(m = \mathsf{murderer}) = 0.2$$

$$\begin{aligned}
&p(\mathsf{knife\ used}|b = \mathsf{not\ murderer}, \quad m = \mathsf{not\ murderer}) &&= 0.3\\
&p(\mathsf{knife\ used}|b = \mathsf{not\ murderer}, \quad m = \mathsf{murderer}) &&= 0.2\\
&p(\mathsf{knife\ used}|b = \mathsf{murderer}, \quad m = \mathsf{not\ murderer}) &&= 0.6\\
&p(\mathsf{knife\ used}|b = \mathsf{murderer}, \quad m = \mathsf{murderer}) &&= 0.1
\end{aligned}$$

The victim lies dead in the room and the inspector quickly finds the murder weapon, a Knife (k). What is the probability that the Butler is the murderer? (Remember that it might be that neither is the murderer). What is the probability that the Butler and not the Maid is the murderer?

Chapter 2

Learning

Much of AI and ML (and science) deals with problems of learning some unknown quantity from data. Let us consider again the dice example described in Section 1.3. The space of possibility is $\Omega = \{1, 2, 3, 4, 5, 6\}$, and let's assume we made $n = 100$ throws and it turns out:

event	1	2	3	4	5	6
freq.	15	12	18	21	16	18

Table 2.1: 100 rolls of a dice

Can we use this data to infer the value of the probability of the faces of the dice? Let us denote these six unknown probabilities as θ_i for $i = 1, \ldots, 6$.[8] Note that $\theta_i \geq 0$ and $\sum_{i=1}^{6} \theta_i = 1$ (they are probabilities).

[8] The notation "θ_i for $i = 1, \ldots, 6$" is mathematical notation to denote compactly $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$.

2.1 Maximum Likelihood Estimator

To find θ_i for $i = 1, \ldots, 6$, we first have to decide which model we believe best describes the process of generating the data (that is, we have to define a generative model of the data). In this case, this is very easy because by definition of θ_i, in one roll

$$p(\text{face i}) = \theta_i. \tag{2.1}$$

This is a generative model. For instance, we can use PyMC3, which is a Python library for probabilistic programming, for simulating a roll from a dice.[9]

[9] If you have problems installing pymc3, you can run the code in Google Colab: colab.research.google.com

Code

```
import pymc3 as pm
#Generative model
#we select probabilities for the six faces
theta = [0.1,0.2,0.2,0.2,0.2,0.1]
#we define a probabilistic model
dice = pm.Categorical.dist(theta)
print(dice.random())#result of one roll
print(dice.random())#result of another roll
```

```
#Output
> 3
> 1
```

[10] It returns the possible results of an experiment that can take on one of k possible categories, with the probability of each categories defined by the vector "theta".

The Categorical distribution[10] is the probabilistic model we defined in Equation (2.1). If we roll the dice twice and we want to talk about the output of the first roll and the output of the second roll, we should then consider all possible 36 outcomes

$$\Omega = \{(1,1),(1,2),\ldots,(6,5),(6,6)\}.$$

If we believe (assume) that the rolls are **independent**, then

$$p(\text{face } i \text{ at roll 1 and face } j \text{ at roll 2 }) = p(\text{face i})p(\text{face j}) = \theta_i\theta_j.$$

Code

```
import numpy as np
#Possibility space of two rolls
Omega=[(i,j) for i in range(1,7) for j in range(1,7)]
#we select probabilities for the six faces
theta = np.array([0.1,0.2,0.2,0.2,0.2,0.1])
theta_2=np.kron(theta,theta)#this computes all possible
                            products
print(theta_2)
#we define a probabilistic model
dice_2 = pm.Categorical.dist(theta_2)
print(Omega[dice_2.random()])#result of one roll
print(Omega[dice_2.random()])#result of another roll
```

```
#Output
> [0.01 0.02 0.02 0.02 0.02 0.01
0.02 0.04 0.04 0.04 0.04 0.02 0.02 0.04
 0.04 0.04 0.04 0.02 0.02 0.04 0.04 0.04
 0.04 0.02 0.02 0.04 0.04 0.04
 0.04 0.02 0.01 0.02 0.02 0.02 0.02 0.01]
> (1, 5)
> (5, 4)
```

If the rolls are independent, the probability of the pair $(1, 5)$ and $(5, 1)$ is the same, equal to $\theta_1\theta_5$. This is true also for any other pair (i, j) and it is also true if we roll the dice more than two times. For instance the probability of obtaining the faces $1, 1, 3$ in three consecutive rolls is equal to the probability of the sequences $1, 3, 1$ and $1, 1, 3$, that is $\theta_1^2\theta_3$. The order does not matter (if the rolls are independent), the only thing that matters are the counts (that is the number of times we obtained a certain face). In PyMC3, we can generate the counts for $n = 100$ independent rolls from a dice as follows:

Code

```
#we select probabilities for the six faces
theta = np.array([0.1,0.2,0.2,0.2,0.2,0.1])
#we define a probabilistic model
dice = pm.Multinomial.dist(100,theta)
print(dice.random())#result of 100 rolls
print(dice.random())#result of other 100 rolls
```

```
#Output
> [15. 13. 16. 17. 26. 13.]
> [ 8. 18. 17. 22. 24. 11.]
```

This time the code returns the number of times the result was face 1 (15 times), face 2 (13 times) etc..[11] Note that we obtained quite different counts in two different 100-rolls of the same dice. There is uncertainty!

[11] The Multinomial distribution models the outcome of n experiments, where the outcome of each trial has a categorical distribution, such as rolling a k-sided dice n times.

The probability of obtaining the counts $[15, 13, 16, 17, 26, 13]$ (note they sum up to 100) given the vector of probabilities of the six faces $\theta = [0.1, 0.2, 0.2, 0.2, 0.2, 0.1]$ is

$$\begin{aligned} p(D|\theta) &= \theta_1^{15}\theta_2^{13}\theta_3^{16}\theta_4^{17}\theta_5^{26}\theta_6^{13} \\ &= 0.1^{15}0.2^{13}0.2^{16}0.2^{17}0.2^{26}0.1^{13} \quad = 4.72 \cdot 10^{-79} \end{aligned}$$

This value is called **likelihood** of the data D given the θ_i. Therefore, we now know how to (i) simulate n independent rolls from a dice; (ii) compute the probability of a sequence of counts given the probabilities of the six faces.

However, we have not solved yet our initial problem that is given the counts in Table 2.1 of 100 rolls of a certain dice, to find the value of θ.

In this case, the likelihood is

$$p(D|\theta) = \theta_1^{15}\theta_2^{12}\theta_3^{18}\theta_4^{21}\theta_5^{16}\theta_6^{18}$$

but we do **not know** θ_i.

A method of estimating the unknowns θ_i is by maximizing the likelihood, that is we aim to find the value of $\theta_1, \ldots, \theta_6$ that is most likely to have generated the data.

We can do it analytically by computing the maximum of the function

$$L(D, \theta_1, \ldots, \theta_6) = \theta_1^{15}\theta_2^{12}\theta_3^{18}\theta_4^{21}\theta_5^{16}\theta_6^{18}$$

This a constrained nonlinear optimization problem because the above function is nonlinear in the unknowns θ_i and we have constraints: $\theta_i \geq 0$ and $\sum_{i=1}^{6} \theta_i = 1$ (they are probabilities). Maximizing a function is equivalent to maximising its logarithm[12] and so we can equivalently maximise:

$$15\ln(\theta_1) + 12\ln(\theta_2) + 18\ln(\theta_3) + 21\ln(\theta_4) + 16\ln(\theta_5) + 18\ln(\theta_6)$$

By computing the derivative (using the Lagrange's multipliers[13] method to take into account of the constraints), the maximum is

$$\theta_i = \frac{y_i}{\sum_{i=1}^{6} y_i},$$

where $y_1 = 15, y_2 = 12, y_3 = 18, y_4 = 21, y_5 = 16, y_6 = 18$ (they are the counts).

We have found that the Maximum Likelihood Estimate (MLE) of the probabilities θ_i is the **relative frequency** of the face i in n rolls:

$$\hat{\theta}_1 = \frac{15}{100},\ \hat{\theta}_2 = \frac{12}{100},\ \hat{\theta}_3 = \frac{18}{100},\ \hat{\theta}_4 = \frac{21}{100},\ \hat{\theta}_5 = \frac{16}{100},\ \hat{\theta}_6 = \frac{18}{100}.$$

We have denoted the MLE of θ_i as $\hat{\theta}_i$ to point out that $\hat{\theta}_i$ is not the true probability of face i but only our estimate based on the 100 counts data using the MLE method.

We can generate count data from a dice for some fixed probabilities of the faces and estimate back those probability from data using MLE. The following code shows how we can do that.

[12] Since the log function is a monotonically increasing function of its argument, maximizing the log is equivalent to maximizing the function. That is the value of θ that maximises $f(\theta)$ is the same of the value that maximises $\ln(f(\theta))$.

[13] For details about Langrange's method see https://en.wikipedia.org/wiki/Lagrange_multiplier

Code

```
#we select probabilities for the six faces
theta = np.array([0.1,0.2,0.2,0.2,0.2,0.1])
#we define a probabilistic model
dice = pm.Multinomial.dist(100,theta)
data=dice.random()
print(data)#counts for 100 rolls
print("MLE=",data/np.sum(data))
#no we throws the dice 10000 times
dice = pm.Multinomial.dist(100000,theta)
data=dice.random()
print(data)#counts for 100000 rolls
print("MLE=",data/np.sum(data))
```

```
[13. 17. 19. 27. 17.  7.]
MLE= [0.13 0.17 0.19 0.27 0.17 0.07]

[10105. 19843. 20176. 19844. 19933. 10099.]
MLE= [0.10105 0.19843 0.20176 0.19844 0.19933 0.10099]
```

You can notice how the MLE estimated probabilities converge to the true ones $[0.1, 0.2, 0.2, 0.2, 0.2, 0.1]$ at the increase of the size of the datasets (from 100 to 100000).[14]

[14] This is a general property of MLE, under certain assumption, MLE converges to the true value of the parameter when the size of the dataset goes to infinity https://en.wikipedia.org/wiki/Maximum_likelihood_estimation#Consistency.

2.1.1 Reguralisation

Consider this other dataset.

event	1	2	3	4	5	6
freq.	3	3	0	0	2	4

Table 2.2: 12 rolls of a dice

The MLE of the probabilities θ_i is:

$$\hat{\theta}_1 = \frac{3}{12}, \quad \hat{\theta}_2 = \frac{3}{12}, \quad \hat{\theta}_3 = \frac{0}{12}, \quad \hat{\theta}_4 = \frac{0}{12}, \quad \hat{\theta}_5 = \frac{2}{12}, \quad \hat{\theta}_6 = \frac{4}{12}.$$

So according to MLE, $\hat{\theta}_3 = \hat{\theta}_4 = 0$ that means *they cannot happen*! This is a common problem with MLE: it can return sharp probabilities after seeing few instances (only 12 in the above case). We would like the estimated probabilities to be less sharp when the number of rolls (dataset) is small.

In Machine Learning, we call this issue with MLE *over-fitting*.[15]

A way to reduce this problem is to add a reguralisation term. To estimate θ_i, we now maximise the function:

$$\left(\theta_1^3\theta_2^3\theta_3^0\theta_4^0\theta_5^2\theta_6^4\right)\left(\theta_1^{\alpha_1}\theta_2^{\alpha_2}\theta_3^{\alpha_3}\theta_4^{\alpha_4}\theta_5^{\alpha_5}\theta_6^{\alpha_6}\right)$$

The first term between brackets is the likelihood, while the second term is the reguralisation. The nonnegative scalars $\alpha_i > 0$ for $i = 1, 2, \ldots, 6$ are called reguralisation parameters. Note that the above expression is equal to

$$\theta_1^{3+\alpha_1}\theta_2^{3+\alpha_2}\theta_3^{\alpha_3}\theta_4^{\alpha_4}\theta_5^{2+\alpha_5}\theta_6^{4+\alpha_6}$$

therefore we can interpret the α_i as pseudo-observations.[16] In this case, our estimates $\hat{\theta}_i$ are equal to:[17]

[15] We have seen that, in the dice example, the MLE are the values of θ_i that maximise $\sum_{i=1}^{6} y_i \ln(\theta_i)$. This is equivalent to minimising $-\sum_{i=1}^{6} y_i \ln(\theta_i)$, which is called *Categorical Cross-Entropy loss* function in machine learning. This explains why we can call this phenomenon over-fitting.

[16] The α_i does not need to be integer, they can be any nonnegative number.

[17] This estimator is called **Maximum A-Posteriori** (MAP). You will learn why in the next chapter.

$$\hat{\theta}_i = \frac{y_i + \alpha_i}{\sum_{j=1}^{6} y_j + \alpha_j},$$

where y_i is the number of occurences of face i in n rolls. For instance if we select $\alpha_i = 1$ (this is called Laplace's smoothing), it is like assuming a-priori that we have observed one count for each face, so

$$\hat{\theta}_1 = \frac{3+1}{12+6} = \frac{4}{18}, \quad \hat{\theta}_2 = \frac{3+1}{12+6} = \frac{4}{18}, \quad \hat{\theta}_3 = \frac{1}{18},$$
$$\hat{\theta}_4 = \frac{1}{18}, \quad \hat{\theta}_5 = \frac{2+1}{12+6} = \frac{3}{18}, \quad \hat{\theta}_6 = \frac{4+1}{12+6} = \frac{5}{18}.$$

Note that, $\hat{\theta}_3, \hat{\theta}_4$ are not zero anymore.

2.2 Application: spam filter continues

A ML algorithm that uses *annotated (labeled)* past-data to predict the label (also called class) of unseen data is called **classifier**. Spam filtering is a **classification** task. In spam filtering:

- Class: spam or not-spam
- Features/attributes/inputs: words in the email.

Consider this dataset of 8 emails

Id	Text	Spam
0	Dear John, You won 100 euros!	1
1	Dear Ashley, You won a new car!	1
2	Hi Helen, We won the game yesterday! Bye	0
3	Huge discount for you buy our new clothes. Bye	0
4	Thank you for your offer	1
5	You won the lottery	1
6	Buy our cruise	1
7	Dear Charlie, Your documents are ready. Bye	0

The email number 0,1,2,5 includes the word "Won" only once; the email number 3,4,6,7 does not include "Won"; the email number 0,1,4,5,6 is spam and the email number 2,3,7 is not. We aim to build a probabilistic classifier that classifies these emails, as spam or not-spam, using only the word "Won", so that dataset reduces to this data table D:

The first row means that the first email includes the word "Won" and it is "Spam" and so on for the other rows (remember that we only care about the word "Won"). We assume that we have only the 8 above emails for simplicity.

$Spam$	Won
1	1
1	1
0	1
0	0
1	0
1	1
1	0
0	0

Table 2.3: One feature dataset

Our goal is to use that dataset to learn the joint distribution

$$p(s = 0, w = 0)$$
$$p(s = 0, w = 1)$$
$$p(s = 1, w = 0)$$
$$p(s = 1, w = 1)$$

that, as shown in Chapter 1, we can use to classify if an email is spam or not. For instance, the probability that an email is spam given it includes the word "Won" can be computed as

$$p(s = 1|w = 1) = \frac{p(s = 1, w = 1)}{p(w = 1)},$$

and so $p(s = 0|w = 1) = 1 - p(s = 1|w = 1)$ is the probability that an email is no-spam given it includes the word "Won". We can expand the above expression as:

$$p(s = 1|w = 1) = \frac{p(w = 1|s = 1)p(s = 1)}{p(w = 1|s = 0)p(s = 0) + p(w = 1|s = 1)p(s = 1)}.$$

Similarly, the probability that an email that does not include the word "Won" is spam

$$p(s = 1|w = 0) = \frac{p(w = 0|s = 1)p(s = 1)}{p(w = 0|s = 0)p(s = 0) + p(w = 0|s = 1)p(s = 1)}.$$

and so $p(s = 0|w = 0) = 1 - p(s = 1|w = 0)$.

Therefore, to compute these posterior probabilities that we use to classify emails as spam or no-spam, we need to know these input probabilities

$$p(w = 1|s = 1), p(w = 0|s = 1),$$
$$p(w = 1|s = 0), p(w = 0|s = 0),$$
$$p(s = 1), p(s = 0).$$

If we denote our dataset with D and the unknown probabilities as these variables:

$$\theta_{w=1|s=1}, \theta_{w=0|s=1},$$
$$\theta_{w=1|s=0}, \theta_{w=0|s=0},$$
$$\theta_{s=1}, \theta_{s=0}.$$

Our goal is to estimate these parameters from data.
We can estimate each row of parameters separately by maximising the likelihood function that is $p(Data|\theta)$: the Maximum Likelihood Estimator.

The solution of this maximisation problem is the relative frequency:

$$\hat{\theta}_{s=1} = \frac{\text{number of spam emails}}{\text{number of emails}} = \frac{5}{8}$$

and so $\hat{\theta}_{s=0} = \frac{3}{8}$.

$$\hat{\theta}_{w=1|s=1} = \frac{\text{number of spam emails that include Won}}{\text{number of spam emails}} = \frac{3}{5}$$

and so $\hat{\theta}_{w=0|s=1} = 1 - \hat{\theta}_{w=1|s=1} = \frac{2}{5}$.

$$\hat{\theta}_{w=1|s=0} = \frac{\text{number of not-spam emails that include Won}}{\text{number of not-spam emails}} = \frac{1}{3}$$

and so $\hat{\theta}_{w=0|s=0} = 1 - \hat{\theta}_{w=1|s=0} = \frac{2}{3}$.

How can we use these probabilities? Consider these new unlabeled emails:

Id	Text	Spam
0	Dear Jerry, You won 1000 euros!	?
1	Dear Sarah, I am writing you...!	?

are they spam? The first email includes the word "Won" while the second it does not. Therefore, to classify these two emails, we just need to compute

$$p(s = 1|w = 1) \text{ and respectively } p(s = 1|w = 0).$$

What is $p(s = 1|w = 1)$? It is the probability that an email is spam given it includes "Won"? By Bayes' rule, it is equal to:

$$\begin{aligned} p(s = 1|w = 1) &= \frac{p(w = 1|s = 1)p(s = 1)}{p(w = 1|s = 0)p(s = 0) + p(w = 1|s = 1)p(s = 1)} \\ &\approx \frac{\hat{\theta}_{w=1|s=1}\hat{\theta}_{s=1}}{\hat{\theta}_{w=1|s=1}\hat{\theta}_{s=1} + \hat{\theta}_{w=1|s=0}\hat{\theta}_{s=0}} \\ &= \frac{\frac{3}{5}\frac{5}{8}}{\frac{3}{5}\frac{5}{8} + \frac{1}{3}\frac{3}{8}} = 0.75 \end{aligned}$$

The symbol $\approx$ means approximately, the approximation is due to the fact that we use the estimated probabilities $\hat{\theta}$.

Therefore, if we make the decision to "move the email to spam folder when $p(s=1|w=1) \geq 0.5$", then any email that includes "Won" will be classified as spam. For an email that does not include "Won":

$$\begin{aligned} p(s=1|w=0) &= \frac{p(w=0|s=1)p(s=1)}{p(w=0|s=0)p(s=0)+p(w=0|s=1)p(s=1)} \\ &\approx \frac{\hat{\theta}_{w=0|s=1}\hat{\theta}_{s=1}}{\hat{\theta}_{w=0|s=1}\hat{\theta}_{s=1}+\hat{\theta}_{w=0|s=0}\hat{\theta}_{s=0}} \\ &= \frac{\frac{2}{5}\frac{5}{8}}{\frac{2}{5}\frac{5}{8}+\frac{2}{3}\frac{3}{8}} = 0.5 \end{aligned}$$

We have just derived a **Multinomial Naive Bayes** classifier. We can do the same computations using *sklearn*.

Code

```
from sklearn.naive_bayes import MultinomialNB
import numpy as np
X=np.array([[1,0],[1,0],[1,0],[0,1],[0,1],
[1,0],[0,1],[0,1]])# 1 Won included, 0 not included
y=np.array( [1,1,0,0,1,1,1,0])#1 spam, 0 not spam
Xtest=np.array([[1,0],[0,1]]).T
clf = MultinomialNB(alpha=0.0)
clf.fit(X, y)
print(clf.predict_proba(Xtest))
print(clf.predict(Xtest))
```

```
>[[0.25 0.75],
 [0.5  0.5 ]]

>[1 1]
```

In the above code, X, y is our training dataset. Each element is a different email and y tells us if the email is spam (value 1) or not. Each row of X tells us if the word "Won" is present in that email (value $[1,0]$) or if it is not present (value $[0,1]$).[18] You can verify that is exactly the dataset we used previously, Table 2.3. Xtest encodes out test dataset. We must first initialise our classifier `MultinomialNB(alpha=0.0)` (α is the reguralisation parameters, by default in sklearn $\alpha = 1.0$). We then compute the MLE via `clf.fit(X, y)`. Finally, we predict the probability of the test set using `clf.predict_proba(Xtest)` (this computes Bayes' rule). This latter function returns two probabilities for each instance in

[18] MultinomialNB in Sklearn accepts counts as input. In case of one single feature, the input must be encoded using "OneHotEncoder", that is why 1 was encoded into $[1,0]$ and 0 into $[0,1]$ (otherwise the feature is not used).

Xtest. The first probability is $p(s = 0|Data)$ and the second is $p(s = 1|Data)$ (note in fact that each row sums up to one). `clf.predict(Xtest)` instead returns a decision (the class): its value is one if $p(s = 1|Data) \geq 0.5$. In other words, an email is classified as "Spam" if $p(s = 1|Data) \geq 0.5$.

2.2.1 Reguralisation

If we set $\alpha = 1$, then the probabilities become:

Code

```
from sklearn.naive_bayes import MultinomialNB
import numpy as np
X=np.array([[1,0],[1,0],[1,0],[0,1],[0,1],
[1,0],[0,1],[0,1]])# 1 Won included, 0 not included
y=np.array( [1,1,0,0,1,1,1,0])#1 spam, 0 not spam
Xtest=np.array([[1,0],[0,1]]).T
clf = MultinomialNB(alpha=1.0)
clf.fit(X, y)
print(clf.predict_proba(Xtest))
print(clf.predict(Xtest))
```

```
>[[0.29577465 0.70422535],
 [0.45652174 0.54347826]]

>[1 1]
```

The MLE estimate becomes:

$$\hat{\theta}_{w=1|s=1} = \frac{\text{number of spam emails that include Won} + \alpha}{\text{number of spam emails} + 2\alpha}$$

and

$$\hat{\theta}_{w=1|s=0} = \frac{\text{number of no-spam emails that include Won} + \alpha}{\text{number of no-spam emails} + 2\alpha}$$

but

$$\hat{\theta}_{s=1} = \frac{\text{number of spam emails}}{\text{total number of emails}}$$

that is in Sklearn no reguralisation is added to the estimate of $\hat{\theta}_{s=1}$.[19]

[19] This is just an implementation choice in Sklearn.

2.2.2 Naive hypothesis

What does Naive mean in **Multinomial Naive Bayes**? Assume now we want to use another word "Bye". Our goal is to use that

dataset to learn the joint distribution

$$\begin{aligned}
&p(s=0,b=0,w=0)\\
&p(s=0,b=0,w=1)\\
&p(s=0,b=1,w=0)\\
&p(s=0,b=1,w=1)\\
&p(s=1,b=0,w=0)\\
&p(s=1,b=0,w=1)\\
&p(s=1,b=1,w=0)\\
&p(s=1,b=1,w=1)
\end{aligned}$$

We have 2^3-1 unknown quantities (because they sum up to one, we have only 7 unknowns and not 8). In a real spam filter, we must consider hundreds of words. For instance, if we consider the 300 most common words in English, then we will have a table with $2^{300}-1$ rows, that is bigger than the number of atoms in the Universe! We cannot even store that table in the Universe. What do we do?

Note that, the probability that an email is spam given it includes "Bye" and "Won" is:

$$p(s=1|b=1,w=1)=\frac{p(b=1,w=1|s=1)p(s=1)}{p(b=1,w=1|s=1)p(s=1)+p(b=1,w=1|s=0)p(s=0)}$$

If we assume that the variables b,w are independent given s, we can simplify it as

$$\approx \frac{p(w=1|s=1)p(b=1|s=1)p(s=1)}{p(w=1|s=1)p(b=1|s=1)p(s=1)+p(w=1|s=0)p(b=1|s=0)p(s=0)}$$

In machine learning, we often assume that the features are conditional independent given the class (this is called "Naive" hypothesis because it is in general not true but it allows us to reduce the number of unknowns). In this case the unknowns are only 5:

$$\begin{aligned}
p(w=1|s=1),p(w=1|s=0),\\
p(b=1|s=1),p(b=1|s=0),\\
p(s=1)
\end{aligned}$$

More in general the number of unknowns is

$$2\cdot\text{number of words}+1.$$

In case we use 300 words, $2\cdot 300+1=601$ probabilities that we need to estimate from data. This is more than feasible. That is why we care about **Conditional Independence** because it allows us to reduce the size of the unknowns. We can now design our implementation of "MultinomialNB".[20]

[20] Do you understand all the parts in the code? Is the code a correct implementation of "MultinomialNB"? How can you test that?

Code

```
import numpy as np
class MultinomialNB(object):
    #class constructor
    def __init__(self, alpha=1.0):
        self.alpha = alpha#set the smoothing parameter
    #fit method
    def fit(self, X, y):
        count_sample = X.shape[0]
        split_per_class = [[x for x, t in zip(X, y) if t
                                                  == c]
                           for c in np.unique(y)]

        self.class_log_prior_ = [np.log(len(i) /
                                        count_sample) for i
                                        in split_per_class]
        count = np.array([np.array(i).sum(axis=0) for i
                                        in split_per_class])
                                         + self.alpha
        self.feature_log_prob_ = np.log(count / count.
                                        sum(axis=1)[np.
                                        newaxis].T)

    def predict_proba(self, X):
        proba= [np.exp((self.feature_log_prob_ * x).sum(
                                        axis=1) + self.
                                        class_log_prior_)
                for x in X]
        prob=[p/np.sum(p) for p in proba]
        return np.vstack(prob)


    def predict(self, X):
        return np.argmax(self.predict_log_proba(X), axis
                                        =1)

#TEST
X=np.array([[1,0],[1,0],[1,0],[0,1],[0,1],[1,0],[0,1],[0
                                ,1]])
y=np.array( [1,1,0,0,1,1,1,0])#1 spam, 0 not spam
Xtest=np.array([[1,0],[0,1]]).T

MNB = MultinomialNB(1)
MNB.fit(X,y)
MNB.predict_proba(Xtest)
```

```
array([[0.29577465, 0.70422535],
       [0.45652174, 0.54347826]])
```

With this assumption, we can now design a real Spam filter, see the notebook[21] `Learning_to_Classify_Text_Analysis_Spam_Filter`.

[21] Before going through this notebook, we suggest to look at the "Text Mining" notebooks, see Section 2.5.

2.3 Language Modelling

Consider these problems:

- Text input on a smart phone. How does the phone know the correct word when you mistype, or type by swiping?
- Similarly, how does an Optical Character Recognition (OCR) system distinguish 1 vs. l or 0 vs. O. They are very similar.

We know that certain letter sequences are more likely than others. **Language modeling** tries to capture the notion that some text is more likely than others. It does so by estimating the probability $p(s)$ of any text s. Given input signal a (be it smartphone input, digitized pixels), the recognition process can be defined as finding the text s that maximizes, among all texts, the conditional probability $p(s|a)$, that is

$$s^* = \arg\max_{s \in \mathcal{S}} p(s|a). \tag{2.2}$$

By Bayes'rule, we have

$$p(s|a) = \frac{p(a|s)p(s)}{p(a)}. \tag{2.3}$$

Note $p(a)$ is a constant w.r.t. the maximization over s, so that the problem reduces to

$$s^* = \arg\max_{s \in \mathcal{S}} p(a|s)p(s). \tag{2.4}$$

The term $p(a|s)$ describes the probability of producing signal a when the underlying text is s. The term $p(s)$ is the *language model*, which we discuss now.

As for the spam filter, we assume words are the basic units, i.e. s is a sequence of words $w_1 \dots w_n$ that we can more compactly denote as $w_{1:n}$. Estimating $p(s)$ directly by counting is not feasible if n is large, because the number of possible sentence with n words is too large. We will instead use the *chain rule*.

Result: Let x, y, z, u four (or more) variables, the chain rule of probability states that the joint distribution $p(x, y, z, u)$ can be factorised as

$$p(x, y, z, u) = p(x|y, z, u)p(y|z, u)p(z|u)p(u)$$

Therefore,

$$p(s) = p(w_{1:n}) = p(w_1)p(w_2|w_1)p(w_3|w_{1:2}) \dots p(w_n|w_{1:n-1}). \tag{2.5}$$

We do not gain much in terms of computation – there are too many possible combinations of words (factorial) and so too many quantities to be estimated.

The basic idea in **n-gram language modeling** is to approximate the chain rule by shortening the histories.

2.3.1 Unigram Language Model

A *unigram* model makes the strong *independence assumption* that words are generated independently from a multinomial distribution with probabilities

$$\theta_1, \theta_2, \dots, \theta_V$$

where V is the size of the vocabulary (the number of words we consider). Note that $\theta_i \geq 0$ and $\sum_{i=1}^{V} \theta_i = 1$ (they are probabilities). That is,

$$p(w_{1:n}) = \prod_{i=1}^{n} p(w_i|\theta) = \prod_{w=1}^{V} \theta_w^{n_w}, \tag{2.6}$$

where n_w is the count of word w in s. This generative model is similar to the one we defined for the dice, but instead of having 6 faces we have V faces (one for each word).[22] The unigram approximation is

$$p(w_i|w_{1:i-1}) \approx p(w_i).$$

In other words, this model ignores word orders. But this works well in practice. This is indeed the model we used for the spam filter.

[22] The Unigram model is also called **bag of words** model, because as generative model is equivalent to extracting a word at random from a bag of words.

Where do we get θ? We can estimate it from a corpus using MLE. We need a vocabulary of V word types, and counts $n_{1:V}$ of each word type in the corpus. The MLE is

$$\hat{\theta}_i = \frac{n_i}{\sum_{i=1}^{V} n_i}. \tag{2.7}$$

where $\sum_{i=1}^{V} n_i$ is the length of the corpus. In this case the MLE is simply the frequency estimate, as we have seen before.

As we have seen, there is a problem with the MLE: if a word (e.g., "bumblebee") is in the vocabulary but not in the corpus (hence $n_i = 0$), the MLE is $\hat{\theta}_i = 0$ (see dice example in Section 2.1.1). Any new sentence with "bumblebee" in it will thus be considered impossible since it have zero probability. This is not

desirable and can be avoided by either restricting the vocabulary to the words in the corpus or adding a symbol to the vocabulary for "unknown word" and using a reguralisation term to prevent that symbol has probability zero.

For instance, consider again our dataset

Id	Text	Spam
0	Dear John, You won 100 euros!	1
1	Dear Ashley, You won a new car!	1
2	Hi Helen, We won the game yesterday! Bye	0
3	Huge discount for you buy our new clothes. Bye	0
4	Thank you for your offer	1
5	You won the lottery	1
6	Buy our cruise	1
7	Dear Charlie, Your documents are ready. Bye	0

but ignore the Spam column. We can use this dataset as a corpus and learn a unigram model. We just need the counts of each word.

Code

```
import nltk as nltk
from sklearn.feature_extraction.text import
                                CountVectorizer
D=["Dear John You won 100 euros",
"Dear Ashley You won a new car",
"Hi Helen We won the game yesterday Bye",
"Huge discount for you buy our new clothes Bye",
 "Thank you for your offer", "You won the lottery",
 "Buy our cruise",
 "Dear Charlie Your documents are ready Bye"]
Corpus = [word.lower() for word in D]# we make
                                everything lowercase
vectorizer=CountVectorizer()
R=vectorizer.fit_transform(Corpus).toarray()
WordsList=np.array(vectorizer.get_feature_names())
print(WordsList)
print(R)#tells us which words are present in each email
print(np.sum(R,axis=0))#total counts of each word in
#the corpus
```

```
 >['100' 'are' 'ashley' 'buy' 'bye' 'car' 'charlie'
 'clothes' 'cruise' 'dear' 'discount' 'documents'
 'euros' 'for' 'game' 'helen' 'hi' 'huge'
 'john' 'lottery' 'new' 'offer' 'our' 'ready'
 'thank' 'the' 'we' 'won' 'yesterday' 'you' 'your']

>[[1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1 0 1 0]
```

```
 [0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 1 0 1 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0
 0 0 0 0 1 1 1 1 0 0]
 [0 0 0 1 1 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 1
 0 1 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 1 0 0 1 0 0 0 0 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 1 0 1 0 1 0]
 [0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0]
 [0 1 0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 1]]

>[1 1 1 2 3 1 1 1 1 3 1 1 1 2 1 1 1 1 1 1 2
1 2 1 1 2 1 4 1 5 2]
```

The MLE is

$$p(w_i) = \theta_i \approx \frac{n_i}{\sum_{i=1}^{V} n_i}$$

for instance $p(\text{bye}) \approx \frac{3}{42}$. We can use the MLE to generate random sentence from the above corpus using the multinomial distribution (as for the Dice example). For instance, we can generate random sentences including 5 words as follows:

Code

```
Counts = np.sum(R,axis=0)
#
theta = Counts/np.sum(Counts)#MLE estimate of theta
#we define a probabilistic model
sentence = pm.Multinomial.dist(5,theta)
print(sentence.random()==1)
print(WordsList[sentence.random()==1])#generate a random
                                email with 5 words
print(WordsList[sentence.random()==1])#generate a random
                                email with 5 words
print(WordsList[sentence.random()==1])#generate a random
                                email with 5 words
```

```
> [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 1. 0.]

> ['documents' 'our' 'we' 'won' 'you']
```

```
> ['discount' 'new' 'the' 'won' 'you']

> ['buy' 'charlie' 'clothes' 'for' 'you']
```

2.3.2 N-gram Language Models

A better approximation is to keep some history. An n-gram language model keeps a history of $n-1$ words:

$$p(w_i|w_{1:i-1}) \approx p(w_i|w_{i-n+1:i-1}). \tag{2.8}$$

The conditioning part $w_{i-n+1:i-1}$ is called 'history', which has $n-1$ previous words. Compared to Equation (2.5), the n-gram language model states that

$$p(w_{1:n}) = \prod_{i=1}^{n} p(w_i|w_{i-n+1:i-1}). \tag{2.9}$$

Unigram is a special case when $n = 1$, bigram $n = 2$ and trigram $n = 3$.

It is worth noting that the number of parameters in θ grows rapidly as $O(V^n)$. Note that for a given history h, $p(w|h)$ is a multinomial of size V, but there are V^{n-1} such possible histories, and the vector of parameters θ consists of all these multinomials. For $V = 10,000$ which is typical, in theory there are 10,000-1 unigram parameters, 10^8 bigram parameters , and 10^{12} trigrams. In practice, the number of n-grams is bounded by corpus length (some combinations of words never appear in a language).[23]

The following code generates, from our simplified dataset of emails, the list of all possible unigrams and bigrams. It also computes the frequencies of all bigrams.

[23] In 2006, Google released a "Web 1T 5-gram" corpus on 6 DVDs, with counts from 10^{12} tokens. http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2006T13.

Code

```
  import nltk as nltk
from sklearn.feature_extraction.text import
                              CountVectorizer
D=["Dear John You won 100 euros",
"Dear Ashley You won a new car",
"Hi Helen We won the game yesterday Bye",
"Huge discount for you buy our new clothes Bye",
 "Thank you for your offer", "You won the lottery",
 "Buy our cruise",
 "Dear Charlie Your documents are ready Bye"]
Corpus = [word.lower() for word in D]# we make
                               everything lowercase
vectorizer=CountVectorizer()
R=vectorizer.fit_transform(Corpus).toarray()
```

```
WordsList=np.array(vectorizer.get_feature_names())
print(WordsList)
vectorizer=CountVectorizer(ngram_range=(2,2))#bigrams
R=vectorizer.fit_transform(Corpus).toarray()
Bigrams=np.array(vectorizer.get_feature_names())
print(Bigrams)
Frequencies=np.zeros((WordsList.shape[0],WordsList.shape
                                [0]))
for i in range(len(WordsList)):
    for j in range(len(WordsList)):
        if WordsList[i]+' '+WordsList[j] in Bigrams:
            Frequencies[i,j]=Frequencies[i,j]+1
```

```
>['100' 'are' 'ashley' 'buy' 'bye' 'car' 'charlie'
 'clothes' 'cruise' 'dear' 'discount' 'documents'
 'euros' 'for' 'game' 'helen' 'hi' 'huge'
 'john' 'lottery' 'new' 'offer' 'our' 'ready'
 'thank' 'the' 'we' 'won' 'yesterday' 'you' 'your']

>['100 euros' 'are ready' 'ashley you' 'buy our'
'charlie your'
 'clothes bye' 'dear ashley' 'dear charlie'
 'dear john' 'discount for'
 'documents are' 'for you' 'for your'
 'game yesterday' 'helen we'
 'hi helen' 'huge discount' 'john you'
 'new car' 'new clothes'
 'our cruise' 'our new' 'ready bye' 'thank you'
 'the game' 'the lottery'
 'we won' 'won 100' 'won new' 'won the'
 'yesterday bye' 'you buy'
 'you for' 'you won' 'your documents'
 'your offer']
```

We can use the frequencies (counts) to estimate the probability (via MLE) that a certain bigram appears in an email. In this example, the corpus is very small and, therefore, many bigrams have zero counts. Therefore we use smoothing with a small α to estimate the probabilities. We can then use the estimated probabilities to generate random emails using *chain rule*. We start from a word "dear", and then we generate a new word according to the estimated conditional probabilities

$$p(100|dear), p(are|dear), p(ashley|dear), \ldots, p(your|dear)$$

using the relative multinomial distribution. Assume the generated random word is "charlie", then we can generate a new word based

on:

$p(100|charlie), p(are|charlie), p(ashley|charlie), \ldots, p(your|charlie)$

Assume the generated random word is "your", then we can generate a new word based on:

$$p(100|your), p(are|your), p(ashley|your), \ldots, p(your|your)$$

and so on.

Code

```
theta=(Frequencies+0.0001)/np.sum(Frequencies+0.0001,
                                  axis=1)[:,None]  #MLE with
                                  smoothing
word="dear"#initial word
Sentence=[word]
for i in range(5):
    ind=np.where((word == WordsList)==True)[0]
    word = WordsList[np.where(pm.Multinomial.dist(1,
                                    theta[ind,:]).random()==
                                    1)[1]][0]#generate
                                    random new word based on
                                     bigram frequency
    Sentence.append(word)
print(Sentence)
word="hi"
Sentence=[word]
for i in range(5):
    ind=np.where((word == WordsList)==True)[0]
    word = WordsList[np.where(pm.Multinomial.dist(1,
                                    theta[ind,:]).random()==
                                    1)[1]][0]
    Sentence.append(word)
print(Sentence)
```

```
> ['dear','charlie','your','documents','are','ready']

> ['hi','helen','we','won','100','euros']
```

You can notice how the random sentences are now more realistic. This means that a bigram is a better language model then unigram.

2.4 Application: spam filter continues

Can we improve our spam filter using bigrams? Our goal is

$$p(spam|w_1, \ldots, w_n) = \frac{p(w_1, \ldots, w_n|spam)p(spam)}{p(w_1, \ldots, w_n}$$

In the Unigram case, we have assumed that

$$p(w_1, \ldots, w_n|s) \approx \prod_{i=1}^{n} p(w_i|s),$$

that is the words are independent given the class ($s = 1$ spam or $s = 0$ not-spam). We could assume a more realistic model:[24]

$$p(w_1, \ldots, w_n|s) \approx p(w_1|s)p(w_2|w_1, s)p(w_3|w_2, s) \ldots p(w_n|w_{n-1}, s)$$

that is the occurrence of the word w_i in an email depends on the class (spam or not-spam) but also on the previous word. In this case, for each word, we have to learn

$$p(w_i = 1|s = 1), p(w_i = 1|s = 0)$$

but also

$$p(w_i = 1|w_{i-1}, s = 1), p(w_i = 1|w_{i-1}, s = 0)$$

for each possible word w_{i-1}.

[24] This means that the words are not anymore conditionally independent given the class. That is the naive hypothesis does not hold anymore.

2.5 Further material

The following material discusses techniques for text pre-processing.

- https://notebooks.azure.com/graememalcolmoutlook/projects/text-mining (see in particular the notebooks 02 and 03.)

2.6 Exercises with solutions

Exercise 1 Consider the following dataset:

money	win	spam
1	0	0
1	1	0
0	1	1
0	1	0
0	0	1
1	0	1
1	0	1
0	1	1

Each row represents a different email and specify if that email includes the word “money” and “win”, and if it is spam or not. For instance, the first email (first row) includes the word “money”, does not include “win” and it is spam.

a What is the MLE of $p(spam = 1)$?

b What is the MLE of $p(win = 0|spam = 1)$ and $p(win = 0|spam = 0)$?

c What is the MLE of $p(money = 1|spam = 1)$ and $p(money = 1|spam = 0)$?

d By using the previous maximum likelihood estimates of the probabilities and assuming that the features (money and win) are conditionally independent given the class (spam), compute $p(spam = 1|money = 1, win = 0)$?

Exercise 2 Our goal is to classify mushrooms as either Poisonous or Edible by only using the colour of the mushroom (red, white, brown). We have trained a Multinomial Naive Bayes classifier that estimated the following probabilities

$$p(Poisonous) = 0.4$$

and

$$p(Red|Poisonous) = 0.7, \quad p(White|Poisonous) = 0.2$$

and

$$p(Red|Edible) = 0.3, \quad p(White|Edible) = 0.3$$

Consider the following test set

color	class
red	Poisonous
white	Poisonous
brown	Edible

a Compute the posterior probability of the two classes in all instances of the test set and the predicted class returned by Multinomial Naive Bayes.

b Evaluate the accuracy of the classifier.

2.6.1 Solutions

Exercise 1 The MLE of $p(spam = 1)$ is

$$\frac{\text{number of spam email}}{\text{number of emails}} = \frac{5}{8}.$$

The MLE of $p(win = 0|spam = 1)$ is

$$\frac{\text{number of emails with win=0, spam=1}}{\text{number of emails with spam=1}} = \frac{3}{5}.$$

The MLE of $p(win = 0|spam = 0)$ is

$$\frac{\text{number of emails with win=0, spam=0}}{\text{number of emails with spam=0}} = \frac{1}{3}.$$

The MLE of $p(money = 1|spam = 1)$ is

$$\frac{\text{number of emails with money=1, spam=1}}{\text{number of emails with spam=1}} = \frac{2}{5}.$$

The MLE of $p(money = 1|spam = 0)$ is

$$\frac{\text{number of emails with money=1, spam=0}}{\text{number of emails with spam=0}} = \frac{2}{3}.$$

Concerning the last question

$$\begin{aligned} p(spam = 1|money = 1, win = 0) &= \frac{p(win = 0, money = 1, spam = 1)}{p(win = 0, money = 1)} \\ &= \frac{p(win = 0, money = 1|spam = 1)p(spam = 1)}{p(win = 0, money = 1)} \end{aligned}$$

By exploiting conditional independence of the inputs given the class, we can write

$$\begin{aligned} p(win = 0, money = 1|spam = 1) &= p(win = 0|spam = 1)p(money = 1|spam = 1) \\ &= \frac{3}{5}\frac{2}{5} = \frac{6}{25} \end{aligned}$$

and

$$\begin{aligned} p(win = 0, money = 1) &= p(win = 0|spam = 1)p(money = 1|spam = 1)p(spam = 1) \\ &\quad + p(win = 0|spam = 0)p(money = 1|spam = 0)p(spam = 0) \\ &= \frac{6}{25}\frac{5}{8} + \frac{2}{3}\frac{1}{3}\frac{3}{8} = 0.234 \end{aligned}$$

and so

$$\begin{aligned} p(spam = 1|money = 1, win = 0) &= \frac{p(win = 0, money = 1, spam = 1)}{p(win = 0, money = 1)} \\ &= \frac{\frac{6}{25}\frac{5}{8}}{0.234} = 0.643 \end{aligned}$$

Exercise 2 Let us denote the class Edible with 1 and Poisonous with 0. By Bayes' rule

$$p(class = 0|color = c) = \frac{p(color = c|class = 0)p(class = 0)}{p(color = c)}$$

where

$$p(color = c) = p(color = c|class = 0)p(class = 0) + p(color = c|class = 1)p(class = 1)$$

Therefore, for the first row in the dataset we have: so

$$p(Poisonous|Red) = \frac{p(Red|Poisonous)p(Poisonous)}{p(Red)} = 0.608$$

and, since $0.608 > 0.5$, our prediction is Poisonous. For the second row

$$p(Poisonous|White) = \frac{p(White|Poisonous)p(Poisonous)}{p(White)} = 0.307$$

and so our prediction is Edible.

$$p(Poisonous|Brown) = \frac{p(Brown|Poisonous)p(Poisonous)}{p(Brown)} = 0.143$$

and so our prediction is Edible. Therefore, our prediction is $[0, 1, 1]$, while the true class is $[0, 0, 1]$. The accuracy is

$$\frac{\text{number of correctly classified instances}}{\text{number instances}} = \frac{2}{3}.$$

2.7 Exercises without solutions

Exercise 1 Our goal is to classify mushrooms as either Poisonous or Edible or Unknown by only using the colour of the mushroom (red, white, brown). The feature is not binary (it has three elements), but the Multinomial Naive Bayes classifier can also be used in this case Consider the following dataset:

colour	win
red	Poisonous
red	Poisonous
white	Poisonous
white	Edible
brown	Poisonous
brown	Edible
white	Edible
brown	Poisonous

a Compute the MLE estimates of $p(colour|Poisonous)$, $p(colour|Edible)$ and $p(Poisonous)$ using the Laplace's reguralisation.

b Use the estimated probabilities to answer the following question "What is the probability that a brown mushroom is edible (based on the data above)?".

Exercise 2 Run "MultinomialNB" on the above dataset and check the results. You can use CountVectorizer to transform the inputs in the right format for MultinomialNB.

Code

```
from sklearn.feature_extraction.text import
                                    CountVectorizer
corpus = ['red','brown','brown','white']
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
print(vectorizer.get_feature_names())
print(X.toarray())
```

Chapter 3

Continuous Probability

This chapter will include some mathematics (integrals), but only to provide a background for the methods we will use in this module. Please don't worry if you feel the mathematics is challenging. It is more important to understand the concepts and the building blocks. The difficult mathematics will be handled by the software.

3.1 Continuous variables

Figure 3.1: https://en.wikipedia.org/wiki/Fishcake

In many practical applications, the space of possibility Ω includes an uncountably infinite number of possible values. For instance, consider the variable "weight" w of fish cakes. This can be any nonnegative number, e.g., 0.2Kg, 0.2500001Kg, 0.157888Kg. Therefore, the space of possibility is $\Omega = \mathbb{R}^+$, where $\mathbb{R}^+ = [0, \infty)$ denotes the space of nonnegative real numbers. Assume fish cakes are advertised to weigh a 0.25 kg, and assume we buy one of these fish cakes from the supermarket:

- What is
$$p(w = 0.25),$$
that is what is the probability that the weight of the fish cakes is exactly 0.25Kg?

- What is
$$p(0.1 \leq w \leq 0.2),$$
that is what is the probability that the weight of the fish cakes is in the interval $[0.1, 0.2]$Kg?

- What is
$$p(w \geq 0.25),$$

that is what is the probability that the weight of the fish cakes is greater than 0.25Kg?

In Chapter 1, we defined a *probability mass function* (PMF) as the list of all probabilities $p(x = \mathsf{x})$ for each element $\mathsf{x} \in \Omega$. Note that, if Ω is infinite as in the fish cakes weight case, then $p(w = \mathsf{w}) = 0$. For instance, think about $p(w = 0.25)$, it refers to the probability that a fish cake weighs exactly 0.25kg (meaning $0.250000\ldots$ with infinite zeroes). This probability is clearly zero and this is also true for any other value of $\mathsf{w} \in \Omega$, that is $p(w = \mathsf{w}) = 0$. However, the probabilities

$$p(0.1 \leq w \leq 0.2), \quad p(w \geq 0.25)$$

are not zero. It is reasonable for instance to say that

$$p(w \geq 0.25) = 0.5 \text{ and } p(w < 0.25) = 0.5$$

and that

$$p(0.1 \leq w \leq 0.2) \geq p(0.15 \leq w \leq 0.2)$$

and that

$$p(w \geq 0.5) \approx 0,$$

where $\approx$ means approximately.

Summing up, for infinite possibility spaces Ω, $p(w = \mathsf{w}) = 0$ but $p(\mathsf{w}_1 \leq w \leq \mathsf{w}_3)$ (with $\mathsf{w}_1 \leq \mathsf{w}_3$) may be different from zero. Moreover, we want to satisfy this consistency condition[25]

$$p(\mathsf{w}_2 \leq w \leq \mathsf{w}_3) \leq p(\mathsf{w}_1 \leq w \leq \mathsf{w}_3),$$

for any $\mathsf{w}_1 \leq \mathsf{w}_2 \leq \mathsf{w}_3$.

How can we define a probabilistic model that satisfies the above conditions?

In the fish cake example, the fact that fish cakes are advertised to weigh 0.25kg means that on average they weigh 0.25kg, but there is a "small" variability (dispersion). We can describe that with a function:

[25] The probability of a subset is not greater than that of any of its supersets.

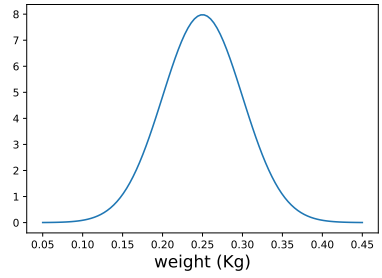


Figure 3.2: PDF for the fish cakes example

this **nonnegative** function is called a *Probability Density Function* (PDF). A PDF is a measure of the possibility[26] of observing w over some range, for instance $0.2 \leq w \leq 0.3$. For instance, given that the values of the curve in Figure 3.2 for all the values in the interval $0.2 \leq w \leq 0.3$ is higher than the values in the interval $0.3 \leq w \leq 0.4$, the PDF in Figure 3.2 tells that a weight in the interval $[0.2, 0.3]$ is more likely to occur than in $[0.3, 0.4]$. A PDF is **not** a probability as you can notice for instance from its value at $w = 0.25$, which is about 8 and so greater than 1. **Probability is represented by area under the curve**.

[26] We used the term "possibility" because the PDF is not a probability.

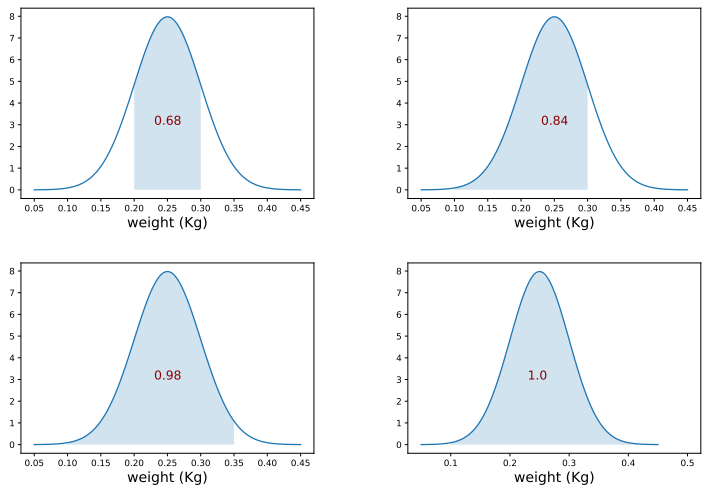


Figure 3.3: Probability is the area under the curve.

In Figure 3.3, the area in the left-top plot corresponds to the probability that $0.2 \leq w \leq 0.3$, that is $P(0.2 \leq w \leq 0.3) = P(w \in [0.2, 0.3]) = 0.68$; the right-top plot is $P(0 \leq w \leq 0.3) = P(w \in [0, 0.3]) = 0.84$; the left-bottom plot is $P(0 \leq w \leq 0.35) = P(w \in [0, 0.35]) = 0.98$ and the right-bottom plot is $P(0 \leq w \leq 0.5) = P(w \in [0, 0.5]) = 1$.

The probability of a specific value of a continuous variable will be zero because the area under a point is zero, as shown hereafter.

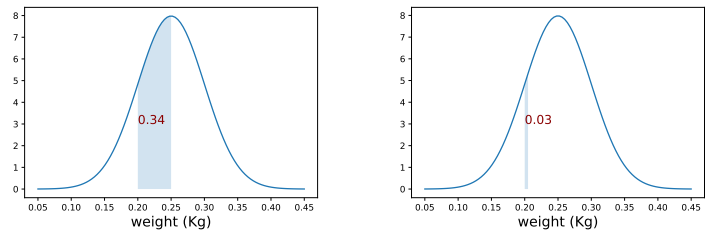


Figure 3.4: Probability is the area under the curve.

Probability is area. Practically, if we denote the PDF as the function $f(w)$, the area under the curve, in the interval $0.2 \leq w \leq 0.3$, is the integral of $f(w)$ in the interval $0.2 \leq w \leq 0.3$:

$$P(w \in [0.2, 0.3]) = \int_{0.2}^{0.3} f(w)dw.$$

3.2 Probability Density Function

Definition: The probability density function (PDF) of a continuous variable x with domain Ω is an integrable function $f(x)$ satisfying the following:

1. $f(x)$ is nonnegative everywhere in the domain Ω, that is, $f(x) \geq 0$, for all x in Ω

2. The area under the curve $f(x)$ in the domain Ω is 1, that is:

$$\int_{\Omega} f(x)dx = 1$$

3. If $f(x)$ is the PDF of x, then the probability that x belongs to some subset A of Ω is given by the integral of $f(x)$ over that subset, that is:

$$P(x \in A) = \int_{A} f(x)dx$$

As you can see, the definition for the PDF of a continuous random variable differs from the definition for the PMF of a discrete random variable by simply changing the summations that appeared in the discrete case to integrals in the continuous case.

The PDF is used to describe probabilities for continuous variables. The area under the density curve between two points corresponds to the probability that the variable falls between those two

values. In other words, the area under the density curve between points a and b is equal to $P(a \leq x \leq b)$. The total area under the graph of $f(x)$ is one.

In the following, we will denote a PDF with $p(x)$ instead of $f(x)$, to resemble the notation we used for a PMF and we will use capital $P(\cdot)$ to denote probability. Therefore, we will write

$$P(x \in A) = \int_A p(x)dx.$$

3.3 Rules of probability for PDFs

The rules of probability you learned in Chapter 1 for PMFs also hold for PDFs. For instance, consider independence/dependence.

Two variables x, y are said to be independent if

$$p(x = \mathsf{x}, y = \mathsf{y}) = p(x = \mathsf{x})p(y = \mathsf{y}) \tag{3.1}$$

for all $\mathsf{x} \in \Omega_x$ and $\mathsf{y} \in \Omega_y$. If this is not true, they are called dependent.

The above definition is the same for discrete and continuous variables. The difference is that in the discrete case $p(x), p(y)$ are PMFs, while they are PDFs for continuous variables.

We can also define a joint PDF of two (or more) variables

$$p(x = \mathsf{x}, y = \mathsf{y})$$

and then derive its marginal as

$$p(x = \mathsf{x}) = \int_{\Omega_y} p(x = \mathsf{x}, y = \mathsf{y})d\mathsf{y}.$$

and

$$p(y = \mathsf{y}) = \int_{\Omega_x} p(x = \mathsf{x}, y = \mathsf{y})d\mathsf{x}.$$

These formulas are the analogous of Equations (1.3)–(1.4). We only take into account that for continuous variables the possibility spaces Ω_x, Ω_y have infinite elements (think about the weight of fish cakes) and, therefore, the sum becomes an integral in this case. *Conditional PDFs* are defined in a similar way:

$$p(x = \mathsf{x}|y = \mathsf{y}) = \frac{p(x = \mathsf{x}, y = \mathsf{y})}{p(y = \mathsf{y})},$$

and

$$p(y = \mathsf{y}|x = \mathsf{x}) = \frac{p(x = \mathsf{x}, y = \mathsf{y})}{p(x = \mathsf{x})}.$$

The various forms of Bayes' rule can derived in a similar way:

$$\begin{aligned} p(x = \mathsf{x}|y = \mathsf{y}) &= \frac{p(x = \mathsf{x}, y = \mathsf{y})}{p(y = \mathsf{y})} \\ &= \frac{p(x = \mathsf{x}, y = \mathsf{y})}{\int p(x = \mathsf{x}, y = \mathsf{y}) d\mathsf{x}} \\ &= \frac{p(y = \mathsf{y}|x = \mathsf{x}) p(x = \mathsf{x})}{\int p(x = \mathsf{x}, y = \mathsf{y}) d\mathsf{x}} \\ &= \frac{p(y = \mathsf{y}|x = \mathsf{x}) p(x = \mathsf{x})}{\int p(y = \mathsf{y}|x = \mathsf{x}) p(x = \mathsf{x}) d\mathsf{x}} \end{aligned} \tag{3.2}$$

The notation $x = \mathsf{x}$ and $y = \mathsf{y}$ is quite heavy. The above rules can be denoted more compactly as:

$$\begin{aligned} p(x|y) &= \frac{p(x, y)}{p(y)} \\ &= \frac{p(x, y)}{\int p(\mathsf{x}, y) d\mathsf{x}} \\ &= \frac{p(y|x) p(x)}{\int p(\mathsf{x}, y) d\mathsf{x}} \\ &= \frac{p(y|x) p(x)}{\int p(y|\mathsf{x}) p(\mathsf{x}) d\mathsf{x}} \end{aligned} \tag{3.3}$$

However, note that, in all above definitions $p(x, y), p(x), p(y), p(x|y)$ are not probabilities: they are PDFs. To obtain probabilities, we need to compute *the area under the curve*, that is an integral. We will see examples in the next sections.

3.4 Gaussian distribution

There are many PDFs. When using a continuous probability distribution to model probability, the distribution used is selected to model and fit the particular situation in the best way. In this chapter, we will study the uniform distribution, the log-norm distribution, and the Gaussian distribution (also called normal distribution).

Gaussian (or normal) distribution has PDF of a continuous variable x that can take values in the possibility space $\Omega = \mathbb{R}$ (a real number). It is denoted as $N(x; \mu, \sigma)$ and defined as

$$p(x) = N(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)}{2\sigma^2}\right)$$

It has two parameters

- $\mu \in \mathbb{R}$ is called mean;
- $\sigma \in \mathbb{R}$ with $\sigma \geq 0$ is called standard deviation;

Note that σ^2 is called variance of x. A PDF is a function and so we can plot it.

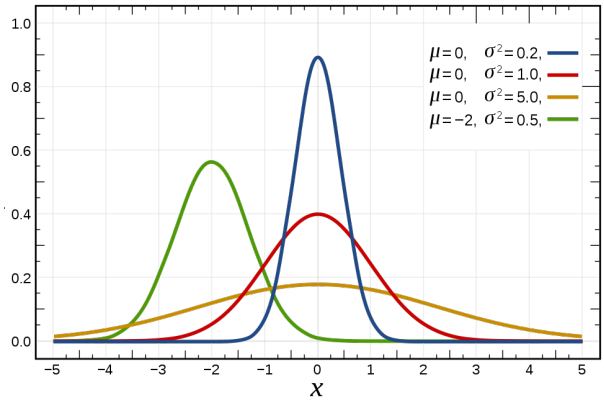


Figure 3.5: Gaussian PDF for different values of μ, σ^2

The Gaussian PDF is unimodal (that is it has a single maximum) and the value of x corresponding to the maximum is equal to the mean parameter μ.

To obtain the probability that $x \in [a, b]$, we need to compute

$$P(a \leq x \leq b) = \int_a^b \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)}{2\sigma^2}\right) dx$$

There is no known closed-form solution for the above integral, that means nobody knows how to solve it analytically. We can use a software to approximate this integral numerically (and the other integrals in this chapter).

Many statistics software packages implements integration routines to compute $P(x \leq b)$. Given this quantity we can compute

$$P(a \leq x \leq b) = P(x \leq b) - P(x \leq a).$$

$P(x \leq b)$ is called cumulative distribution function. Hereafter we report the general definition.

Definition: The **cumulative distribution function** (CDF) of x is defined by $P(x \leq \mathsf{x})$. It is a function of x that gives the probability that the variable x is less than or equal to x.

The Gaussian CDF is usually denoted as $\Phi(\mathsf{x})$. Figure 3.6 shows the Gaussian CDF for different values of μ, σ

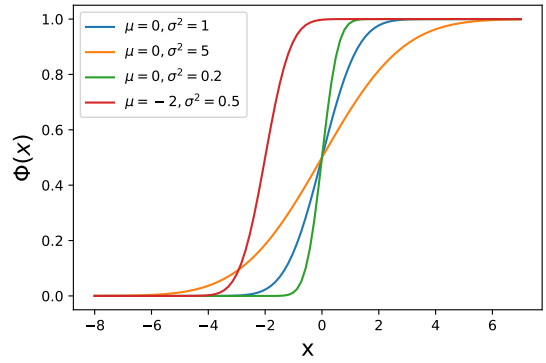


Figure 3.6: Gaussian PDF for different values of μ, σ^2

We can use the CDF to compute $P(a \leq x \leq b)$ as follows:

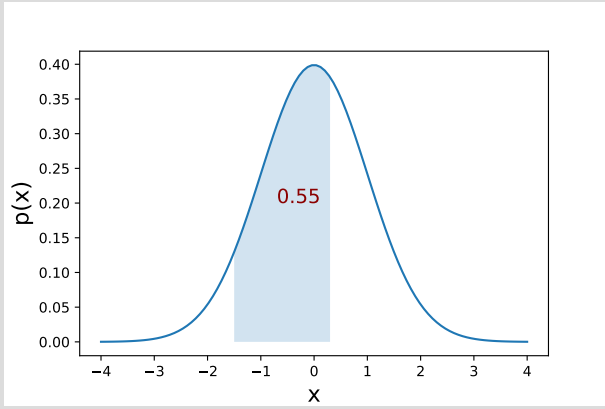
$$P(a \leq x \leq b) = \Phi(b) - \Phi(a).$$

The following code shows the use of the CDF to compute the area under the curve, that is $P(a \leq x \leq b)$.

Code

```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
mean=0.0
standard_deviation = 1.0
x_values = np.linspace(-4,4, 100)
dist = stats.norm(mean, standard_deviation)#normal
                                distribution
plt.plot(x_values, dist.pdf(x_values))
a=-1.5
b=0.3
x=np.linspace(a,b,30)
plt.fill_between(x,x*0,dist.pdf(x),alpha=0.2)
Prob=dist.cdf(b)-dist.cdf(a)#we use the CDF to compute
                                the probability
plt.text(-0.7,0.2,np.round(Prob,2),fontsize=14,c='
                                darkred')
```

```
plt.xlabel("x", fontsize=16);
plt.ylabel("p(x)", fontsize=16);
```



There is another way to compute integrals that is via **sampling**. This is the main approach we will use in the next chapters. A probability distribution is a generative model, meaning that we can generate a set of numbers that follows this probability distribution. Generally, any programming language or environment has a built-in random number generator for the most important probability distributions. For instance using Scipy, we can generate random numbers from a Gaussian distribution as follows:

Code

```
import scipy.stats as stats
dist = stats.norm(0, 1)#mean 0, standard deviation 1
print(dist.rvs(3))# we generate 3 random numbers
print("")
dist = stats.norm(-5, 1)#mean -5, standard deviation 1
print(dist.rvs(3))# we generate 3 random numbers
```

```
[-0.58409392,  1.21400745, -1.45271744]

[-5.58569326, -5.17342203, -4.66530743]
```

We can use these samples to approximately compute integrals. What is the probability that a number sampled from the standard normal distribution is in between -1.5 and 0.3? We can estimate this probability from our sample by counting the number of samples between -1.5 and 0.3, and dividing by the total number of

[27] This is basically the frequentist definition of probability: number of favourable cases to the event A (in this case the event is $x \in [-1.5, 0.3]$) divided by the total number of cases.

samples.[27] The following code compares $P(x \in [-1.5, 0.3])$ computed either using the CDF or using sampling. You can notice how the relative frequency:

$$Fr(x \in [-1.5, 0.3]) = \frac{\text{the number of samples between -1.5 and 0.3}}{\text{total number of samples}},$$

converges to the true probability (computed via the CDF) at the increasing of the number of samples.

Code

```
dist = stats.norm(0, 1)#mean 0, standard deviation 1
a=-1.5
b=0.3
Prob=dist.cdf(b)-dist.cdf(a)
print("Probability computed via CDF",Prob)
print("")
samples=dist.rvs(30)#we generate 30 random numbers from
                    dist
print("Probability computed via sampling",len(np.where((
                    a<=samples) & (samples<=b))[
                    0])/len(samples))
print("")
samples=dist.rvs(50)#we generate 50 random numbers from
                    dist
print("Probability computed via sampling",len(np.where((
                    a<=samples) & (samples<=b))[
                    0])/len(samples))
print("")
samples=dist.rvs(500000)#we generate 500000 random
                    numbers from dist
print("Probability computed via sampling",len(np.where((
                    a<=samples) & (samples<=b))[
                    0])/len(samples))
```

```
Probability computed via CDF 0.5511042

Probability computed via sampling 0.7

Probability computed via sampling 0.62

Probability computed via sampling 0.551654
```

We finish this section explaining why the parameters μ, σ^2 of a Gaussian distribution are called mean and variance.

Definition: The mean and variance of a PDF $p(x)$ are defined as:

$$\mu = \int_{\Omega} x\ p(x)dx$$
$$\sigma^2 = \int_{\Omega} (x-\mu)^2 p(x)dx$$

For the Gaussian PDF, one can verify that

$$\int_{-\infty}^{\infty} x \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)}{2\sigma^2}\right) dx = \mu$$
$$\int_{-\infty}^{\infty} (x-\mu)^2 \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)}{2\sigma^2}\right) dx = \sigma^2$$

We can verify that using sampling. The mean of the samples converges to μ at the increasing of the number of samples. The mean of $(\text{samples} - \mu)^2$ converges to the variance.

Code

```
dist = stats.norm(0, 1)#mean 0, standard deviation 1
samples=dist.rvs(500000)#we generate 500000 random
                        numbers from dist
print("Mean",np.sum(samples)/len(samples))
mu=np.sum(samples)/len(samples)
print("Variance",np.sum((samples-mu)**2)/len(samples))
dist = stats.norm(-2, 2)#mean 2, standard deviation 2
samples=dist.rvs(500000)#we generate 500000 random
                        numbers from dist
print("Mean",np.sum(samples)/len(samples))
mu=np.sum(samples)/len(samples)
print("Variance",np.sum((samples-mu)**2)/len(samples))
```

```
Mean -0.0035
Variance 1.002
Standard Deviation 1.0011
Mean -1.9996
Variance 4.0121
Standard Deviation 2.0002
```

3.4.1 Conditional probability

For discrete variables, we defined the probability of event B conditioned on knowing event A (or more shortly, the probability of B

given A) as

$$P(x \in B | x \in A) = \frac{P(x \in A \cap B)}{P(x \in A)} \tag{3.4}$$

The same definition holds for continuous variables. The only difference is that the above probabilities are computed by integrating the PDF.

$$P(x \in B | x \in A) = \frac{\int_{A \cap B} p(x)dx}{\int_A p(x)dx} \tag{3.5}$$

Example: Consider again the fish cakes example. The PDF in Figure 3.2 is a Gaussian PDF with mean 0.25 and standard deviation 0.05. This represents the uncertainty about the fish cakes weight. Let B be the event $w \in [0.2, 0.3]$, then

$$P(x \in B) = \int_{0.2}^{0.3} \frac{1}{\sqrt{2\pi 0.05^2}} \exp\left(-\frac{(x-0.25)}{2 \cdot 0.05^2}\right) = 0.68.$$

Let the event A be $w \geq 0.25$ (that is $w \in [0.25, \infty)$), the probability of B given A is:

$$P(x \in B | x \in A) = \frac{\int_{0.25}^{0.3} \frac{1}{\sqrt{2\pi 0.05^2}} \exp\left(-\frac{(x-0.25)}{2 \cdot 0.05^2}\right) dx}{\int_{0.25}^{\infty} \frac{1}{\sqrt{2\pi 0.05^2}} \exp\left(-\frac{(x-0.25)}{2 \cdot 0.05^2}\right) dx} = \frac{0.34}{0.5} = 0.68,$$

where we have exploited that the intersection of the interval $B = [0.2, 0.3]$ and $A = [0.25, \infty)$ is the interval $A \cap B = [0.25, 0.3]$.

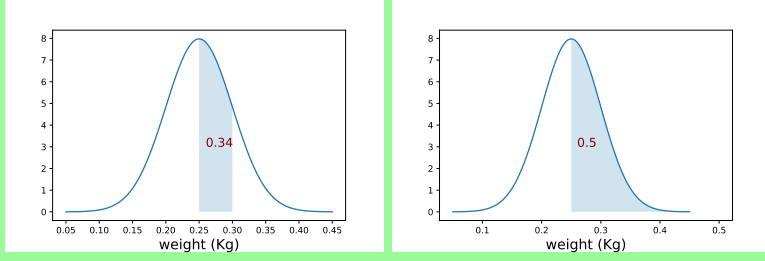


Figure 3.7: These probabilities (areas under the curve) can be computed either via the CDF or via sampling.

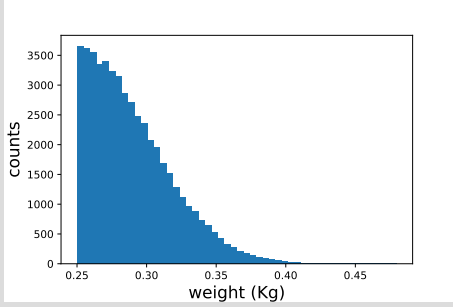
We can compute the above probabilities via sampling. We first implement the code using "Scipy":

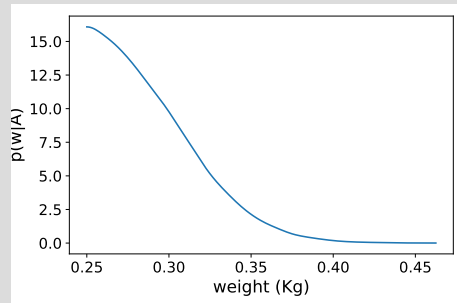
Code

```
import scipy as sp
```

```
import numpy as np

def event(x,interval):
    return np.logical_and(np.greater(x,interval[0]),np.
                                  less(x,interval[1]))

interval=[0.25,np.inf]# even A, x>0.25

dist = stats.norm(0.25, 0.05)#mean 0.25, standard
                              deviation 0.05
samples=dist.rvs(100000)#we generate 10000 random
                              numbers from dist
ind=event(samples,interval)#select the samples that
                              belong to the interval A
samples=samples[ind]
print("number of samples that are in [0.25,$\infty$):",
                              len(samples))
plt.hist(samples,50)#this is the posterior PDF
plt.xlabel("weight (Kg)",fontsize=16)
plt.ylabel("counts",fontsize=16)
plt.figure()
az.plot_dist(samples)#this is the posterior PDF
plt.xlabel("weight (Kg)",fontsize=16)
plt.ylabel("p(w|A)",fontsize=16)
print("P(B|A)=",np.sum(event(samples,[0.2,0.3]))/len(
                              samples))
```

```
number of samples that are in [0.25,infinity): 50047

P(B|A)=0.685036066097868
```





We will now go through each part line-by-line. First we defined a function *event* that returns True whenever the input x is between the extremes of the `interval`. Let us plot this function:

```
x=np.linspace(-1,1,100)
interval=[0.25,np.inf]
plt.plot(x,event(x,interval))
plt.xlabel("weight (Kg)",fontsize=16);
```

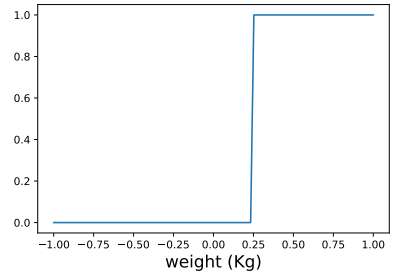


Figure 3.8: Indicator function of the set $[0.25, \infty)$

In the example, the event A is the interval $[0.25, \infty)$. The function *event* returns 1 when x is in the interval and 0 otherwise. In mathematics, this function is called **indicator function** of the set A.

Then we generated 100,000 samples from a Gaussian distribution with mean 0.25 and standard deviation 0.05. We then removed all the samples that are not in the interval (with the instructions `ind=event(samples,interval)`, `samples=samples[ind]`).[28] The remaining samples are posterior samples, in the sense that they now include the information that the event A has occurred. We can visualise the posterior using either a histogram or a density plot. To compute $P(B|A)$, we just need to count the posterior samples that are in the interval

[28] Remember that, given we have been told that A occurred, so any thing outside of A is not possible. We just have to consider the event A. The space of possibilities Ω has been reduced to A, that is why we remove all samples that are not in A.

$B = [0.2, 0.3]$ (the posterior samples take already into account the event A). We obtain 0.68.

Note that, only 50,000 samples out of the 100,000 generated samples satisfied the event A. What happens if we choose A small, for instance $A = [0.25, 0.26]$? If we run the above code again under this setting, then *number of samples that are in [0.25,26]* is about 8000. We have thrown away 92,000 samples. Since we are now using only 8000 samples to compute $P(B|A)$ (for instance consider $B = [0.25, 0.255]$), the value we compute is not a good approximation of the integral. In other words, the above approach is not very smart.

There are specialised libraries that can more efficiently sample from the posterior. We will use *PyMC3*, which is a Python library for probabilistic programming. It means it allows us to work with any probabilistic distribution (and their combination) and can be used to efficiently compute samples from the posterior. It is based on *theano* that, similarly to *tensorflow*, is a library for automatic differentiation. In *PyMC3*, we can write the above model as follows:

Code

```
import pymc3 as pm
import theano.tensor as tt
import arviz as az
def event_A(x,interval):
    return pm.math.log(pm.math.sigmoid((x-interval[0])*
                                5000.0)*pm.math.sigmoid
                                ((-x+interval[1])*5000.0
                                ))

interval=[0.25,np.inf]# event A, w>0.25
mymodel=pm.Model()
with mymodel:
    x = pm.Normal('x',0.25,0.05)
    pm.DensityDist('event',event_A,observed=dict(x=x,
                                interval=interval))

with mymodel:
    samples=pm.sample(30000,tune=10000, chains=1)
```

We will now go through each part line-by-line. First we defined a function *event_A*. We can plot this function

```
x=np.linspace(-0.5,1,100)
def event_A(x,interval):
    return pm.math.sigmoid((x-interval[0])*scale)*pm.
                                math.sigmoid((-x+
                                interval[1])*scale)
scale=50
```

```
plt.plot(x,event_A(x,interval).eval())
plt.xlabel("event A: [0.25,$\infty$)",fontsize=16)
plt.ylabel("sigmoid approximation",fontsize=16)
```

The function *event_A* approximates the indicator function with a sigmoid function.[29] The quality of the approximation is determined by the scale factor.

[29] Here the definition of sigmoid function https://en.wikipedia.org/wiki/Sigmoid_function.

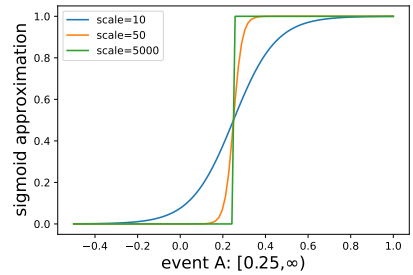


Figure 3.9: Approximation of the indicator function of the set $[0.25, \infty)$

Sigmoid functions are often called **soft indicators**, because they are smooth and can approximate the indicator function with arbitrary precision. For $scale = 5000$ the approximation is very good (the sigmoid is practically a step). Note that `pm.math.sigmoid` is the *Theano* implementation of the sigmoid logistic function $f(x) = \frac{1}{1+e^{-x}}$. `event_A(x,interval)` returns a *Theano* expression, to evaluate it we need to run the command *eval* as in `event_A(x,interval).eval()`. This returns a number for each value of x.

The advantage of the sigmoid approximation of the indicator function is that now we can rank samples. For instance, consider the samples -1 and -0.1. They are both not included in A (we should reject them), but `event_A(-0.1,interval)` is greater than `event_A(-1,interval)`. This tells us that -0.1 is closer to A and, therefore, we can think of an algorithm that uses this distance to smartly generate samples that are in A or very close to A. This is what *PyMC3* does internally. `event_A(x,interval)` can return very small numbers and so to avoid underflow problems we need to use the log-scale:

```
def event_A(x,interval):
    return pm.math.log(pm.math.sigmoid((x-interval[0])*
                                       scale)*pm.math.sigmoid((
                                       -x+interval[1])*scale))
```

where `pm.math.log` is the *Theano* implementation of the logarithm.[30]

We can now move to the block:

```
mymodel= pm.Model()
with mymodel:
    x = pm.Normal('x',0.25,0.05)
    pm.DensityDist('event',event_A,observed=dict(x=x,
                                interval=interval))
```

The first line, `mymodel = pm.Model()` creates a new Model object which is a container for the subsequent instructions. Following instantiation of the model, the subsequent specification of the model components is performed inside a `with` statement: `with mymodel:`. This creates a context manager, with `mymodel` as the context, that includes all statements until the indented block ends. This means all PyMC3 objects introduced in the indented code block below the `with` statement are added to the model. In particular, the statement `x = pm.Normal('x',0.25,0.05)` creates a variable x which is called "x" and is Gaussian (Normal) distributed[31] with mean 0.25 and standard deviation 0.05. The subsequent statement

`pm.DensityDist('event',event_A,`
`observed=dict(x=x, interval=interval)`

is a custom probability distribution that simply calls the external function `event_A` passing the argument x and *interval* as a dictionary `dict(x=x, interval=interval)`. The statement `observed` tells PyMC3 that it must compute a posterior distribution using `event_A` as a soft indicator.

Finally, the block

```
with mymodel:
    samples=pm.sample(30000,tune=1000, chains=1)
```

samples 31,000=30,000+1,000 samples from the posterior. The algorithm[32] that computes the posterior samples is randomly initialised and, therefore, the result may depend on the initialisation. To prevent that, we can discard the first 1,000 samples (the number of samples to use for tuning is 1,000 in the above example).[33] This is done automatically and so PyMC3 returns 30,000 samples at the end. A chain is a single run of the algorithm. When running MCMC, it is a best practice to use multiple chains, as they can help diagnose convergence problems. We will go back to this aspect later on.

This is the output of the code:

Code

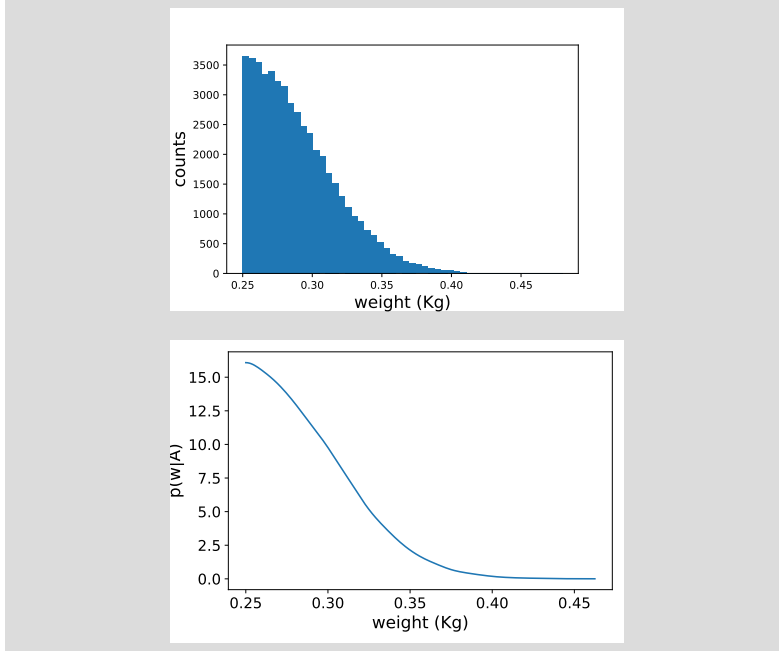
number of samples that are in [0.25,∞): 30000
P(B|A)=0.6801

[30] PymC3 expects this output to be in log-scale.

[31] That is it has a Gaussian PDF.

[32] The algorithm is a type of Markov chain Monte Carlo sampler https://en.wikipedia.org/wiki/Markov_chain_Monte_Carlo.

[33] Tune does other things, but for us it is just an initialisation step.



and you can see that now we have all the 30,000 samples.

3.4.2 More than one variable

Figure 3.10: https://en.wikipedia.org/wiki/Fishcake

Consider again the fish cakes example. Assume that instead of their weight, now we care about their size: length x and width y in cm. They are both continuous variables. We assume that on average the length and width are both $8cm$, but there is a "small" variability (dispersion). The two marginal PDFs $p(x)$ and $p(y)$ of length and, respectively, width

$$\begin{aligned} p(x) &= N(x; 8, 0.5) \\ p(y) &= N(y; 8, 0.5) \end{aligned} \tag{3.6}$$

are depicted in the following figure.

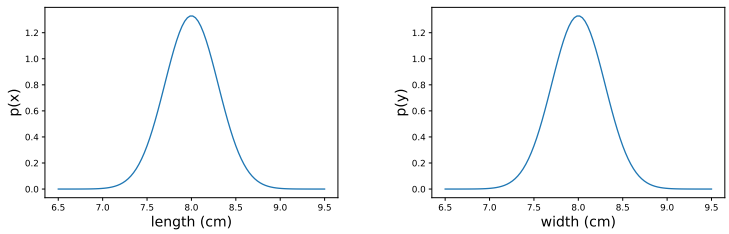


Figure 3.11: Marginal PDFs for length and width.

We are interested in joint PDF that allows us to compute probabilities of events involving both variables and understand the relationship between the variables. This is simplest when the variables are independent. In this case the joint PDF is simply the product of the marginals:

$$p(x, y) = N(x; 8, 0.5)N(y; 8, 0.5).$$

We can plot this 2D function: the top figure is the 2D plot and the bottom one is the respective contour plot.[34]

[34] see https://jakevdp.github.io/PythonDataScienceHandbook/04.04-density-and-contour-plots.html.

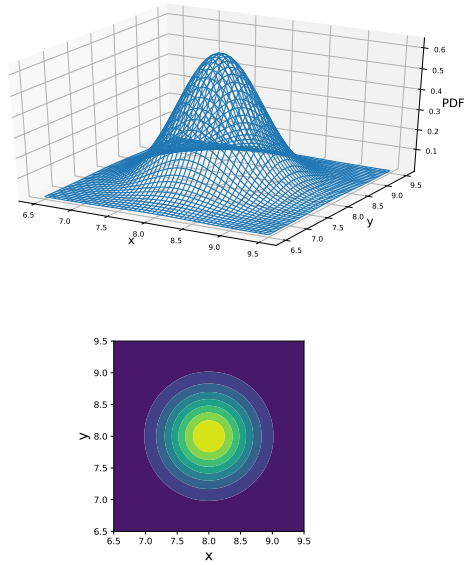


Figure 3.12: PDF of length and width in the independence case.

For length and width of fish cakes, independence is not true. The raw material is weighed, divided in portions and then shaped.

Therefore, we expect that when the length is above its nominal value 8cm, then width must below the nominal value 8cm and vice versa. The two variables are dependent.

Definition: Let x, y be continuous variables with domain $\text{dom}(x) = \text{dom}(y) = \mathbb{R}$, we say that the two variables are jointly Gaussian (or Normal) distributed when their joint PDF is

$$p(x, y) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} e^{-\frac{v}{2(1-\rho^2)}},$$

with

$$v = \frac{(x-\mu_1)^2}{\sigma_1^2} - \frac{2\rho(x-\mu_1)(y-\mu_2)}{\sigma_1\sigma_2} + \frac{(y-\mu_2)^2}{\sigma_2^2}$$

where μ_1, σ_1^2 are the mean and variance of the variable x, μ_2, σ_2^2 are the mean and variance of the variable y and $\rho \in (-1, 1)$ is the correlation parameter. For $\rho = 0$, we have independence and in fact we can verify that:

$$p(x, y) = N(x; \mu_1, \sigma_1)N(y; \mu_1, \sigma_1).$$

while for $\rho \neq 0$ the variables are dependent.

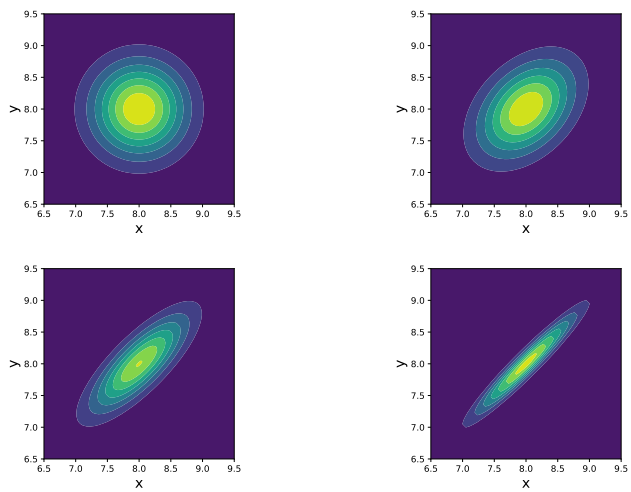


Figure 3.13: Joint Gaussian for ρ equal to 0 (top-left), 0.4 (top-right), 0.8 (bottom-left) and 0.96 (bottom-right).

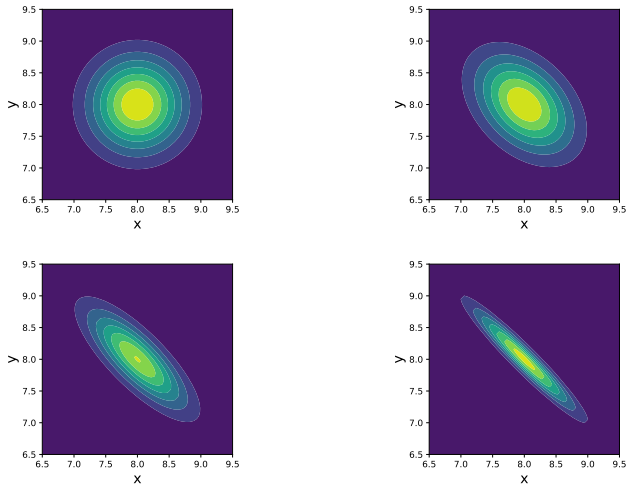


Figure 3.14: Joint Gaussian for ρ equal to 0 (top-left), -0.4 (top-right), -0.8 (bottom-left) and -0.96 (bottom-right).

We can sample from a joint Gaussian distrbution using "Scipy". First we must define the so called *covariance matrix*:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2\rho \\ \sigma_1\sigma_2\rho & \sigma_2^2 \end{bmatrix}$$

Code

```
from scipy.stats import multivariate_normal
mu1=8;mu2=8;
sigma1=0.5;sigma2=0.5;rho=-0.1
Sigma=np.array([[sigma1**2, sigma1*sigma2*rho], [sigma1*
                            sigma2*rho, sigma2**2]])
rv = multivariate_normal(np.array([mu1, mu2]), Sigma)
print(rv.rvs(5))#generate 5 samples
print(" ")
rho=-0.96
Sigma=np.array([[sigma1**2, sigma1*sigma2*rho], [sigma1*
                            sigma2*rho, sigma2**2]])
rv = multivariate_normal(np.array([mu1, mu2]), Sigma)
print(rv.rvs(5))#generate 5 samples
```

```
[[8.40469192 7.7901591 ]
 [8.02543562 8.19595529]
 [7.45203008 8.03929828]
 [7.15711687 8.24776802]
 [7.98334769 8.00491533]]
```

```
[[7.68162502 8.0748369 ]
 [7.06329723 8.81296076]
 [8.95384252 7.08512699]
 [7.64934523 8.42402029]
 [7.10482331 8.91477652]]
```

It returns a 2D matrix with 5 (number of samples) rows and 2 columns. The fist column is the x-variable and the second the y-variable. Each row is a joint sample from the posterior. Note that, in the second case, whenever x is greater than 8, then y is lower than 8 and vice versa. This is the dependence model we discussed previously. That is, when the length (x) is above its nominal value 8cm, then width (y) must below the nominal value 8cm and vice versa.

3.5 Exercises without solutions

Exercise 1 The PDF of the uniform distribution in the interval $[c, d]$ with $c, d \in \mathbb{R}$ and $c \leq d$, is

$$p(x) = \begin{cases} \frac{1}{d-c} & x \in [c, d] \\ 0 & x \notin [c, d], \end{cases}$$

and its plot is

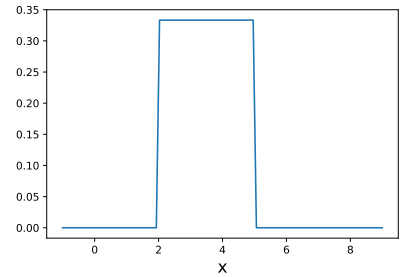


Figure 3.15: Uniform PDF for $[c, d] = [2, 5]$

A Uniform distribution tells us that all values in the interval $[2, 5]$ are equally possible.

In this case, we can compute analytically the probability that $x \in [a, b]$ with $c \leq a \leq b \leq d$

$$P(a \leq x \leq b) = \int_a^b \frac{1}{d-c} dx = \frac{b-a}{d-c}.$$

Consider the case $[c, d] = [2, 5]$.

1. Compute $P(a \leq x \leq b)$ for $[a, b] = [2.5, 3.5]$ via sampling using "Scipy". The uniform distribution in Scipy is

```
import numpy as np
import scipy.stats as stats
c=2
d=5
dist = stats.uniform(c,d-c)
```

2. Compute the mean of the variable x and the variance via sampling using "Scipy".
3. Compute $P(a \leq x \leq b | x \leq 4)$ for $[a, b] = [2.5, 3.5]$ via sampling using "PyMC3" (adapt the code we used for the Gaussian distribution).
4. Compute the posterior mean and variance of x given the event $x \leq 4$ (use the posterior samples you generated for answering the previous question).

Exercise 2 In many practical application, we deal with transformation of variables. For instance, let x be a variable Gaussian distributed with zero mean and standard deviation 1. Let μ and $\sigma > 0$ be two real numbers. We are interested on the transformed variable:

$$y = e^{\mu + \sigma x}$$

which is always nonnegative.

Consider the case $\mu = -1.0, \sigma = 0.75$.

1. Compute $P(0 \leq y \leq 0.4)$ via sampling using "Scipy":

```
import numpy as np
import scipy.stats as stats
dist = stats.norm(0,1)
x = dist.rvs(1000)
y=np.exp(mu+sigma*x)
```

2. Compute the mean of the variable y and the variance via sampling using "Scipy".
3. Plot the distrbution of y using either a histogram or a density plot.
4. Compute $P(0 \leq y \leq 0.4 | y \leq 1)$ via sampling using "PyMC3" (adapt the code we used for the Gaussian distribution).[35]
5. Compute the posterior mean and variance of y given the event $y \leq 1$ (use the posterior samples you generated for answering the previous question).

[35] See this https://docs.pymc.io/notebooks/api_quickstart.html#Deterministic-transforms to understand how you can write the transformation $y = np.exp(mu + sigma * x)$ in PyMC3.

Chapter 4

Learning II

In Chapter 2 we derived the Maximum Likelihood Estimator (MLE) for the multinomial distribution. The MLE is general and can be applied to any probability distribution (discrete or continuous variables). In this chapter, we will first derive the MLE for the parameters of the Gaussian distribution. We will again discuss the limitations of the MLE approach presented in Chapter 2, including the limitations of reguralisation. Finally, we will introduce a more principled approach to learning. We will finish this Chapter by applying it to linear regression.

4.1 MLE of the parameters of a Gaussian PDF

Assume we bought and weighed 10 fish cakes, resulting in the dataset

$$D = \{0.15, 0.21, 0.22, 0.18, 0.27, 0.29, 0.26, 0.12, 0.25, 0.24\}.$$

We assume that the weight is Gaussian distributed, that is

$$p(w|\mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{w-\mu}{\sigma}\right)^2}.$$

where we used the conditional PDF $p(w|\mu, \sigma)$ to point out that the PDF depends on the parameters μ, σ. We assume that these parameters are unknown and our goal is to use the above dataset to estimate them.

We first compute the MLE of these parameters. Assuming independence between the instances of the dataset (given the unknown parameters μ, σ). The likelihood of seeing all the data is[36]

[36] The symbol $\prod_{i=1}^{n} f(i)$ denotes product, and it is equal to $f(1)f(2)\cdots f(n)$.

$$L(D, \mu, \sigma) = \prod_{i=1}^{n} p(w_i|\mu, \sigma)$$

where n is the number of instances in our dataset ($n = 10$ in the example).

Viewing this as a function of the parameters,

$$L(D, \mu, \sigma) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{w_i-\mu}{\sigma}\right)^2}.$$

it tells us how likely it is the production process of fish cakes has parameters μ and σ given the dataset D. The *maximum likelihood estimator* for the parameters are the values $\hat{\mu}$ and $\hat{\sigma}$ that maximize $L(D, \mu, \sigma)$

It is convenient (and equivalent) to maximize the "log-likelihood" $l = \ln L(D, \mu, \sigma)$. We have

$$l = \sum_{i=1}^{n} \left[\ln \frac{1}{\sqrt{2\pi}} - \ln \sigma - \frac{1}{2} \left(\frac{w_i - \mu}{\sigma} \right)^2 \right].$$

To compute the maximum we compute the first derivative with respect to each parameter μ, σ^2. Note that

$$\frac{\partial l}{\partial \mu} = \frac{1}{\sigma^2} \sum_{i=1}^{n} (w_i - \mu),$$

so equating the derivative to zero $\partial l/\partial \mu = 0$, then $\mu = \sum_{i=1}^{n} w_i/n$. It is not hard to check this is indeed a maximum for l, so the maximum likelihood estimator for the mean of a Gaussian is $\hat{\mu} = \sum_{i=1}^{n} w_i/n$, the average of the sample.

With a similar derivation, we can see that the MLE for the variance is $\hat{\sigma^2} = \sum_{i=1}^{n} (w_i - \hat{\mu})^2/n$.

For the dataset above, one has

$$\hat{\mu} = 0.219, \quad \hat{\sigma}^2 = 0.002689$$

and so $\sigma = 0.05189$.

4.2 Reguralisation

Regularization is the process of introducing additional information in order to prevent issues with MLE. We have already seen an issue with MLE: it can return sharp estimates after seeing few instances.

For instance, consider the dataset

$$D = \{0.24, 0.24, 0.24\}$$

the MLE estimate of the variance is $\hat{\sigma}^2 = 0$ and, therefore, MLE predicts with certainty that all fish cakes weigh 0.24Kg.

To prevent that we can use reguralisation, as we did in Chapter 2. Once again the idea is to multiply the likelihood for a weighting function[37] of the parameters:

$$L(D, \mu, \sigma)p(\mu, \sigma) = \left(\prod_{i=1}^{n} p(w_i|\mu, \sigma)\right) p(\mu, \sigma) \tag{4.1}$$

[37] A nonnegative function of the parameters that express some prior knowledge on the parameters. It is usually a PDF.

Our goal is to maximise the above function to determine an estimate of the parameters μ, σ. It is convenient (and equivalent) to maximize the logarithm:

$$\ln(p(\mu, \sigma)) + \sum_{i=1}^{n} \ln(p(w_i|\mu, \sigma))$$

Consider again the fish cakes example, in this case:

$$l = \sum_{i=1}^{n} \left[\ln \frac{1}{\sqrt{2\pi}} - \ln \sigma - \frac{1}{2} \left(\frac{w_i - \mu}{\sigma} \right)^2 \right] + \log(p(\mu, \sigma))$$

and assume that

$$p(\mu, \sigma) = p(\mu)p(\sigma)$$

with $p(\mu), p(\sigma)$ showed in Figure 4.1. In this example the reguralisation function for μ is a Gaussian with mean 0.25 and standard deviation 0.1: this models our knowledge that fish cakes are advertised to weigh 0.25 kg and that there is some variability. The reguralisation function for σ is instead Uniform in the interval $[0.0001, 0.3]$ and allows us to exclude values that are smaller than 0.001 (avoiding the above mentioned problem with MLE) and larger than 0.3.[38]

[38] One could choose different reguralisation functions, the choice of the reguralisation function should reflect the prior information we have about the data.

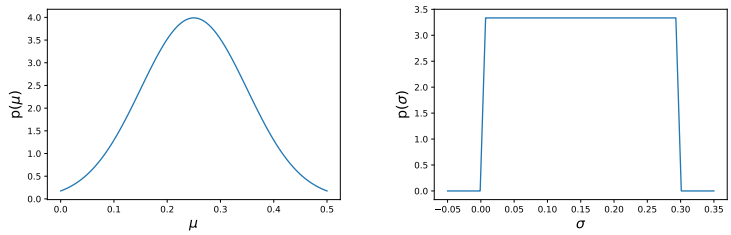


Figure 4.1: Weighting function for μ and σ

To compute the value of μ, σ that maximise Equation (4.1) we can use the PyMC3:

Code

```
import pymc3 as pm
mymodel=pm.Model()
with mymodel:
    mu = pm.Normal('mu',0.25,0.1)
    sigma = pm.Uniform('sigma',0.0001,0.3)
    pm.Normal('Like',mu,sigma,observed=D)

with mymodel:
    estimate=pm.find_MAP()
print(estimate['mu'],estimate['sigma'])
```

```
0.219, 0.0519
```

We will now go through each part line-by-line. The first two lines `mu = pm.Normal('mu',0.25,0.1)` `sigma = pm.Uniform('sigma',0.0001,0.3)` define the two reguralisation function for μ, σ: we say to PyMC3 that the variable mu is Gaussian and the variable `sigma` is Uniform distributed. `pm.Normal('Like',mu,sigma,observed=D)` is instead the likelihood. It tells us that all observations (that is the elements of D) are Gaussian distributed with mean μ and standard deviation σ. This is exactly the model in Equation (4.1). Finally `pm.find_MAP()` solves the optimisation problem and returns a dictionary including the optimum value of μ and σ.[39] The obtained values are very close to the MLE estimates (meaning the contribution of the weighting functions is small).

[39] The value of μ, σ that maximise Equation (4.1) is called Maximum A-Posteriori (MAP) estimate. We will explain later why.

Note that, for $D = \{0.24, 0.24, 0.24\}$, the MAP estimate is

Code

```
import pymc3 as pm
D=np.array([0.24,0.24,0.24])
mymodel=pm.Model()
with mymodel:
    mu = pm.Normal('mu',0.25,0.1)
    sigma = pm.Uniform('sigma',0.0001,0.3)
    pm.Normal('Like',mu,sigma,observed=D)

with mymodel:
    estimate=pm.find_MAP()
print(estimate['mu'],estimate['sigma'])
```

```
0.24, 0.0001
```

and so the value of σ is not zero anymore. Adding reguralisation prevents "overfitting" when dealing with small datasets.

4.3 Another way of learning

The reason we introduced reguralised MLE (also called MAP) is to prevent issues with small datasets.[40] We will now show that reguralised MLE (and also MLE) have another big issue, they cannot distinguish between small datasets and large datasets. In other words, they cannot represent uncertainty.

[40] Overfitting is usually a problem when the dataset is small (compared to the number of feature).

Before we have seen that for the dataset

$$D_1 = \{0.15, 0.21, 0.22, 0.18, 0.27, 0.29, 0.26, 0.12, 0.25, 0.24\}.$$

the MLE estimate of μ, σ is $\hat{\mu} = 0.219,\ \hat{\sigma} = 0.0519$ and the reguralised MLE (MAP) is the same.

Consider now the following dataset

$$\begin{aligned} D_2 = \{&0.212, 0.188, 0.367, 0.1886, 0.1595, 0.1766, 0.1787, \\ &0.1786, 0.231, 0.2335, 0.2599, 0.2136, 0.2114, 0.1825, \\ &0.3127, 0.1861, 0.2161, 0.2534, 0.2699, 0.1528\} \end{aligned}$$

You can verify that the MLE estimate of μ, σ is again $\hat{\mu} = 0.219, \hat{\sigma} = 0.0519$ and the reguralised MLE (MAP) is the same.

The dataset D_1 has size 10, while the dataset D_2 has size 20. From the MLE and reguralised MLE estimates ($\hat{\mu} = 0.219,\ \ \hat{\sigma} = 0.0519$), we do not have any way to distinguish the two cases. A larger dataset should provide a more reliable estimates, but MLE and reguralised MLE cannot quantify this reliability (or equivalently the uncertainty).

We will now introduce a different and more general way of learning that solves this issue. Let us go back to Equation (4.1):

$$\left(\prod_{i=1}^{n} p(w_i|\mu, \sigma)\right) p(\mu, \sigma)$$

which we can write as[41]

$$p(data|\mu, \sigma)p(\mu, \sigma)$$

[41] We have seen that $\prod_{i=1}^{n} p(w_i|\mu, \sigma)$ is the likelihood, that is $p(data|\mu, \sigma)$.

Given the above two PDFs we can apply Bayes' rule and compute

$$p(\mu, \sigma|data) = \frac{p(data|\mu, \sigma)p(\mu, \sigma)}{\int_{\Omega_\mu}\int_{\Omega_\sigma} p(data|\mu, \sigma)p(\mu, \sigma)d\mu d\sigma} \tag{4.2}$$

which is the posterior distribution of μ, σ given the data. The posterior distribution is the complete answer to the question: what is the value of μ, σ? The posterior distribution is a probability distribution that represents our updated beliefs about the parameters

μ, σ after having seen the data. It is a probability distribution and, therefore, it represents the uncertainty that is both present in the likelihood (and so in the data) and in the prior knowledge $p(\mu, \sigma)$.

We have already seen how to compute samples from the posterior in PyMC3:

Code

```
mymodel=pm.Model()
with mymodel:
    mu = pm.Normal('mu',0.25,0.1)
    sigma = pm.Uniform('sigma',0.0001,0.2)
    pm.Normal('Like',mu,sigma,observed=D)

with mymodel:
    samples=pm.sample(10000,tune=3000,chains=1)
```

```
0.24, 0.0001
```

As we did in Chapter 3, we ask PyMC3 to sample from the posterior of the probabilistic model that has prior over the parameters `mu = pm.Normal('mu',0.25,0.1)`, `sigma = pm.Uniform('sigma',0.0001,0.3)` and likelihood `pm.Normal('Like',mu,sigma,observed=D)` (PyMC3 understands that the latter is a likelihood because of the keyword "observed" that has as argument the dataset.).

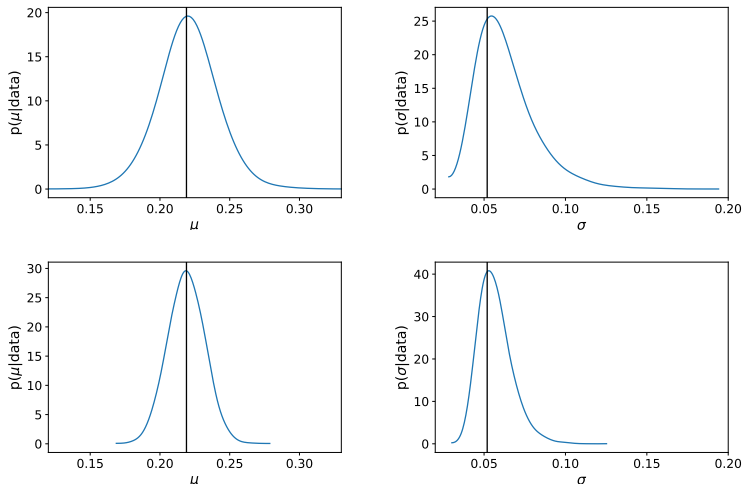


Figure 4.2: Posterior of $p(\mu|D_1)$ and $p(\sigma|D_1)$ (first row), and $p(\mu|D_2)$ and $p(\sigma|D_2)$ (second row).

The posteriors are showed in Figure 4.2: $p(\mu|D_1)$ and $p(\sigma|D_1)$ (first row), and $p(\mu|D_2)$ and $p(\sigma|D_2)$ (second row). The blue curves represent our beliefs about the value of μ, σ after seeing the data. We can immediately see that the posterior computed using the dataset D_2 are more concentrated (there is less dispersion). The posteriors tell us that we are less uncertain about the value of μ, σ when we observe the larger dataset D_2. The vertical lines correspond to the MAP estimates.[42] The MAP is the same in the two cases (compare first and second row), but the posteriors are significantly different. The posteriors represent the uncertainty.

[42] This explains why the reguralised MLE is called MAP, because it returns the values of the parameters that maximise the posterior.

The standard learning approach in Machine Learning is based on either

- MLE,

or

- reguralised MLE.

This is true for linear regression, logistic regression but also for neural networks, deep neural networks and any other standard Machine Learning algorithm. This means that standard Machine Learning cannot represent uncertainty.

Result: Given two datasets D_1 and D_2, assume that D_2 is larger than D_1 and denote with q_1 and q_2 two predictions (for the same input) of a Machine Learning algorithm trained with D_1 and, respectively, D_2. Standard Machine Learning algorithms cannot tell that the prediction q_2 is less uncertain than q_1.

Representing uncertainty is fundamental in decision making (and so in applications) because it allows us to quantify the risk of a decision (prediction).

4.4 Application: sales forecast

Linear Regression is an algorithm that aims to predict a dependent variable value (y) based on a given independent variable (x). In particular, it finds out a linear relationship between x (input) and y (output):

$$y = \beta x + \alpha.$$

with $\alpha, \beta \in \mathbb{R}$. Assume that x represents yearly money invested in TV advertisements and y the sales of a product:[43]

[43] From the dataset "Advertising.csv".

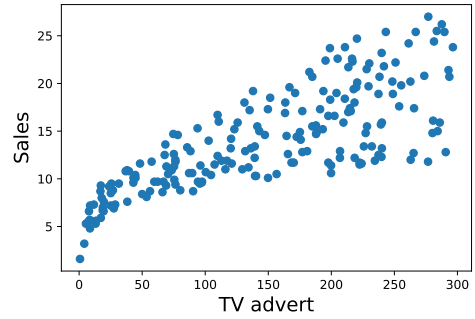


Figure 4.3: TV adverts vs. sales

In linear regression, given the dataset $D = \{(x_i, y_i) : i = 1, \ldots, n\}$ (the points in the scatter plot), the parameters α, β (intercept and slope) are determined by minimising the squared-error loss function:[44]

[44] This approach to obtain α, β is also called least squares method.

$$\min_{\alpha,\beta} \sum_{i=1}^{n} (y_i - x_i\,\beta - \alpha)^2$$

To compute the minimum we compute the first derivative with

respect to each parameter α, β. Note that

$$\frac{\partial(\cdot)}{\partial\beta} = -2\sum_{i=1}^{n}(y_i - x_i\,\beta - \alpha)x_i$$

and

$$\frac{\partial(\cdot)}{\partial\alpha} = -2\sum_{i=1}^{n}(y_i - x_i\,\beta - \alpha)$$

so equating the derivatives to zero we obtain

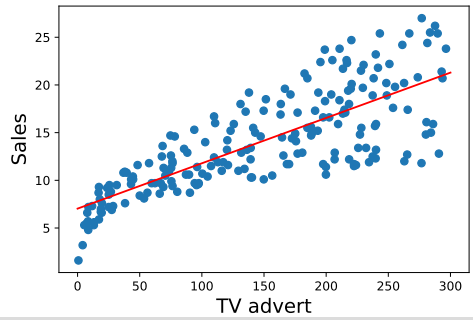
$$\hat{\beta} = \frac{\sum_{i=1}^{n} x_i y_i - \bar{y}x_i}{\sum_{i=1}^{n} x_i^2 - \bar{x}x_i}, \qquad \hat{\alpha} = \bar{y} - \hat{\beta}\bar{x},$$

where $\bar{y} = (\sum_{i=1}^{n} y_i)/n$ and $\bar{x} = (\sum_{i=1}^{n} x_i)/n$. This is what "Sklearn" computes when we perform linear regression. The estimated regression line $\hat{\alpha} + \hat{\beta}x$ is the prediction at x. For instance, it tells us that if our investment in TV advertisements goes from 105 to 200 the number of sales should increase from 12 to 16 (hundreds).

Code

```
import numpy as np
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(x.reshape(-1,1), y)
print("beta=",reg.coef_[0],"alpha=",reg.intercept_)
print("prediction at x=105:", reg.predict(np.array([[105
                            ]])))
print("prediction at x=200:", reg.predict(np.array([[200
                            ]])))
xx=np.linspace(0,300)
plt.plot(xx,xx*reg.coef_[0]+reg.intercept_,c='r')#
                            regression line
plt.scatter(x,y)
```

```
beta= 0.04753664043301975 alpha= 7.032593549127695
prediction at x=105: [12.02394079]
prediction at x=200: [16.53992164]
```



4.4.1 MLE derivation of least squares

We can derive the same estimates via MLE. In this case, the difference is in the probabilistic interpretation of the error $y_i - \alpha - \beta x_i$.

MLE starts from the premise that, although the relation between x and y may be linear, the data may not follow it exactly. The main reason is that the observation mechanism is usually **imperfect**.

This means that

$$y_{observed} \neq y_{real}$$

We account for the imperfect observation mechanism with a probabilistic model

$$p(y_{observed}|y_{real})$$

In linear regression, it is usually assumed that this PDF is Gaussian:

$$p(y_{observed}|y_{real}) = N(y_{observed}; y_{real}, \sigma) = N(y_{observed}; \beta\, x + \alpha, \sigma)$$

We usually denote $y_{observed}$ simply with y.

For a single observation

$$p(y|y_{real}) = p(y|x, \alpha, \beta, \sigma) = N(y; \beta\, x + \alpha, \sigma)$$

We have n observations (training data):

$$D = \{(x_i, y_i) :\ i = 1, \ldots, n\}$$

Assuming the observations are **conditional independent** given the parameters,

$$p(D|\alpha, \beta, \sigma) = p(D|\theta) = \prod_{i=1}^{n} N(y_i; \beta\, x_i + \alpha, \sigma)$$

where we have used $\theta = [\alpha, \beta, \sigma]$ to compactly denote all the parameters, we are explicitly modelling the fact that the observations are noisy and, therefore, we need an additional parameter σ.

There is an equivalent way to derive the least squares estimates for linear regression, that is by maximisiming the likelihood (that is MLE). That is, the solutions of these two optimisation problems

$$\min_{\alpha,\beta} \sum_{i=1}^{n} (y_i - \beta\, x_i - \alpha)^2$$

and

$$\max_{\alpha,\beta,\sigma} p(D|\alpha, \beta, \sigma) = \max_{\alpha,\beta,\sigma} \prod_{i=1}^{n} N(y_i; \beta\, x_i + \alpha, \sigma)$$

is the same. This is the proof.

Since for any function maximisiming the function or its logarithm is the same, we can more conveniently maximise the logarithm of the likelihood:

$$\log\left(\prod_{i=1}^{n} N(y_i; \beta\, x_i + \alpha, \sigma)\right) = \sum_{i=1}^{n} \log\left(N(y_i; \beta\, x_i + \alpha, \sigma)\right)$$
$$= -\frac{n}{2}\log(2\sigma^2\pi) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta\, x_i - \alpha)^2$$

The value of α, β, σ that maximises the above negative function is equal to the value that minimizes

$$\frac{n}{2}\log(2\sigma^2\pi) + \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta\, x_i - \alpha)^2$$

The minimum with respect to α, β can be obtained by solving $\min_{\alpha,\beta} \sum_{i=1}^{n}(y_i - \beta\, x_i - \alpha)^2$, which is equivalent to minimizing the squared loss.

We have shown that the squared loss estimator is a MLE estimator.

4.4.2 A better way to learn α, β

We have seen that MLE cannot represent uncertainty. Can we compute the posterior

$$p(\alpha, \beta|D) = ?$$

This is better than MLE (or, equivalently, least squares) because the posterior tells us about the uncertainty on α, β.

Yes we can and it is very similar to the model we have seen before for the fish cakes.

Code

```
x=data['TV'].values
y=data['Sales'].values
regmodel=pm.Model()
with regmodel:
    alpha = pm.Normal('alpha',0.0,30)
    beta  = pm.Normal('beta',0.0,30)
    sigma = pm.Uniform('sigma',0.0001,30)
    mu = beta*x+alpha
    pm.Normal('Like',mu,sigma,observed=y)

with regmodel:
    samples=pm.sample(10000,tune=3000,chains=1)
print(samples[0])
print(samples[1])
print(samples[5000])
```

```
{'alpha': 7.42, 'beta': 0.044,  'sigma': 2.99}
{'alpha': 7.91, 'beta': 0.043,  'sigma': 3.01}
{'alpha': 6.59, 'beta': 0.0483, 'sigma': 3.386}
```

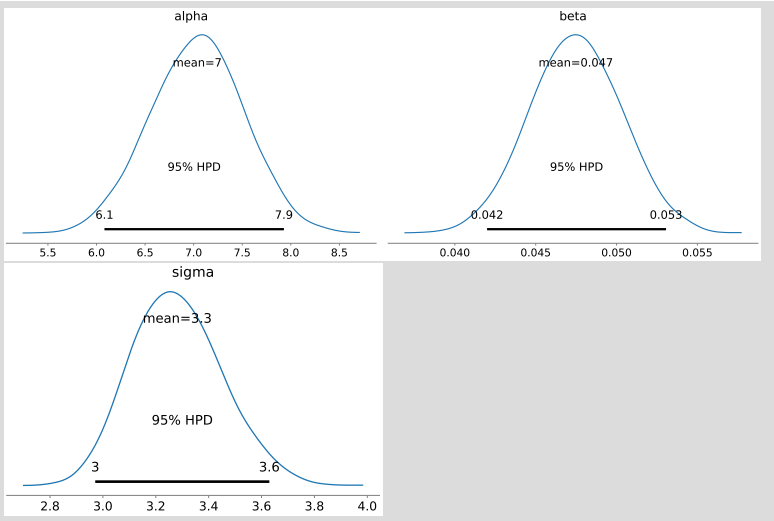
This model is very similar to the one we have defined previously in this chapter for estimating μ, σ. The difference is that μ is a function of x, that is $\mu = \beta x + \alpha$. The variable x is observed (the x are the points in the dataset), but we do not know α, β. We model our uncertainty with a Gaussian distribution with zero mean and large standard deviation to represent our prior uncertainty about α, β. We then combine it with the $p(\text{data}|\alpha, \beta, \sigma)$, that is the likelihood to obtain the posterior.

$$\underbrace{p(\alpha, \beta, \sigma|\text{data})}_{\text{posterior}} = \frac{\underbrace{p(\text{data}|\alpha, \beta, \sigma)}_{\text{likelihood}} \underbrace{p(\alpha, \beta, \sigma)}_{\text{prior}}}{\underbrace{p(\text{data})}_{\text{evidence}}}$$

The above code returns samples from the posterior $p(\alpha, \beta, \sigma|\text{data})$. More precisely, `samples` is a list of dictionaries. Each element of the list is a different sample from the posterior. We can visualise the posterior of alpha, beta and sigma by using a density plot.

Code

```
import arviz as az
az.plot_posterior(samples, var_names=['alpha','beta'],
                           credible_interval=0.95)
az.plot_posterior(samples, var_names='sigma',
                           credible_interval=0.95)
```



The maxima of the PDFs of alpha and beta correspond to the least squares estimate computed by least-squares (7 and respectively 0.47), but there is a lot of uncertainty especially in the intercept parameter α.[45] Moreover, we have an additional information that is the posterior of σ ("Sklearn" does not return this parameter). This is a measure of the noise in the data. It tells us that the observations y can be relatively far apart from the regression line (note in fact that the density of the parameter σ is concentrated on the interval $[2.8, 3.8]$. This is a large standard deviation compared to the value of `sales` (sales range from about 0 to 28).

To understand the latter sentence we can plot some regression lines. Each element `sample[i]` include a **joint** sample of α, β, σ from the posterior. For each sample, we can plot a regression line

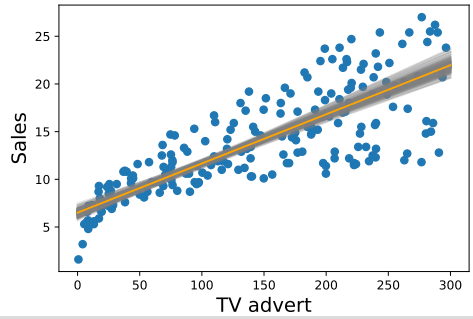
$$y_{real} = \beta[i]x + \alpha[i]$$

as follows:

[45] The interval $[6.1, 7.9]$ includes the 95% of the probability, that is $P(6.1 \leq \alpha \leq 7.9) = 0.95$. This is called 95% high probability density interval (HPD) and it is a good way to measure the uncertainty associated to a PDF. Comparing the width of the 95% HPD of α and β, we can immediately see that the uncertainty on α is higher.

Code

```
xx=np.linspace(0,300)
for i in range(200):#we consider only 200 samples
    plt.plot(xx,xx*samples['beta'][i]+samples['alpha'][i
                                ],c='gray',alpha=0.2)#
                                regression line
plt.scatter(x,y)
plt.xlabel("TV advert", fontsize=16)
plt.ylabel("Sales", fontsize=16)
```



Each grey line corresponds to a different sample and shows how the uncertainty on α, β is transferred to the prediction of the regression model.[46] The orange line is the mean of all grey lines and coincides with the least squares regression line.[47]

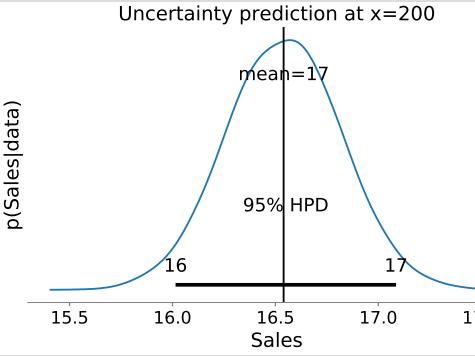
We can plot the uncertainty of the prediction at $x = 200$ as follows:

[46] The regression line at the value x is indeed the prediction of the linear regression.

[47] This is true in this case but not in general.

Code

```
import arviz as az
az.plot_dist(200*samples['beta']+samples['alpha'])
plt.axvline(reg.predict(np.array([[200]])),color='black'
                                    )
```



The vertical line is the prediction of the least-squares method ("Sklearn"). We have now a measure of uncertainty.

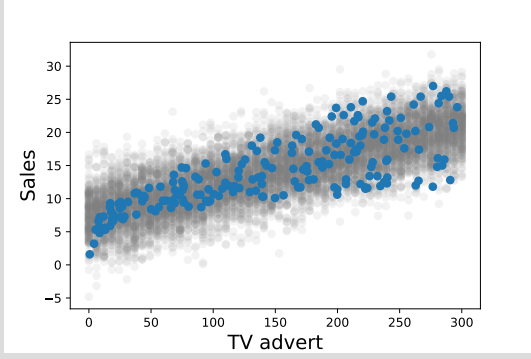
What is the role of σ? Contrarily to "Sklearn", we introduced an additional parameter, σ, that is the standard deviation of the observation noise. This tells us the noise in the data ($y_{observed} \neq y_{real}$), that is

$$N(y_{observed}; \beta[i]x + \alpha[i], \sigma[i]),$$

We can visualise the uncertainty on $y_{observed}$ as follows:

Code

```
xx=np.linspace(0,300)
for i in range(200):
    plt.scatter(xx,pm.Normal.dist(xx*samples['beta'][i]+
                                  samples['alpha'][i],
                                  samples['sigma'][i]).
                                  random(1),c='gray',alpha
                                  =0.1)#regression line
plt.scatter(x,y,Zorder=1000)
```



The grey points scatter plots represent datasets simulated according to the probabilistic model we have learned. It tells us that there is a lot of uncertainty. For instance, the scatter plot of $y_{observed}$ at $x = 200$

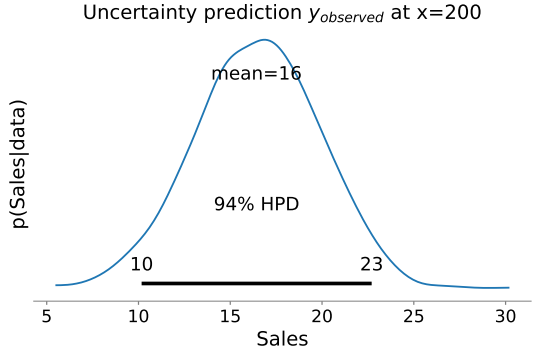


Figure 4.4: Density plot prediction of $y_{observed}$ at $x = 200$

Although the average value of the sales at $x = 200$ is 16.5 (hundreds), this plot shows that there is a huge uncertainty. Therefore, it tells us that if we invest 200 in TV advertisements, the probability that the number of sales is less than 12 is about 0.1 (10%).

There is a relatively high risk that, even if we decide to invest 200, the number of sales will be equal to the ones we would get by investing 105 (for instance).

Code

```
acc=[]
for i in range(2000):
    acc.append(pm.Normal.dist(200.0*samples['beta'][i]+
                              samples['alpha'][i],
                              samples['sigma'][i]).
                              random(1))
acc=np.array(acc)
print(np.sum((acc<12))/len(acc))
az.plot_posterior(np.array(acc))
```

```
0.101
```

This information is not provided by standard linear regression ("Sklearn").

By simulating the data from the posterior,

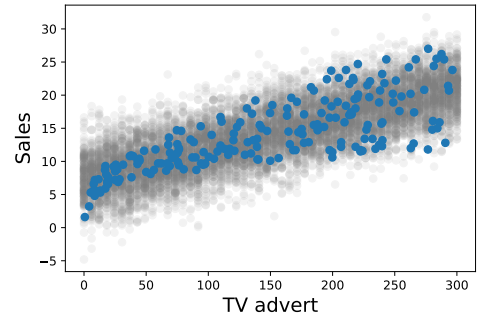


Figure 4.5: Simulation of $y_{observed}$ from the posterior

we also gain an important insight. The simulation can be used to analyze the degree to which data generated from the model deviate from the true data. Note for instance that, the variability of the true data in the interval Tv advert. $\in [0, 50]$ is much lower than the variability of the $y_{observed}$ generated from the posterior, while for instance in the interval Tv advert. $\in [150, 200]$ they are more in agreement. This tells us that the probabilistic model we defined cannot accurately reproduce the patterns observed in the real data. In other words, the linear regression model we defined is not appropriated for this dataset, see Exercise 3 in the next section for a better model.

4.5 Exercises with solutions

Exercise 1 Consider the linear regression model

Code

```
x=data['TV'].values
y=data['Sales'].values
regmodel=pm.Model()
with regmodel:
    alpha = pm.Normal('alpha',0.0,30)
    beta  = pm.Normal('beta',0.0,30)
    sigma = pm.Uniform('sigma',0.0001,30)
    mu = beta*x+alpha
    pm.Normal('Like',mu,sigma,observed=y)
```

1. Sample 200 prior regression lines from the above model, that the regression lines before seeing the data and plot them.
2. How do these sampled lines change if you change the priors to `pm.Normal('alpha',0.0,10), pm.Normal('beta',0.0,10)` or to `pm.Normal('alpha',0.0,50), pm.Normal('beta',0.0,50)`.
3. Sample 200 posterior regression lines after seeing only the first data point in the dataset $x[0], y[0]$ and plot them.
4. Sample and plot 200 posterior regression lines after seeing the first and second data point in the dataset.
5. Sample and plot 200 posterior regression lines after seeing the first, second and third data point in the dataset.
6. Sample $y_{observed}$ for the scenario discussed in the above question.
7. Multiply the y (sales) in the dataset by 100 and run the following model:

```
regmodel=pm.Model()
with regmodel:
    alpha = pm.Normal('alpha',0.0,30)
    beta  = pm.Normal('beta',0.0,30)
    sigma = pm.Uniform('sigma',0.0001,20)
    mu = pm.Deterministic('mu',beta*x+alpha)
    yo=pm.Normal('Like',mu,sigma,observed=y*100)
with regmodel:
    samples=pm.sample(5000,tune=1000, chains=1)
for i in range(100):
    plt.plot(x,samples['alpha'][i]+samples['beta'][
                                i]*x,color='gray',
                                alpha=0.25)
plt.scatter(x,y*100,zorder=1000)
```

what do you notice?

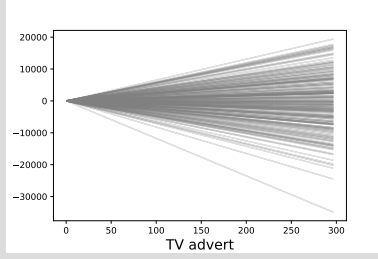
4.5.1 Solutions

Exercise 1

1. We sample regression lines from the priors as follows

Code

```
alpha = pm.Normal.dist(0.0,30).random(size=200)
beta = pm.Normal.dist(0.0,30).random(size=200)
for i in range(200):
    plt.plot(x,alpha[i]+beta[i]*x,color='gray',
                                  alpha=0.25)
```



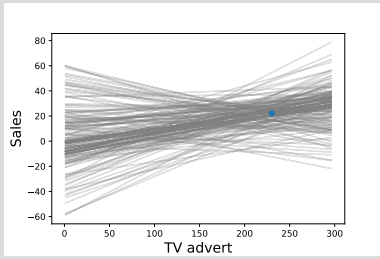
This plot shows our prior hypotheses (our prior beliefs) on the regression line (only 200 samples). Note that, in PyMC3, when the distributions are outside the `with` statement, they are defined as `pm.Normal.dist(0.0,30)` instead of `pm.Normal('alpha',0.0,30)`. In this case, to sample from the distribution, we must run the method `random`.

2. We can sample regression lines from the posterior after seeing one observation as follows:

Code

```
regmodel=pm.Model()
with regmodel:
    alpha = pm.Normal('alpha',0.0,30)
    beta  = pm.Normal('beta',0.0,30)
    sigma = pm.Uniform('sigma',0.0001,20)
    mu = pm.Deterministic('mu',beta*x[0]+alpha)
    yo=pm.Normal('Like',mu,sigma,observed=y[0])
with regmodel:
    samples=pm.sample(5000)
for i in range(200):
    plt.plot(x,samples['alpha'][i]+samples['beta'][
                                  i]*x,color='gray',
                                  alpha=0.25)
```

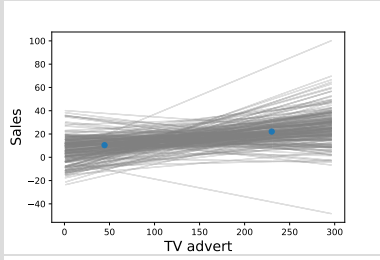
```
plt.scatter(x[0],y[0],zorder=1000)
```

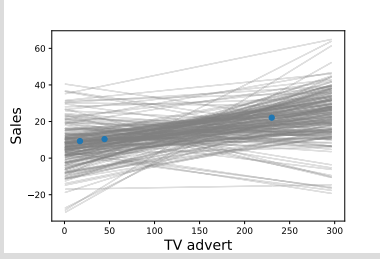


By comparing this plot with the previous one, we can see how one observation changes our prior beliefs of the regression model.

3. After tho and respectively, three observations we have

Code



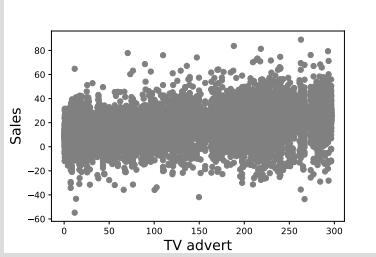


4. We cample $y_{observed}$ as follows:

Code

```
regmodel=pm.Model()
with regmodel:
    alpha = pm.Normal('alpha',0.0,30)
    beta  = pm.Normal('beta',0.0,30)
    sigma = pm.Uniform('sigma',0.0001,20)
    mu = pm.Deterministic('mu',beta*x[0:3]+alpha)
    yo=pm.Normal('Like',mu,sigma,observed=y[0:3])
```

```
with regmodel:
    samples=pm.sample(5000)
for i in range(100):
    plt.scatter(x,pm.Normal.dist(samples['alpha'][i
                                 ]+samples['beta'][i]*
                                 x,samples['sigma'][i]
                                 ).random(size=1))
```



5. We can notice that PyMC3 needs more time to run `pm.sample`. This is due to the prior misspecification, that is a-priori we have assumed that the intercept is $N(\alpha, 0, 30)$ (the PDF is practically zero outside the interval $[-100, 100]$), but `y[np.argmin(x)]*200=320`. Therefore, a-priori we believe that values around 320 are very very low probable and instead the datasets includes values in that range. We can solve this problem in two ways, either by changing the priors (for instance by increasing the standard deviation in $N(\alpha, 0, 30)$ to $N(\alpha, 0, 250)$ or by choosing a mean that is compatible with the scale of the y) or by scaling the y (using for instance the `Standardscaler` in "Sklearn").

4.6 Exercises without solutions

Exercise 2 The dataset "Advertising.csv" includes three inputs TV, Radio and Newspaper advertisements and the output sales. Your task is to repeat the regression analysis we performed for TV-sales in this chapter, using least-squares (LS) (using "Sklearn") and Bayesian probabilistic linear regression (BLR) (using "PyMC3"), separately for Radio-Sales and Newspaper-Sales.

1. Plot the regression line for LS and the regression lines for 150 samples from the posterior of BLR.

2. Plot the density plot of the prediction of y_{real} at x=250.

3. Plot the scatter plot for $y_{observed}$.

Exercise 3 Consider again TV-sales and evaluate the output of the following probabilistic model:

Code

```
regmodel=pm.Model()
with regmodel:
    alpha = pm.Normal('alpha',0.0,30)
    beta  = pm.Normal('beta',0.0,30)
    sigma1 = pm.Uniform('sigma1',0.0001,20)
    sigma2 = pm.Uniform('sigma2',0.0001,20)
    z = beta*x+alpha
    sigma=sigma1+x*sigma2#)
    pm.Normal('Like',z,sigma,observed=y)

with regmodel:
    samples=pm.sample(10000,tune=3000,chains=1)
```

1. What is the difference between this model and the previous one?
2. Is this a better model yes or no? Why?