Model-Based Statistical Learning, lecture 2

Linear models for classification: generative approaches

Pierre-Alexandre Mattei

Inria

This lecture is mostly based on Section 4.2 of Bishop's (2006) book

Outline of lecture

What is classification about? A concrete example

Probabilistic models for classification: the two schools

Probabilistic generative models

Modelling and estimating the class proportions

Multivariate Gaussian models for continuous data

What is classification about?

Let's start with a **concrete example** before digging into the equations...

Police uncover Italian wine fraud

Maggie Rosen
August 23, 2007

3 shares

Police have broken up an international counterfeit wine racket involving top Italian wines.

German and Italian police forces have uncovered a cross-border scam involving table wine from Puglia and Piedmont sold as Barolo, Brunello di Montalcino, Amarone and Chianti.

Unlabelled wine was brought into Germany, where it was given fake DOC and DOCG seals and phoney labels from well-known producers as well as non-existent wineries.

Ten people have been charged. It is thought the con could be worth €750,000.

Article from https://www.decanter.com/wine-news/police-uncover-italian-wine-fraud-88060/

What is classification about? A concrete example

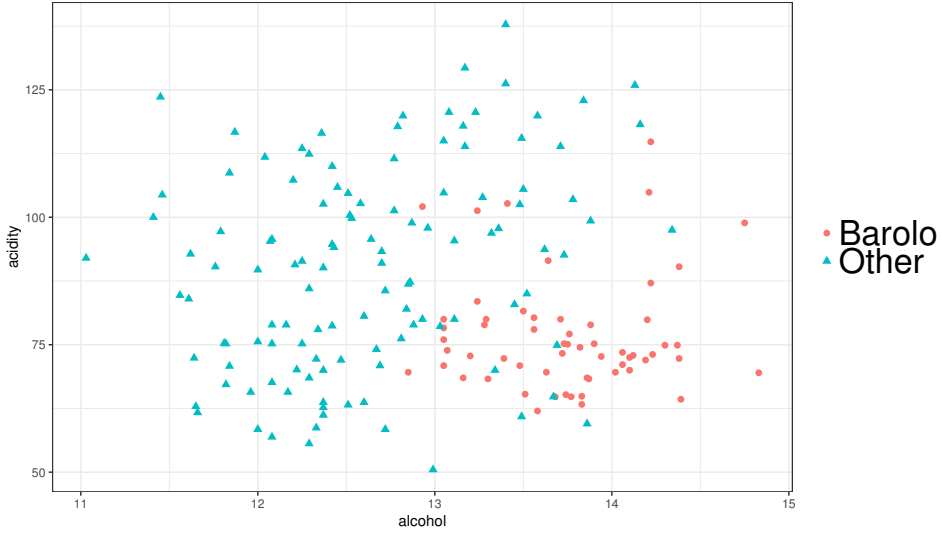
Would it be possible to **create an AI device that learns to estimates the probability that a bottle of wine is fake or not?** The device would need some information about the wine, for example some chemical properties: alcohol content and acidity.

One of the wines the bad guys counterfeited was from the Barolo region. According to Wikipedia, those wines have "pronounced tannins and acidity", and "moderate to high alcohol levels (Minimum 13%)". This would help a trained human recognise them, but **could we train an algorithm to learn those characteristics?**

Picture from Wikipedia

What is classification about? A concrete example

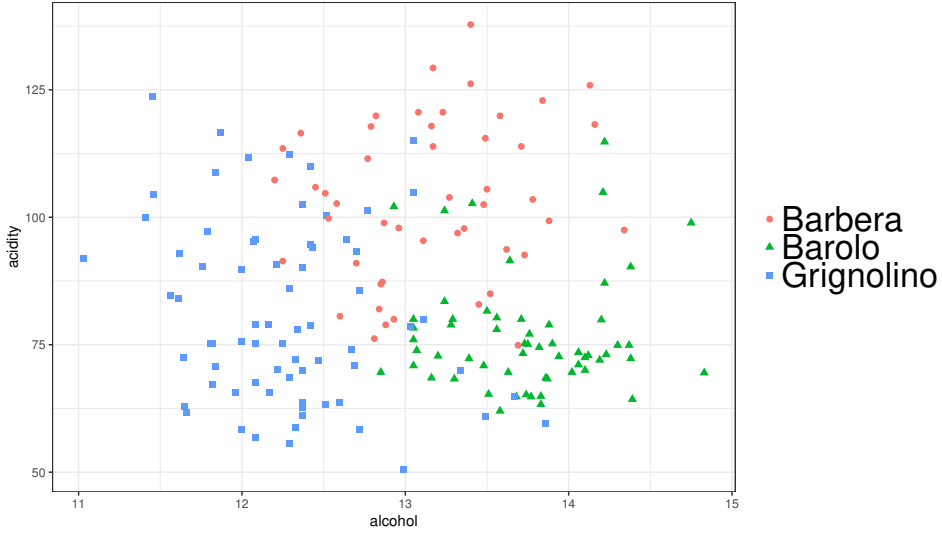
To train our algorithm, we need to feed him some data. Let's look at a few different bottles. We focus on two simple chemical properties that are easy to measure.



Forina, M., Armanino, C., Castino, M. and Ubigli, M. (1986). Multivariate data analysis as a discriminating method of the origin of wines. *Vitis* **25**, 189–201.

What is classification about? A concrete example

To train our algorithm, we need to feed him some data. Let's look at a few different bottles. We focus on two simple chemical properties that are easy to measure.



Forina, M., Armanino, C., Castino, M. and Ubigli, M. (1986). Multivariate data analysis as a discriminating method of the origin of wines. *Vitis* **25**, 189–201.

What is classification about? A general mathematical framework

Back to the maths. In classification the goal is to **build a decision rule that will allow us to assign any data point $\mathbf{x} \in \mathcal{X}$ to one of K discrete classes $\mathcal{C}_1, ..., \mathcal{C}_K$.**

- $\mathbf{x}$ is called an **input vector**, and contains the **features** of the data.
- $\mathcal{X}$ is called the **input space**, and can be continuous, discrete, or even both!

For more on 1-of-K coding, see p.180 of Bishop.

What is classification about? A general mathematical framework

Back to the maths. In classification the goal is to **build a decision rule that will allow us to assign any data point $\mathbf{x} \in \mathcal{X}$ to one of K discrete classes $\mathcal{C}_1, ..., \mathcal{C}_K$.**

- $\mathbf{x}$ is called an **input vector**, and contains the **features** of the data.
- $\mathcal{X}$ is called the **input space**, and can be continuous, discrete, or even both!

This decision rule will be learnt using some data $\{\mathbf{x}_n, \mathbf{t}_n\}_{n \leq N}$. For all $n \in \{1, ..., n\}$, t_n **represents the class of $\mathbf{x}_n$.** If there are only $K = 2$ classes, we call this problem **binary classification**, and we use a binary representation

$$t_n = \begin{cases} 1 & \text{if } \mathbf{x}_n \text{ is in class } \mathcal{C}_1 \\ 0 & \text{if } \mathbf{x}_n \text{ is in class } \mathcal{C}_2 \end{cases}$$

If $K > 2$, we use a **1-of-K coding** (also called **one-hot**) representation: $\mathbf{t}_n$ is a column vector with K coefficients that correspond to the possible classes. All coefficients of $\mathbf{t}_n$ are zero, except the one that corresponds to the class of $\mathbf{x}_n$, which is equal to one.

For more on 1-of-K coding, see p.180 of Bishop.

Back to our wine example: Barolo versus not Barolo

What does our data set look like mathematically?
First, let us assume that we just want to see if a wine is a true Barolo or not. It's a binary classification task. The first data points will look like this:

$$\mathbf{x}_1 = (14.23, 73.1), t_1 = 1$$
$$\mathbf{x}_2 = (14.13, 125.9), t_2 = 0$$
$$\mathbf{x}_3 = (13.20, 72.8), t_3 = 1$$
$$\dots$$

They follow the pattern

$$\mathbf{x}_n = (\text{alcohol}, \text{acidity}), t_n = \begin{cases} 1 & \text{if it's a } \textbf{Barolo} \text{ (class } \mathcal{C}_1) \\ 0 & \text{if it's } \textbf{not a Barolo} \text{ (class } \mathcal{C}_2) \end{cases}$$

We can store the whole data set $\{\mathbf{x}_n, \mathbf{t}_n\}_{n \leq N}$ inside one $n \times 2$ design matrix $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_N)^\intercal$, where each row correspond to one data point $\mathbf{x}_n$, and a binary vector $\mathbf{t} = (t_1, ..., t_N)^T$.

Back to our wine example: Barolo versus Grignolino versus Barbera

Now, we assume that we have $K = 3$ **classes: Barolo**, **Grignolino**, and **Barbera.** Since $K > 2$, we will use the **1-of-K vector representation** for the classes.

We use bold cases to denote vectors of size > 1, $\mathbf{t}_n$ will be therefore bold because we have more than two classes. Precisely:

$$\mathbf{t}_n = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \text{ if } \mathbf{x}_n \text{ is a } \textbf{Barolo} \text{ (class } \mathcal{C}_1)$$

$$\mathbf{t}_n = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \text{ if } \mathbf{x}_n \text{ is a } \textbf{Grignolino} \text{ (class } \mathcal{C}_2)$$

$$\mathbf{t}_n = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \text{ if } \mathbf{x}_n \text{ is a } \textbf{Barbera} \text{ (class } \mathcal{C}_3)$$

We can store the whole data set $\{\mathbf{x}_n, \mathbf{t}_n\}_{n \leq N}$ inside one $n \times 2$ design matrix $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_N)^\intercal$, where each row correspond to one data point $\mathbf{x}_n$, and one $N \times 3$ binary matrix $\mathbf{T} = (\mathbf{t}_1, ..., \mathbf{t}_N)^T$.

Let's look at the data in Python

For the programming exercise, we'll give 3 `.txt` files that contain the features, the binary classes (Barolo vs rest), and the 1-of-3 classes.
Let's first look at the features (aka design matrix):

```
import numpy as np
x = np.loadtxt("x.txt")
y_bin = np.loadtxt("y_bin.txt")
y_lofK = np.loadtxt("y_lofK.txt")
```

```
print(x)
```

```
[[ 13.83       63.299999]
 [ 12.84       82.      ]
 [ 13.86       59.5     ]
 [ 12.58      102.699997]
 [ 12.96       97.900002]
 [ 14.06       73.5     ]
 [ 12.08       78.900002]
 [ 13.11       95.400002]
 [ 12.29       86.      ]
```

Let's look at the data in Python

Now, let's look at the classes:

```
print(y_bin)
```

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

```
print(y_1ofK)
```

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

Now that we have a **goal**, and **some data**, what can we do to actually make predictions?

There are many approaches, and the most popular are shown in Bishop. This week, we'll focus on **probabilistic methods**, which aim at **training probabilisitic models of the data**, that can be used later on for **prediction.**

Outline of lecture

What is classification about? A concrete example

Probabilistic models for classification: the two schools

Probabilistic generative models

Modelling and estimating the class proportions

Multivariate Gaussian models for continuous data

Probabilistic models for classification: the two schools

There are **two schools** of modelling when it comes to building a probabilistic model of our data $\{\mathbf{x}_n, \mathbf{t}_n\}_{n \leq N}$. Both make the common and simple assumption that the data are made of **independent and identically distributed realisations of a random variable** $(\mathbf{x}, \mathbf{t})$, or equivalently $(\mathbf{x}, \mathcal{C}_k)$.

- The **generative school** builds models for the **joint distribution** $p(\mathbf{x}, \mathbf{t})$**.** This is quite hard, because this means we'll have in particular to learn the distribution of the data $p(\mathbf{x})$, which can be very complicated. **We'll study that today.**
- The **discriminative school** builds models for the **conditional distribution** $p(\mathbf{t}|\mathbf{x})$**.** This is much easier, because we don't care about $p(\mathbf{x})$, but this is also often less powerful (in particular, all the training set needs to be labelled). **We've studied that yesterday.**

You can also look at Tom Minka's short note on those differences: *Discriminative models, not discriminative training* [1].

[1] https://tminka.github.io/papers/minka-discriminative.pdf

Outline of lecture

What is classification about? A concrete example

Probabilistic models for classification: the two schools

Probabilistic generative models

Modelling and estimating the class proportions

Multivariate Gaussian models for continuous data

Probabilistic generative models

So, we want to define a distribution $p(\mathbf{x}, \mathcal{C}_k)$ Using the product rule, we can write

$$p(\mathbf{x}, \mathcal{C}_k) = p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k).$$

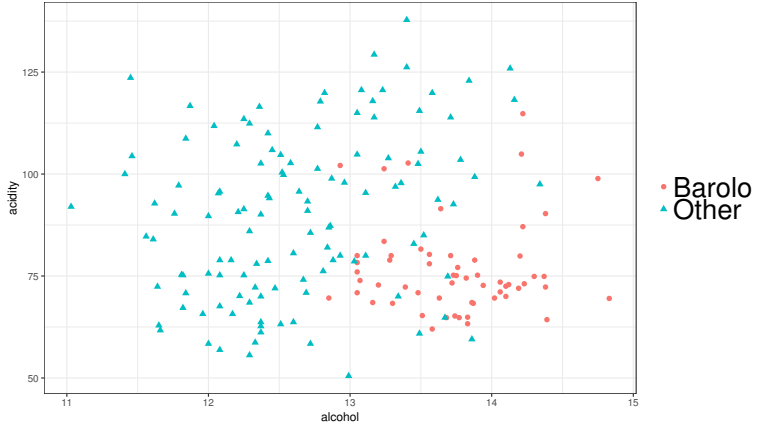
This involve two terms:

- The **class conditional densities** $p(\mathbf{x}|\mathcal{C}_k)$, which are the K different distributions $p(\mathbf{x}|\mathcal{C}_1), ..., (\mathbf{x}|\mathcal{C}_K)$ of the K classes.
- The **class prior probabilities** $p(\mathcal{C}_k)$, which are the proportions $p(\mathcal{C}_1), ..., (\mathcal{C}_K)$ of the classes in the population.

Of course, **both $p(\mathbf{x}|\mathcal{C}_k)$ and $p(\mathcal{C}_k)$ will need to be learned**, but in this section, we'll just assume that we know them. We'll show how to learn them in the next section (and in the exercises).

What can we do with such generative models?

To simplify notations, I'll assume from now on that **we only have $K = 2$ classes $\mathcal{C}_1$ and $\mathcal{C}_2$** (e.g. Barolo wine and not Barolo wine).
Let's assume we have learned $p(\mathbf{x}|\mathcal{C}_1)$ (density of the Barolos), $p(\mathbf{x}|\mathcal{C}_2)$ (density of the other wines), $p(\mathcal{C}_1)$ (proportion of Barolos), $p(\mathcal{C}_2)$ (proportion of the other wines).

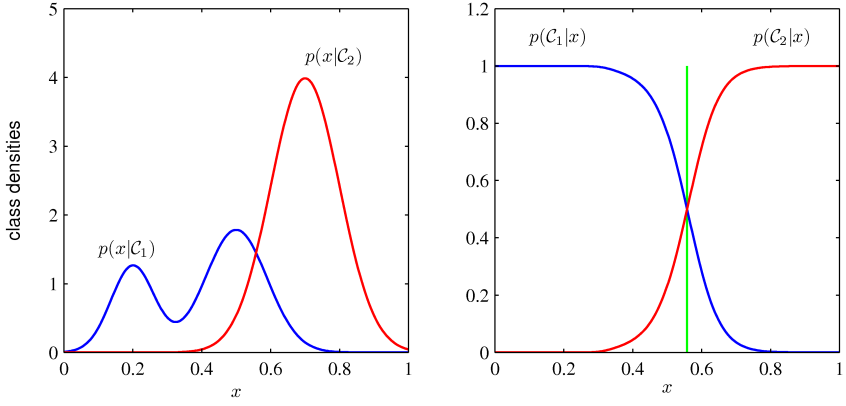


What can we do with such generative models?

Our main goal is to estimate the probability that the bottle is truly a Barolo. With this model, we can exactly compute this quantity using **Bayes's rule** and the law of total probability:

$$p(\mathcal{C}_1|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}.$$

A simplified and idealised way of visualising this is when there is only a single feature x (let's say acidity), as in Fig. 1.27 of Bishop:



Note that these curves do not correspond to the wine data.

What can we do with such generative models?

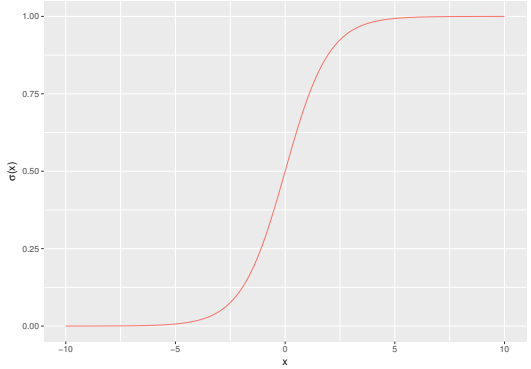
Let's look more closely at our formula:

$$p(\mathcal{C}_1|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}.$$

It only involves the class-conditional densities and the class proportions. It can also be rewritten:

$$p(\mathcal{C}_1|\mathbf{x}) = \sigma(a(\mathbf{x})), \text{where } a(\mathbf{x}) = \ln \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)} = \ln \frac{p(\mathbf{x}, \mathcal{C}_1)}{p(\mathbf{x}, \mathcal{C}_2)},$$

and σ is the **logistic sigmoid function** $\sigma(a) = \frac{1}{1+\exp(-a)}$:



What can we do with such generative models?

Let's look more closely at our formula:

$$p(\mathcal{C}_1|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}.$$

It only involves the class-conditional densities and the class proportions. It can also be rewritten:

$$p(\mathcal{C}_1|\mathbf{x}) = \sigma(a(\mathbf{x})), \text{where } a(\mathbf{x}) = \ln \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)} = \ln \frac{p(\mathbf{x}, \mathcal{C}_1)}{p(\mathbf{x}, \mathcal{C}_2)},$$

and σ is the **logistic sigmoid function** $\sigma(a) = \frac{1}{1+\exp(-a)}$.

Why on earth do we need this weird sigmoid function? There are several reasons, some will become clearer later in the course, but some others can be understood this week:

- **Interpretation:** we can interpret $a(\mathbf{x})$ as **the amount of evidence about $\mathbf{x}$ being a true Barolo**, because it's the logarithm of the ratio of probabilities of being true and being fake: using Bayes's rule, we can see that $a(\mathbf{x}) = \ln \frac{p(\mathcal{C}_1|\mathbf{x})}{p(\mathcal{C}_2|\mathbf{x})}$.
- **Unification of the generative and discriminative schools:** using the logistic function makes a bridge between the generative methods, and the discriminative ones. It also makes a bridge with **neural networks** (aka deep learning), that you'll in the deep learning course.

What can we do with such generative models?

So, we can compute the probability of being a true Barolo:

$$p(\mathcal{C}_1|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}.$$

But, rather that probabilities, we may be interested in **classifying the data set.** A simple decision rule is to assign wine $\mathbf{x}_n$ to its **most probable class:**

$$\hat{t}_n = \begin{cases} 1 & \text{if } p(\mathcal{C}_1|\mathbf{x}) \geq p(\mathcal{C}_2|\mathbf{x}) \\ 0 & \text{if } p(\mathcal{C}_2|\mathbf{x}) > p(\mathcal{C}_1|\mathbf{x}), \end{cases}$$

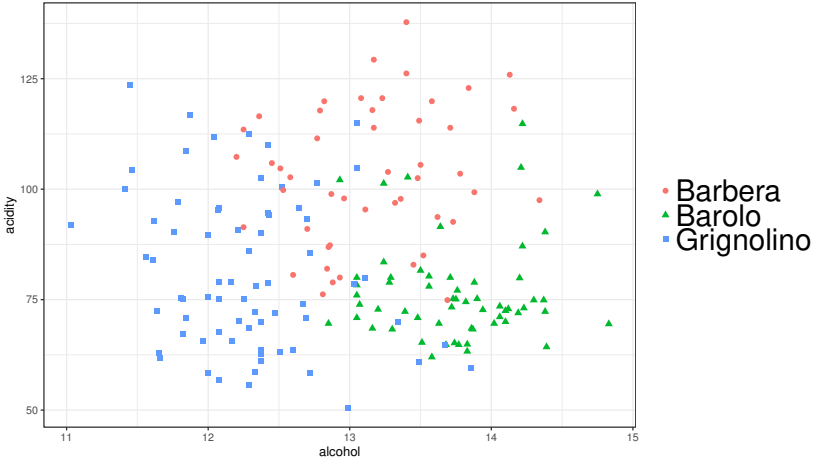
in the current case of binary classification, this reduces to

$$\hat{t}_n = \begin{cases} 1 & \text{if } p(\mathcal{C}_1|\mathbf{x}) \geq 0.5 \\ 0 & \text{if } p(\mathcal{C}_2|\mathbf{x}) > 0.5. \end{cases}$$

The case where $p(\mathcal{C}_1|\mathbf{x}) = p(\mathcal{C}_2|\mathbf{x}) = 0.5$ will never happen in practice, so we don't really worry about it...

What about when there are more than two classes?

Let's go back to the Barolo vs. Grignolino vs. Barbera problem... Can we find similar equations to classify the $K = 3$ wines? Again, we assume that we know $p(\mathbf{x}|\mathcal{C}_1)$ (density of the Barolos), $p(\mathbf{x}|\mathcal{C}_2)$ (density of the other Grignolinos), $p(\mathbf{x}|\mathcal{C}_3)$ (density of the other Barberas), $p(\mathcal{C}_1)$ (proportion of Barolos), $p(\mathcal{C}_2)$ (proportion of the Grignolinos), and $p(\mathcal{C}_3)$ (proportion of the Barberas).



What about when there are more than two classes?

Again, we can use Bayes's rule and the law of total probability: for each $k \in \{1, ..., K\}$, we have

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_{j=1}^{K} p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)}.$$

In this case we don't use the sigmoid function, but we can **compute all posterior probabilities** using the **softmax function** (aka normalised exponential), that's defined as:

$$\text{softmax}(a_1, ..., a_K) = \left(\frac{\exp a_1}{\sum_{j=1}^{K} \exp a_j}, \frac{\exp a_2}{\sum_{j=1}^{K} \exp a_j}, ..., \frac{\exp a_K}{\sum_{j=1}^{K} \exp a_j} \right)^\intercal.$$

Indeed, we can rewrite:

$$(p(\mathcal{C}_1|\mathbf{x}), p(\mathcal{C}_2|\mathbf{x}), ..., p(\mathcal{C}_K|\mathbf{x}))^\intercal = \text{softmax}(a_1(\mathbf{x}), ..., a_K(\mathbf{x})),$$

where

$$a_k(\mathbf{x}) = \ln p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k) = \ln p(\mathbf{x}, \mathcal{C}_k).$$

A few things about this softmax function

- For all possible $a_1, ..., a_K$, it's easy to check that the coefficients of softmax$(a_1, ..., a_K)$ are ≥ 0 and sum to one: in other words, **the softmax function always outputs probabililty distributions over the K classes.**

A few things about this softmax function

- For all possible $a_1, ..., a_K$, it's easy to check that the coefficients of softmax$(a_1, ..., a_K)$ are ≥ 0 and sum to one: in other words, **the softmax function always outputs probabililty distributions over the K classes.**
- For example, if, for a given wine sample $\mathbf{x}_0$, if we find that

$$\text{softmax}(a_1(\mathbf{x}_0), a_2(\mathbf{x}_0), a_3(\mathbf{x}_0)) = (0.52, 0.08, 0.4)^\intercal,$$

this means that **the probability of being a Barolo is $p(\mathcal{C}_1|\mathbf{x}_0) = 52\%$, the probability of being a Grignolino is $p(\mathcal{C}_2|\mathbf{x}_0) = 8\%$**, and **the probability of being a Barbera is $p(\mathcal{C}_3|\mathbf{x}_0) = 40\%$.**

A few things about this softmax function

- For all possible $a_1, ..., a_K$, it's easy to check that the coefficients of softmax$(a_1, ..., a_K)$ are ≥ 0 and sum to one: in other words, **the softmax function always outputs probabililty distributions over the K classes.**
- For example, if, for a given wine sample $\mathbf{x}_0$, if we find that

$$\text{softmax}(a_1(\mathbf{x}_0), a_2(\mathbf{x}_0), a_3(\mathbf{x}_0)) = (0.52, 0.08, 0.4)^\intercal,$$

  this means that **the probability of being a Barolo is $p(\mathcal{C}_1|\mathbf{x}_0) = 52\%$, the probability of being a Grignolino is $p(\mathcal{C}_2|\mathbf{x}_0) = 8\%$**, and **the probability of being a Barbera is $p(\mathcal{C}_3|\mathbf{x}_0) = 40\%$.**
- Its name comes from the fact that it can be seen a **smoothed version of the maximum function.** Why? Because, if we have some $a_1, ..., a_K$ and that one of them is much bigger than the others – say a_{j_0} – then, we'll have $[\text{softmax}(a_1, ..., a_K)]_{j_0} \approx 1$ and $[\text{softmax}(a_1, ..., a_K)]_j \approx 0$ for $j \neq j_0$.

A few things about this softmax function

- For all possible $a_1, ..., a_K$, it's easy to check that the coefficients of $\text{softmax}(a_1, ..., a_K)$ are ≥ 0 and sum to one: in other words, **the softmax function always outputs probabililty distributions over the K classes.**
- For example, if, for a given wine sample $\mathbf{x}_0$, if we find that

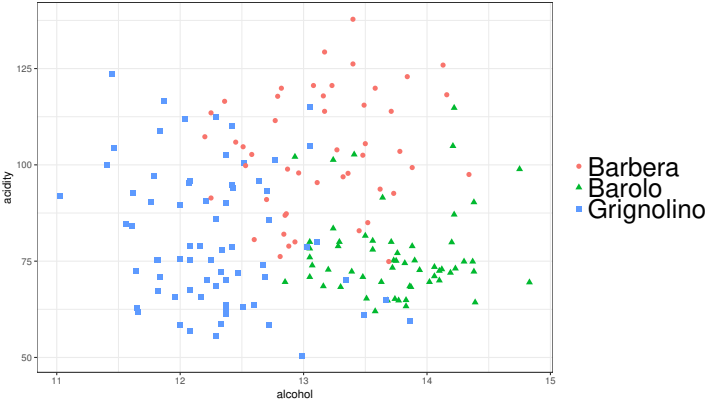
$$\text{softmax}(a_1(\mathbf{x}_0), a_2(\mathbf{x}_0), a_3(\mathbf{x}_0)) = (0.52, 0.08, 0.4)^\intercal,$$

  this means that **the probability of being a Barolo is $p(\mathcal{C}_1|\mathbf{x}_0) = 52\%$, the probability of being a Grignolino is $p(\mathcal{C}_2|\mathbf{x}_0) = 8\%$**, and **the probability of being a Barbera is $p(\mathcal{C}_3|\mathbf{x}_0) = 40\%$.**
- Its name comes from the fact that it can be seen a **smoothed version of the maximum function.** Why? Because, if we have some $a_1, ..., a_K$ and that one of them is much bigger than the others – say a_{j_0} – then, we'll have $[\text{softmax}(a_1, ..., a_K)]_{j_0} \approx 1$ and $[\text{softmax}(a_1, ..., a_K)]_j \approx 0$ for $j \neq j_0$.
- Like the sigmoid, the softmax function is also very important for **discriminative models and neural networks.**

What we have done, what we still have to do...

What we have done. If we know the class-conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ and the class proportions $p(\mathcal{C}_k)$, **we can compute the posterior probabilities** $p(\mathcal{C}_k|\mathbf{x})$, and use them to **classify data.**

What we still have to do. How can we find the class-conditional densities and the proportions?



Outline of lecture

What is classification about? A concrete example

Probabilistic models for classification: the two schools

Probabilistic generative models

Modelling and estimating the class proportions

Multivariate Gaussian models for continuous data

Modelling and estimating the class proportions

Let's go back to the simpler case where we only have **two classes** (e.g. Barolo vs not Barolo). In that case, the class $t \in \{0, 1\}$ is a **binary random variable**. We don't have any modelling choice there: the only way to model a binary variable is by using a **Bernoulli distribution**. We'll denote by $\pi \in [0, 1]$ its parameter[2]. This leads to

$$p(\mathcal{C}_1|\pi) = \pi, \text{ and } p(\mathcal{C}_2|\pi) = 1 - \pi.$$

Another way to write this is $p(t|\pi) = \pi^t(1 - \pi)^{1-t}$.

[2] Because π is the Greek letter for "P", the first letter of "Proportion".

Modelling and estimating the class proportions

Let's go back to the simpler case where we only have **two classes** (e.g. Barolo vs not Barolo). In that case, the class $t \in \{0, 1\}$ is a **binary random variable**. We don't have any modelling choice there: the only way to model a binary variable is by using a **Bernoulli distribution**. We'll denote by $\pi \in [0, 1]$ its parameter[2]. This leads to

$$p(\mathcal{C}_1|\pi) = \pi, \text{ and } p(\mathcal{C}_2|\pi) = 1 - \pi.$$

Another way to write this is $p(t|\pi) = \pi^t(1 - \pi)^{1-t}$.

We still need to estimate π...
For that purpose, we'll use the **maximum likelihood (ML)** estimation techinque.

[2]Because π is the Greek letter for "P", the first letter of "Proportion".

Modelling and estimating the class proportions

Let's write the log-likelihood function:[3]

$$\sum_{n=1}^{N} \ln p(\mathbf{x}_n, t_n|\pi) = \sum_{n=1}^{N} \ln p(\mathbf{x}_n|t_n)p(t_n|\pi) = \sum_{n=1}^{N} \ln p(\mathbf{x}_n|t_n) + \sum_{n=1}^{N} \ln p(t_n|\pi),$$

[3]This slide and the following one will be useful for this week's exercise 4.9...

Modelling and estimating the class proportions

Let's write the log-likelihood function:[3]

$$\sum_{n=1}^{N} \ln p(\mathbf{x}_n, t_n|\pi) = \sum_{n=1}^{N} \ln p(\mathbf{x}_n|t_n)p(t_n|\pi) = \sum_{n=1}^{N} \ln p(\mathbf{x}_n|t_n) + \sum_{n=1}^{N} \ln p(t_n|\pi),$$

only the second term depends on π, so we only need to maximise this one:

$$\sum_{n=1}^{N} \ln p(t_n|\pi) = \sum_{n=1}^{N} \ln \left(\pi^{t_n}(1-\pi)^{1-t_n}\right) = \sum_{n=1}^{N} t_n \ln \pi + \sum_{n=1}^{N} (1-t_n) \ln(1-\pi),$$

[3] This slide and the following one will be useful for this week's exercise 4.9...

Modelling and estimating the class proportions

Let's write the log-likelihood function:[3]

$$\sum_{n=1}^{N} \ln p(\mathbf{x}_n, t_n|\pi) = \sum_{n=1}^{N} \ln p(\mathbf{x}_n|t_n)p(t_n|\pi) = \sum_{n=1}^{N} \ln p(\mathbf{x}_n|t_n) + \sum_{n=1}^{N} \ln p(t_n|\pi),$$

only the second term depends on π, so we only need to maximise this one:

$$\sum_{n=1}^{N} \ln p(t_n|\pi) = \sum_{n=1}^{N} \ln \left(\pi^{t_n}(1-\pi)^{1-t_n}\right) = \sum_{n=1}^{N} t_n \ln \pi + \sum_{n=1}^{N} (1-t_n)\ln(1-\pi),$$

denoting by $N_1 = \sum_{n=1}^{N} t_n$, which is the number of samples from class 1 in the data set, we end up with

$$\sum_{n=1}^{N} \ln p(t_n|\pi) = N_1 \ln \pi + (N - N_1)\ln(1-\pi)$$

[3]This slide and the following one will be useful for this week's exercise 4.9...

Modelling and estimating the class proportions

The derivative of the likelihood with respect to π will therefore be

$$\frac{N_1}{\pi} - \frac{N - N_1}{1 - \pi}.$$

Modelling and estimating the class proportions

The derivative of the likelihood with respect to π will therefore be

$$\frac{N_1}{\pi} - \frac{N - N_1}{1 - \pi}.$$

Solving the equation

$$\frac{N_1}{\pi} = \frac{N - N_1}{1 - \pi},$$

leads to

$$\frac{1 - \pi}{\pi} = \frac{N - N_1}{N_1},$$

Modelling and estimating the class proportions

The derivative of the likelihood with respect to π will therefore be

$$\frac{N_1}{\pi} - \frac{N - N_1}{1 - \pi}.$$

Solving the equation

$$\frac{N_1}{\pi} = \frac{N - N_1}{1 - \pi},$$

leads to

$$\frac{1 - \pi}{\pi} = \frac{N - N_1}{N_1},$$

and to

$$\frac{1}{\pi} - 1 = \frac{N}{N_1} - 1 \implies \pi = \frac{N_1}{N}.$$

This means that **the maximum likelihood estimate (MLE) of π is exactly the fraction of samples observed in class 1.**

Modelling and estimating the class proportions

What happens when the number K of classes is bigger than 2? In that case, **the class is a discrete random variable with K outcomes.** Again, there is only one possible modelling choice for such a random variable: a **multinomial distribution.** The parameters $\pi_1, ..., \pi_K$ are stored in a K-dimensional vector $\boldsymbol{\pi}$. They are ≥ 0 and sum to one, and correspond to the proportions of the classes: $p(C_k|\boldsymbol{\pi}) = \pi_k$. We also have, for a 1-of-K representation $\mathbf{t} = (t_1, ..., t_K)^T$,

$$p(\mathbf{t}|\boldsymbol{\pi}) = \pi_1^{t_1} \times \pi_2^{t_2} \times ... \times \pi_K^{t_K}.$$

The MLE of $\boldsymbol{\pi}$ is given by

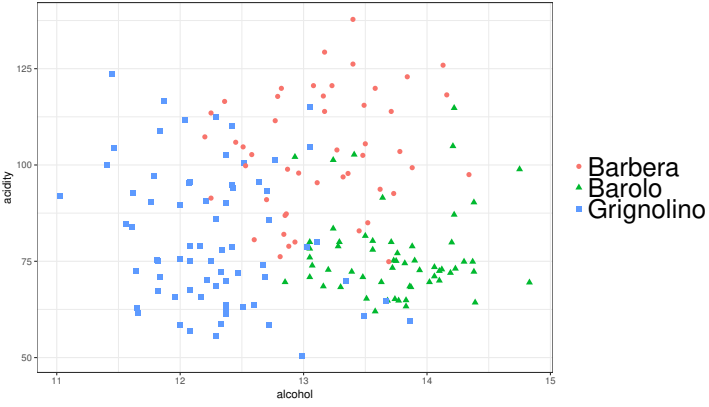
$$\boldsymbol{\pi} = \left(\frac{N_1}{N}, \frac{N_2}{N}, ..., \frac{N_K}{N} \right)^T,$$

which is exactly the vector of observed proportions for all the classes.

What we have done, what we still have to do...

What we have done. If we know the class-conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ and the class proportions $p(\mathcal{C}_k)$ **(we just learned how to compute those!)**, we can compute the posterior probabilities $p(\mathcal{C}_k|\mathbf{x})$, and use them to classify data.

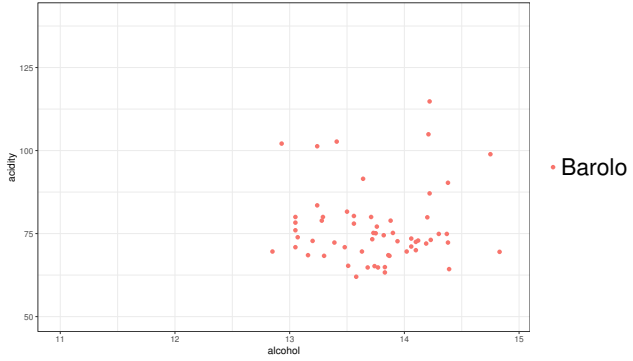
What we still have to do. The only things remaining are the class-conditional densities. The main approach is to **choose a simple (parametric) family of densities for** $p(\mathbf{x}|\mathcal{C}_k)$, and to find "good" parameters using **the maximum likelihood primciple.** For the wine data set, what kind of family would seem natural to use?



What we have done, what we still have to do...

What we have done. If we know the class-conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ and the class proportions $p(\mathcal{C}_k)$, we can compute the posterior probabilities $p(\mathcal{C}_k|\mathbf{x})$, and use them to classify data.

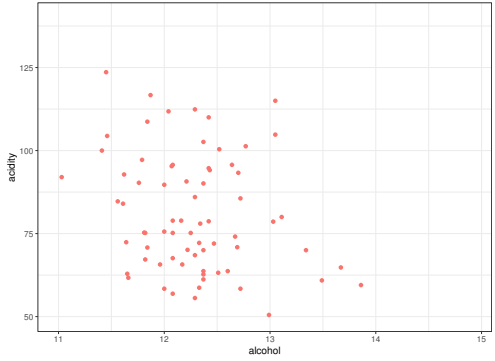
What we still have to do. How can we find the class-conditional densities and the proportions? The main approach is to **choose a simple (parametric) family of densities for** $p(\mathbf{x}|\mathcal{C}_k)$, and to find "good" parameters using **the maximum likelihood primciple.** For the wine data set, what kind of family would seem natural to use?
To have an idea, let's look at the classes independently: this one needs to be modelled using $p(\mathbf{x}|\mathcal{C}_1)$**.**



What we have done, what we still have to do...

What we have done. If we know the class-conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ and the class proportions $p(\mathcal{C}_k)$, we can compute the posterior probabilities $p(\mathcal{C}_k|\mathbf{x})$, and use them to classify data.

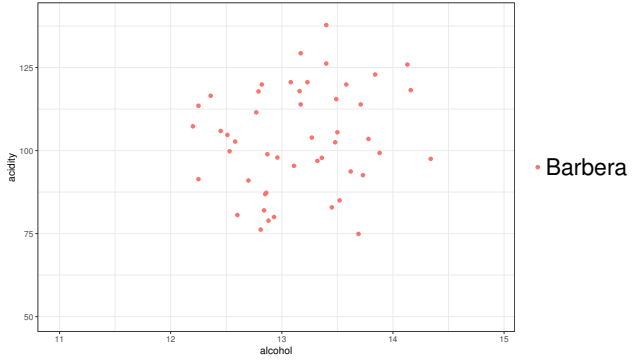
What we still have to do. How can we find the class-conditional densities and the proportions? The main approach is to **choose a simple (parametric) family of densities for** $p(\mathbf{x}|\mathcal{C}_k)$, and to find "good" parameters using **the maximum likelihood primciple.** For the wine data set, what kind of family would seem natural to use?
To have an idea, let's look at the classes independently: this one needs to be modelled using $p(\mathbf{x}|\mathcal{C}_2)$**.**



What we have done, what we still have to do...

What we have done. If we know the class-conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ and the class proportions $p(\mathcal{C}_k)$, we can compute the posterior probabilities $p(\mathcal{C}_k|\mathbf{x})$, and use them to classify data.

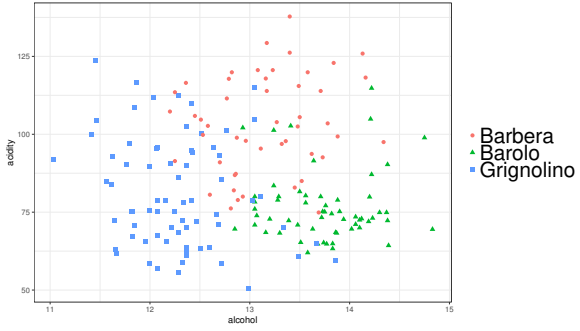
What we still have to do. How can we find the class-conditional densities and the proportions? The main approach is to **choose a simple (parametric) family of densities for** $p(\mathbf{x}|\mathcal{C}_k)$, and to find "good" parameters using **the maximum likelihood primciple.** For the wine data set, what kind of family would seem natural to use?
To have an idea, let's look at the classes independently: this one needs to be modelled using $p(\mathbf{x}|\mathcal{C}_2)$**.**



A few observations about the class-conditional densities of the wine data set

- It makes sense to want **smooth densities,**
- Given the class, the wines seem to be **quite localised around their class-conditional means,**
- Given the class, the wines seem to be **quite localised around their class-conditional means,**
- There is **a little bit of intra-class correlations** (especially for Grignolinos), but not a lot.

Do you know a bivariate family of distributions that would satisfy these needs?



Outline of lecture

What is classification about? A concrete example

Probabilistic models for classification: the two schools

Probabilistic generative models

Modelling and estimating the class proportions

Multivariate Gaussian models for continuous data

Recap - Probability densities

A **probability density** $p(x)$ over $x \in \mathbb{R}$ must satisfy

$$p(x) \geq 0 \quad \text{and} \quad \int_{-\infty}^{\infty} p(x)\,\mathrm{d}x = 1$$

The probability of x lying in the interval (a, b) is

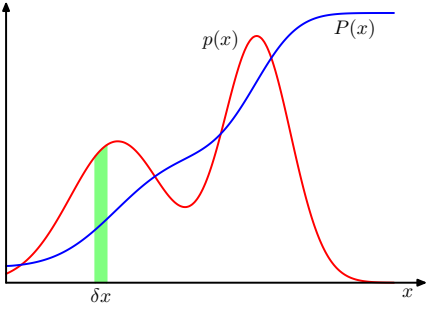
$$p(x \in (a, b)) = \int_{a}^{b} p(x)\,\mathrm{d}x$$

This is also valid in dimensions > 1, we are looking at **multivariate densities** $p(\mathbf{x})$ over $\mathbf{x} \in \mathbb{R}^D$, which satisfy

$$p(\mathbf{x}) \geq 0 \quad \text{and} \quad \int_{-\infty}^{\infty} p(\mathbf{x})\,\mathrm{d}\mathbf{x} = 1$$

$$p(\mathbf{x} \in A) = \int_{A} p(\mathbf{x})\,\mathrm{d}\mathbf{x},$$

for any subset $A \subset \mathbb{R}^D$.



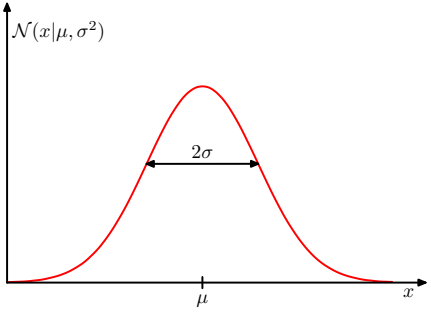
Figure(s) from **bishop2006**.

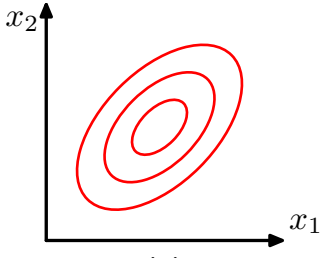
Recap - The Gaussian distribution

The univariate **Gaussian** distribution is defined by

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

where $\mu \in \mathbb{R}$ is the **mean** and $\sigma^2 \in \mathbb{R}$ is the **variance**.





The D-**dimensional Gaussian** is defined as, for each $\mathbf{x} \in \mathbb{R}^D$

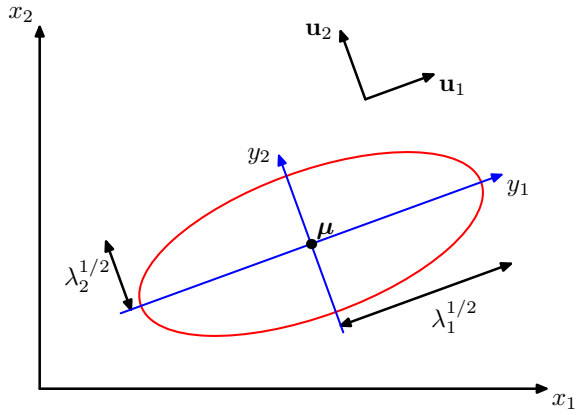
$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\intercal \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

where $\boldsymbol{\mu} \in \mathbb{R}^D$ **is the mean vector**, $\boldsymbol{\Sigma} \in \mathbb{R}^{D\times D}$ **is the covariance matrix** and $|\boldsymbol{\Sigma}|$ is the determinant of $\boldsymbol{\Sigma}$.

$\boldsymbol{\Sigma}$ must be **positive definite** (all eigenvalues are positive).

Figure(s) from **bishop2006**.

Recap - Geometry of the multivariate Gaussian



Red curve: elliptical surface of constant density of $\exp(-1/2)$.

Major axes are defined by the **eigenvectors** $\mathbf{u}_i$ of $\boldsymbol{\Sigma}$ and the corresponding **eigenvalues** λ_i.

Figure(s) from **bishop2006**.

Recap - Partitioned Gaussian distributions

Partitioned Gaussians

Given a joint Gaussian distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with $\boldsymbol{\Lambda} \equiv \boldsymbol{\Sigma}^{-1}$ and

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{pmatrix}, \qquad \boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{pmatrix}$$

$$\boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{ab} \\ \boldsymbol{\Sigma}_{ba} & \boldsymbol{\Sigma}_{bb} \end{pmatrix}, \qquad \boldsymbol{\Lambda} = \begin{pmatrix} \boldsymbol{\Lambda}_{aa} & \boldsymbol{\Lambda}_{ab} \\ \boldsymbol{\Lambda}_{ba} & \boldsymbol{\Lambda}_{bb} \end{pmatrix}.$$

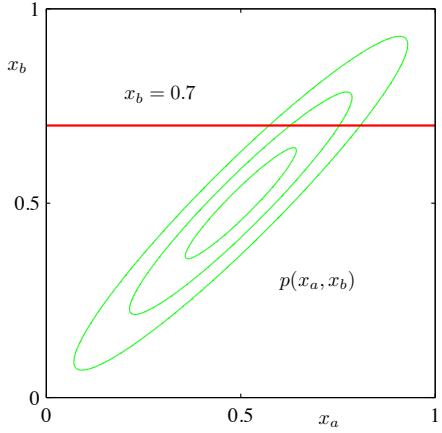
Conditional distribution:

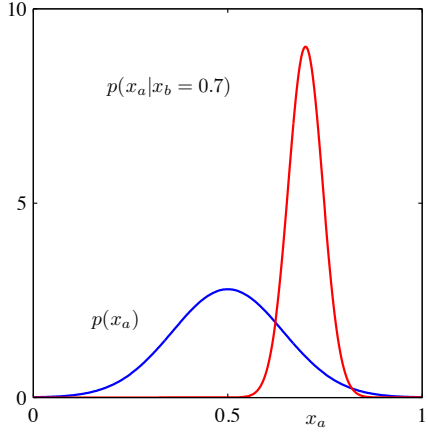
$$\begin{aligned} p(\mathbf{x}_a|\mathbf{x}_b) &= \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{a|b}, \boldsymbol{\Lambda}_{aa}^{-1}) \\ \boldsymbol{\mu}_{a|b} &= \boldsymbol{\mu}_a - \boldsymbol{\Lambda}_{aa}^{-1}\boldsymbol{\Lambda}_{ab}(\mathbf{x}_b - \boldsymbol{\mu}_b). \end{aligned}$$

Marginal distribution:

$$p(\mathbf{x}_a) = \mathcal{N}(\mathbf{x}_a|\boldsymbol{\mu}_a, \boldsymbol{\Sigma}_{aa}).$$

Recap - Partitioned Gaussian distributions

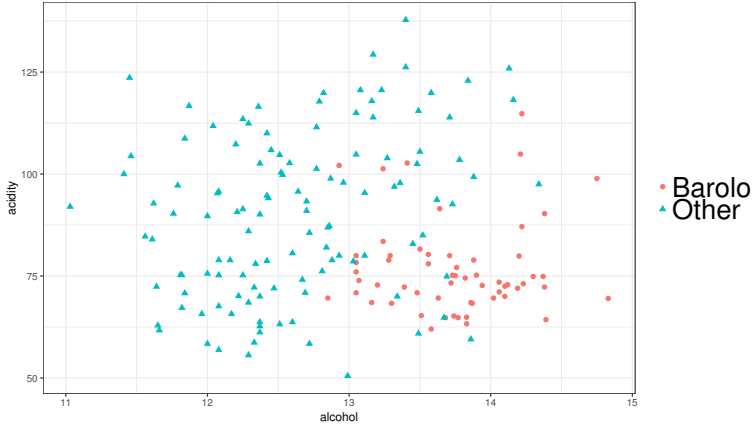




Figure(s) from **bishop2006**.

Using the multivariate Gaussian as class-conditional densitities

Let's go back to our wine data in the binary case, since the classes "kind of look like ellipses", **it would make sense to model them using multivariate Gaussians.** Namely, we will make the (strong but justified in the case of the wine data) **assumption that $p(\mathbf{x}|\mathcal{C}_1)$ and $p(\mathbf{x}|\mathcal{C}_2)$ are 2 bidimensional Gaussian distributions.**



Using the multivariate Gaussian as class-conditional densitities

More specifically, we assume that

$$p(\mathbf{x}|\mathcal{C}_1) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}), \text{ and } p(\mathbf{x}|\mathcal{C}_2) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}).$$

Note that the distributions have **different means but the same covariance matrix Σ**. It is possible to use two different covariances $\boldsymbol{\Sigma}_1$ and $\boldsymbol{\Sigma}_2$, but this would be (slightly) more complex. Another way to write it:

$$p(\mathbf{x}|t) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \boldsymbol{\Sigma})^t \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})^{1-t}.$$

Let's assume for a minute that we actually know all the parameter. Then, our classification rule will be:

$$p(\mathcal{C}_1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + w_0)$$

where

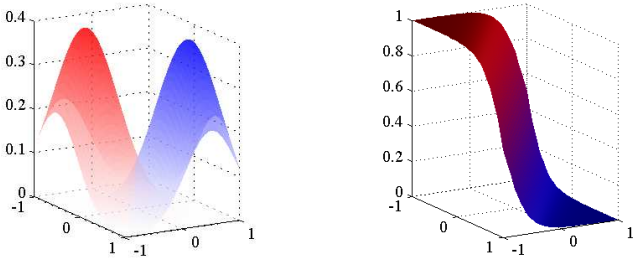
$$\mathbf{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2), \text{ and } w_0 = -\frac{1}{2}\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2 + \ln\frac{\pi}{1-\pi}.$$

Using the multivariate Gaussian as class-conditional densitities

If we look at the classification rule, what's inside the sigmoid is **a linear function:**

$$p(\mathcal{C}_1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + w_0).$$

This means that the decision boundary will be linear (a line or a plane). The main underlying reason is that the two class-conditional densities share the same covariance matrix.



Like for π before, we will find $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$, and $\boldsymbol{\Sigma}$ using **maximum likelihood.**

Fig. 4.10 from Bishop.

Maximum likelihood for the means

Let's write the likelihood: as before, we have

$$\sum_{n=1}^{N} \ln p(\mathbf{x}_n, t_n|\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}, \pi) = \sum_{n=1}^{N} \ln p(\mathbf{x}_n|t_n, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) + \sum_{n=1}^{N} \ln p(t_n|\pi),$$

this time, its the **first term** that depends on the parameters of interest (the means). So we want to maximise

$$\begin{aligned}\sum_{n=1}^{N} \ln p(\mathbf{x}_n|t_n, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) &= \sum_{n=1}^{N} \ln \left(\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma})^{t_n} \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})^{1-t_n}\right) \\ &= \sum_{n=1}^{N} t_n \ln \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}) + \sum_{n=1}^{N} (1-t_n) \ln \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma}).\end{aligned}$$

A little but very useful thing about the Gaussian distribution

The log of the density of a D-dimensional Gaussian density is rather simple:

$$\ln \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{D}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\intercal \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}).$$

An important point is that **the first part of the expression** does not depend on the mean! Using that in our likelihood formula, if we focus on $\boldsymbol{\mu}_1$, we see that we want to maximise the quantity

$$-\frac{1}{2}\sum_{n=1}^{N} t_n(\mathbf{x}_n - \boldsymbol{\mu}_1)^\intercal \boldsymbol{\Sigma}^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_1).$$

To differentiate this with respect to the means, we'll use Formula 86 from *The Matrix Cookbook*[4]:

$$\frac{\partial}{\partial \mathbf{s}}(\mathbf{x}-\mathbf{s})^\intercal \mathbf{W}(\mathbf{x}-\mathbf{s}) = -2\mathbf{W}(\mathbf{x}-\mathbf{s}),$$

as long as $\mathbf{W}$ is symmetric. So setting to zero the derivative with respect to $\boldsymbol{\mu}_1$ leads to

$$\sum_{n=1}^{N} t_n \boldsymbol{\Sigma}^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_1) = 0 \implies \sum_{n=1}^{N} t_n(\mathbf{x}_n - \boldsymbol{\mu}_1) = 0 \implies \boldsymbol{\mu}_1 = \frac{1}{N_1}\sum_{n=1}^{N} t_n \mathbf{x}_n$$

[4] https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf

Maximum likelihood for the means

A similar computation leads to the following MLE for $\boldsymbol{\mu}_2$:

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

Basically, the estimates of the class-conditional means are just the sample means of the two classes!
Regarding the MLE of $\boldsymbol{\Sigma}$, we won't do the proof[5], so we admit that the estimate is

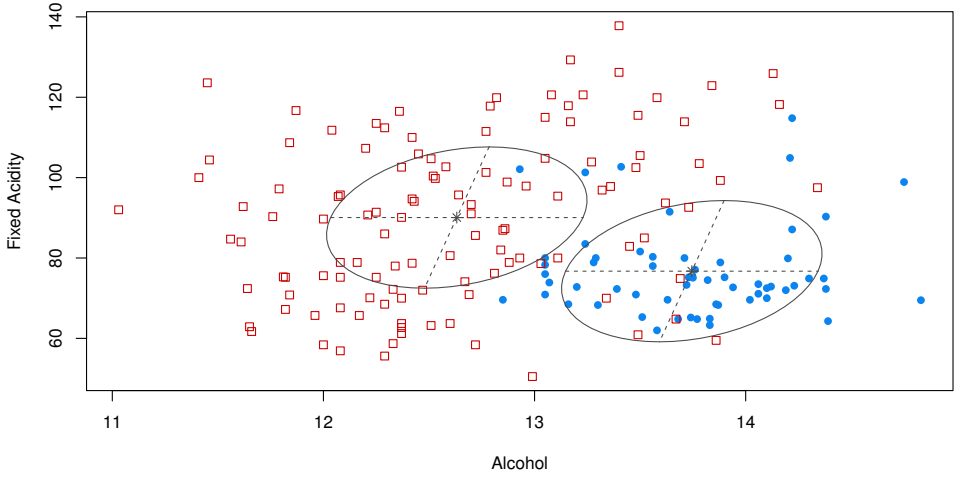
$$\boldsymbol{\Sigma} = \frac{N_1}{N} \sum_{n=1}^{N} t_n (\mathbf{x}_n - \boldsymbol{\mu}_1)(\mathbf{x}_n - \boldsymbol{\mu}_1)^\intercal + \frac{1 - N_1}{N} \sum_{n=1}^{N} (1 - t_n)(\mathbf{x}_n - \boldsymbol{\mu}_2)(\mathbf{x}_n - \boldsymbol{\mu}_2)^\intercal,$$

which is just a weighted average of the two covariance matrices of the classes.

[5] If you want to try to write a proof yourself, you can take a inspirational look at `https://en.wikipedia.org/wiki/Estimation_of_covariance_matrices`

Let’s go back to our wine AI thing

We have now everything we need to distinguish true Barolos from fake ones...



Let’s go back to our wine AI thing

But, as we can see, we’re still making quite a lot of errors (in black):