Machine Learning

Mohamad Ghassany

2017-01-25

Contents

Welcome

Welcome to this course. It is only a little introduction to Machine Learning.

The aim of Machine Learning is to build computer systems that can adapt to their environments and learn form experience. Learning techniques and methods from this field are successfully applied to a variety of learning tasks in a broad range of areas, including, for example, spam recognition, text classification, gene discovery, financial forecasting. The course will give an overview of many concepts, techniques, and algorithms in machine learning, beginning with topics such as linear regression and classification and ending up with topics such as kmeans and Expectation Maximization. The course will give the student the basic ideas and intuition behind these methods, as well as a more formal statistical and computational understanding. Students will have an opportunity to experiment with machine learning techniques in R and apply them to a selected problem.

Introduction

What is Machine Learning ?

What is Machine Learning?

Two definitions of Machine Learning are offered. Arthur Samuel described it as: “the field of study that gives computers the ability to learn without being explicitly programmed.” This is an older, informal definition.

Tom Mitchell provides a more modern definition: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”

Machine Learning is also called Statistical Learning.

Example: playing checkers.

E = the experience of playing many games of checkers

T = the task of playing checkers.

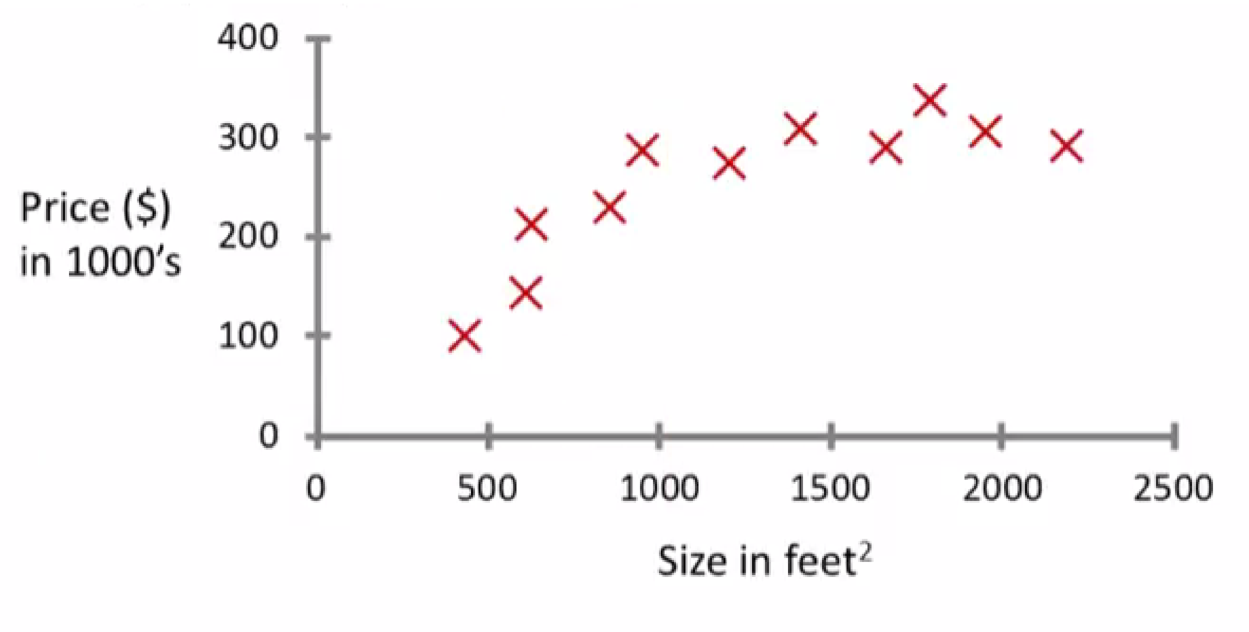
P = the probability that the program will win the next game.

In general, any machine learning problem can be assigned to one of two broad classifications:

Supervised learning and Unsupervised learning.

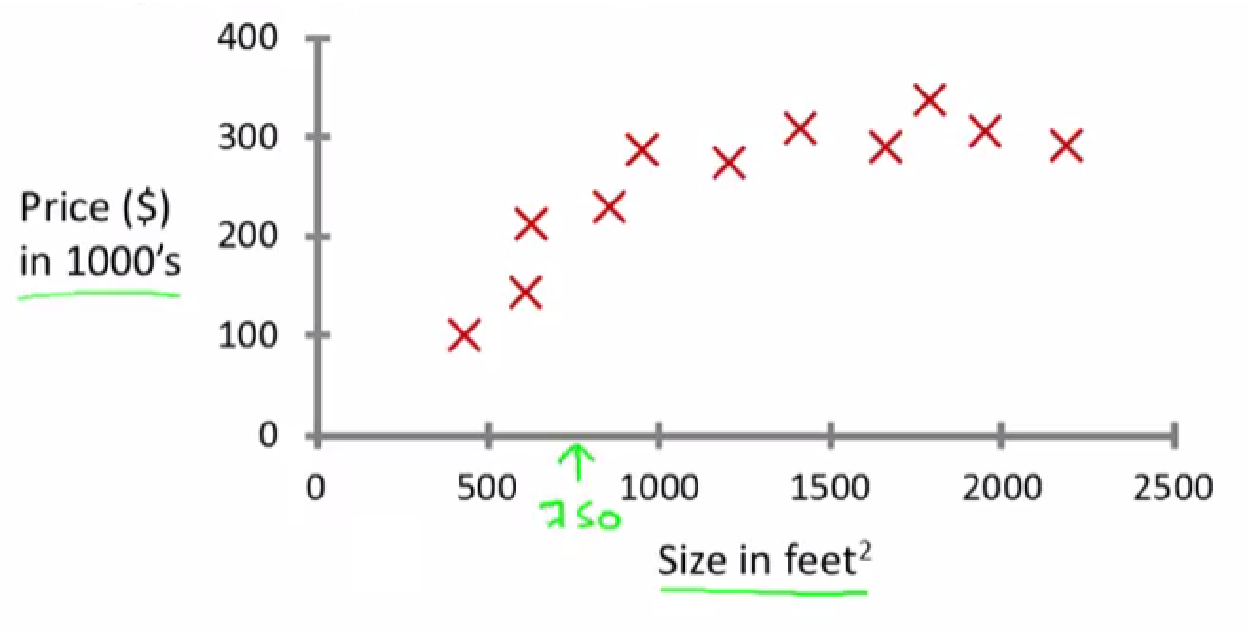
Supervised Learning

Supervised Learning is probably the most common type of machine learning problem. Let’s start with an example of what is it. Let’s say we want to predict housing prices. We plot a data set and it looks like this.



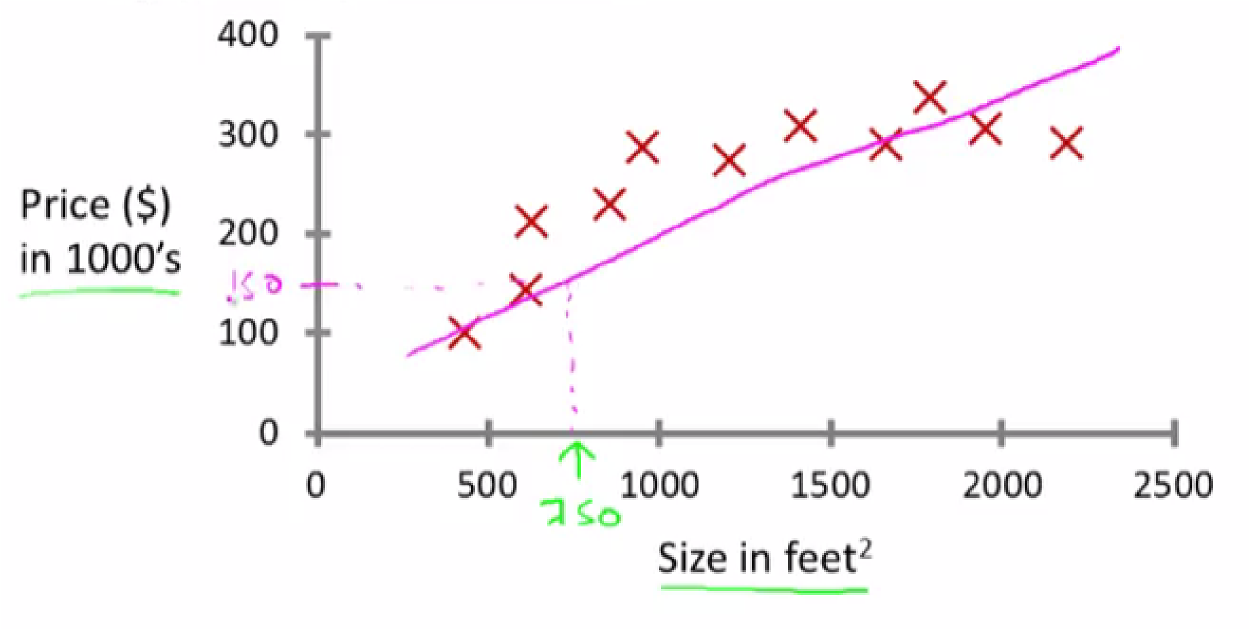
Here on the horizontal axis, the size of different houses in square feet, and on the vertical axis, the price of different houses in thousands of dollars.

So. Given this data, let's say we own a house that is, say 750 square feet and hoping to sell the house and we want to know how much we can get for the house.

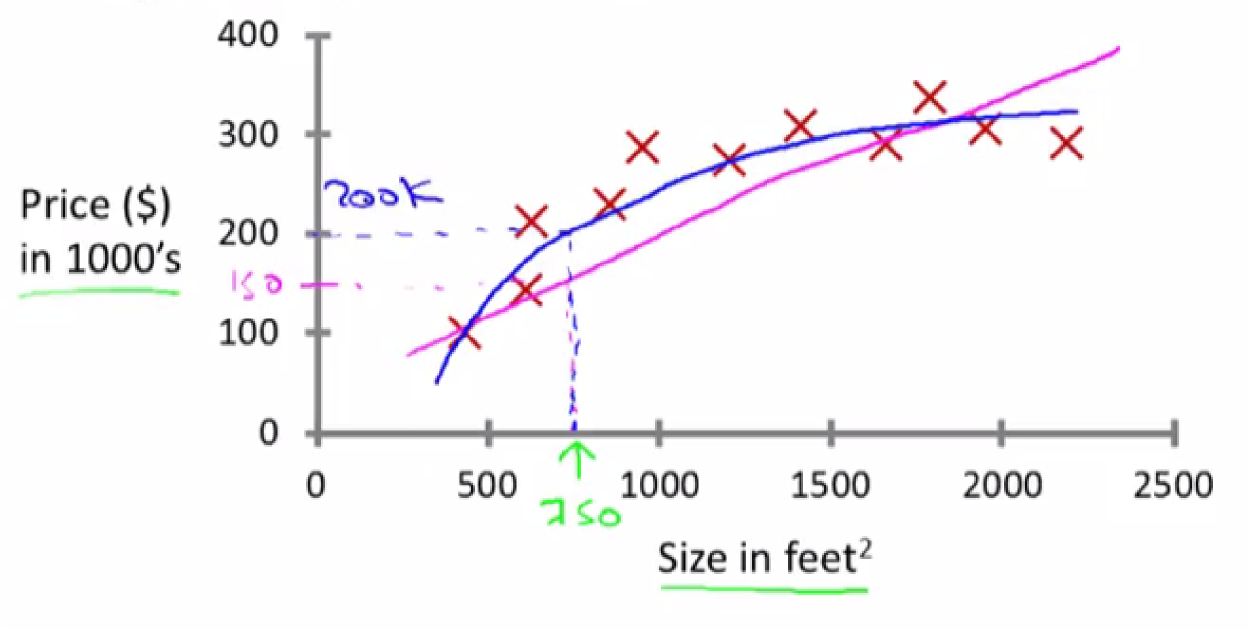


So how can the learning algorithm help?

One thing a learning algorithm might be able to do is put a straight line through the data or to **"fit"** a straight line to the data and, based on that, it looks like maybe the house can be sold for maybe about $150,000.



But maybe this isn't the only learning algorithm we can use. There might be a better one. For example, instead of sending a straight line to the data, we might decide that it's better to fit a *quadratic function* or a *second-order polynomial* to this data.



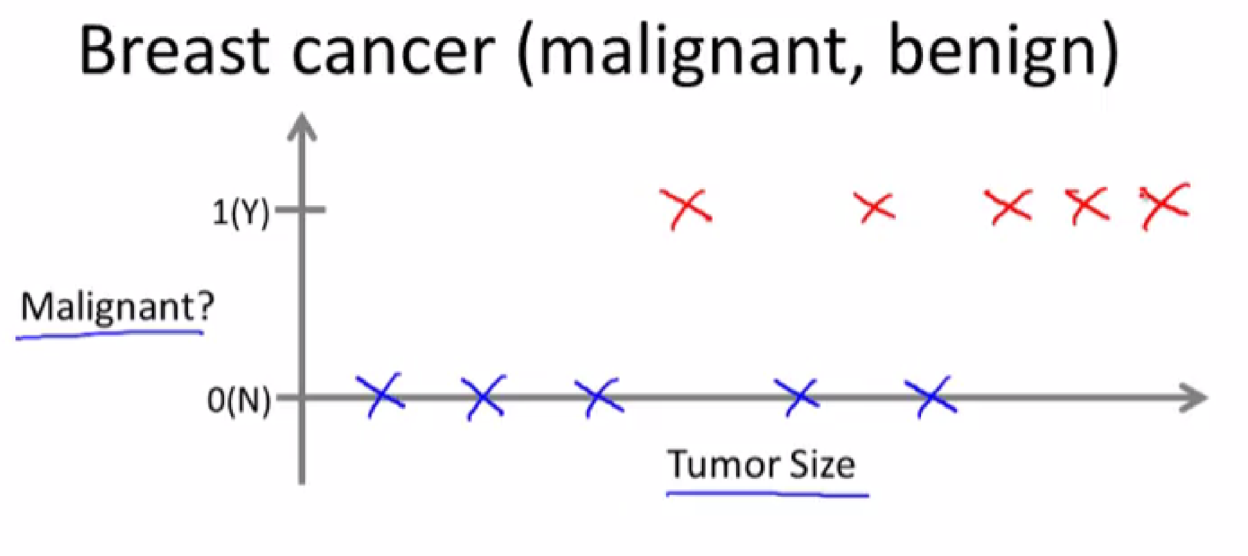
If we do that, and make a prediction here, then it looks like, well, maybe we can sell the house for closer to $200,000.

This is an example of a supervised learning algorithm.

The term supervised learning refers to the fact that we gave the algorithm a data set in which the **"right answers"** were given.

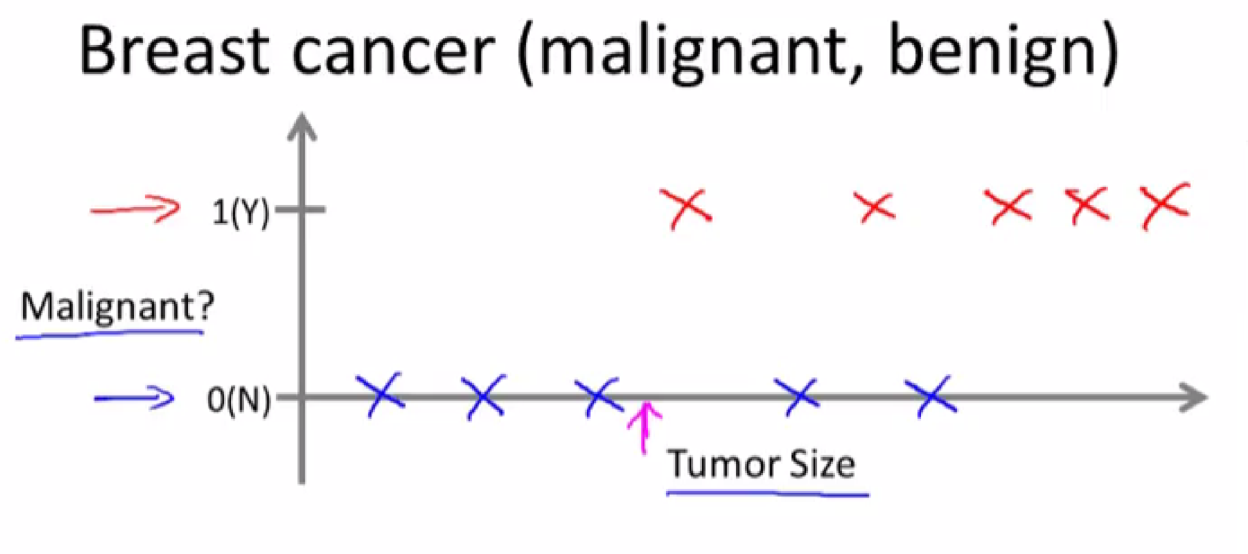
The example above is also called a regression problem. A regression problem is when we try to predict a **continuous** value output. Namely the price in the example.

Here's another supervised learning example. Let's say we want to look at medical records and try to predict of a breast cancer as malignant or benign. If someone discovers a breast tumor, a lump in their breast, a malignant tumor is a tumor that is harmful and dangerous and a benign tumor is a tumor that is harmless. Let's see a collected data set and suppose in the data set we have the size of the tumor on the horizontal axis and on the vertical axis we plot one or zero, yes or no, whether or not these are examples of tumors we've seen before are malignant (which is one) or zero if not malignant or benign.



In this data set we have five examples of benign tumors, and five examples of malignant tumors.

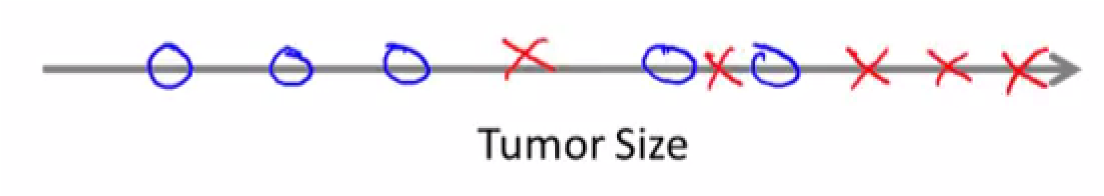
Let's say a person who tragically has a breast tumor, and let's say her breast tumor size is known (rose arrow in the following figure).



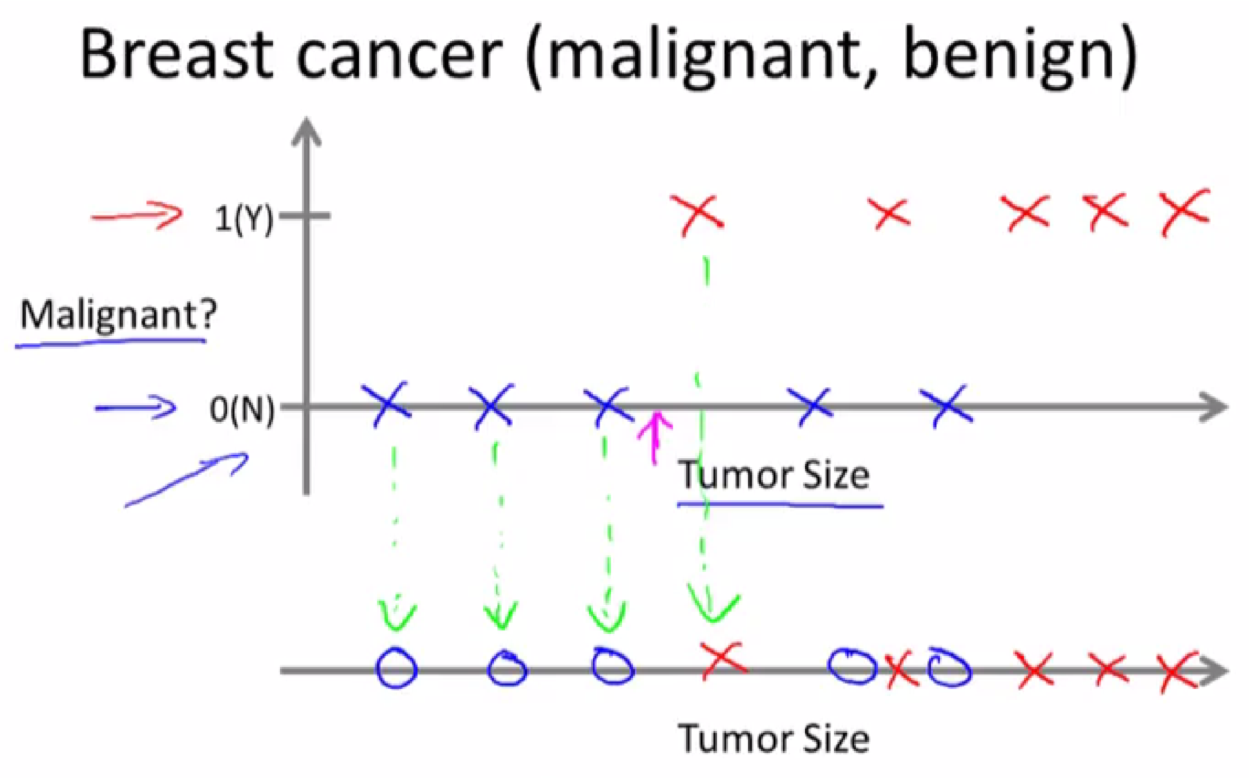
The machine learning question is, can you estimate what is the probability that a tumor is malignant versus benign? To introduce a bit more terminology this is an example of a ***classification*** problem.

The term classification refers to the fact that here we're trying to predict a **discrete** value output: zero or one, malignant or benign. And it turns out that in classification problems sometimes you can have more than two values for the two possible values for the output.

In classification problems there is another way to plot this data. Let's use a slightly different set of symbols to plot this data. So if tumor size is going to be the attribute that we are going to use to predict malignancy or benignness, we can also draw the data like this.



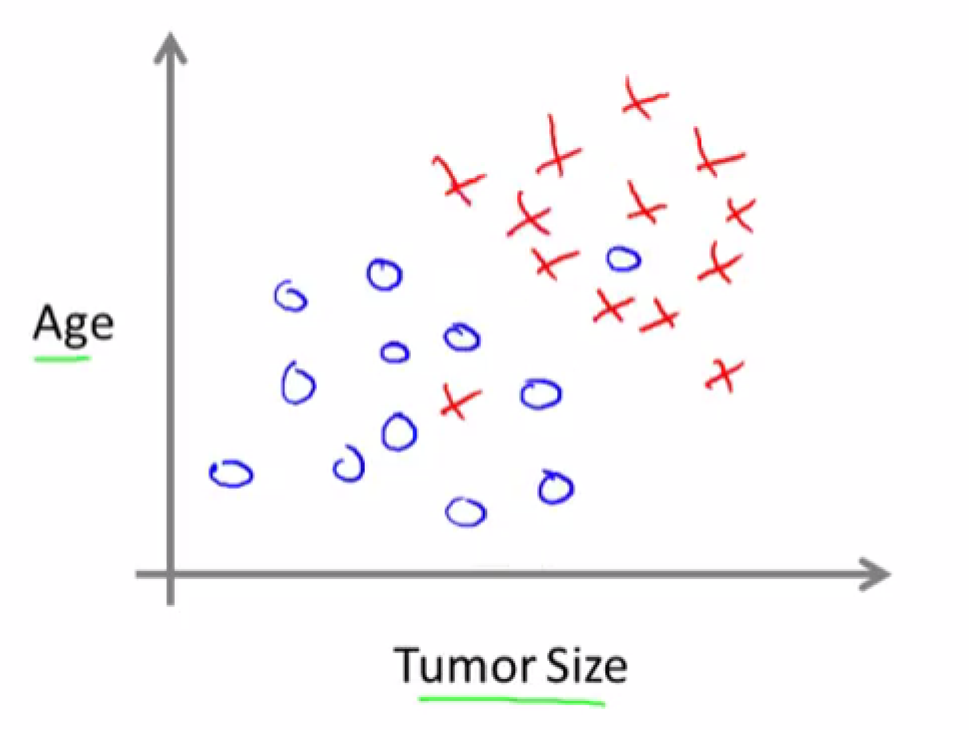
All we did was we took the data set on top and just mapped it down using different symbols. So instead of drawing crosses, we are now going to draw `O`'s for the benign tumors.



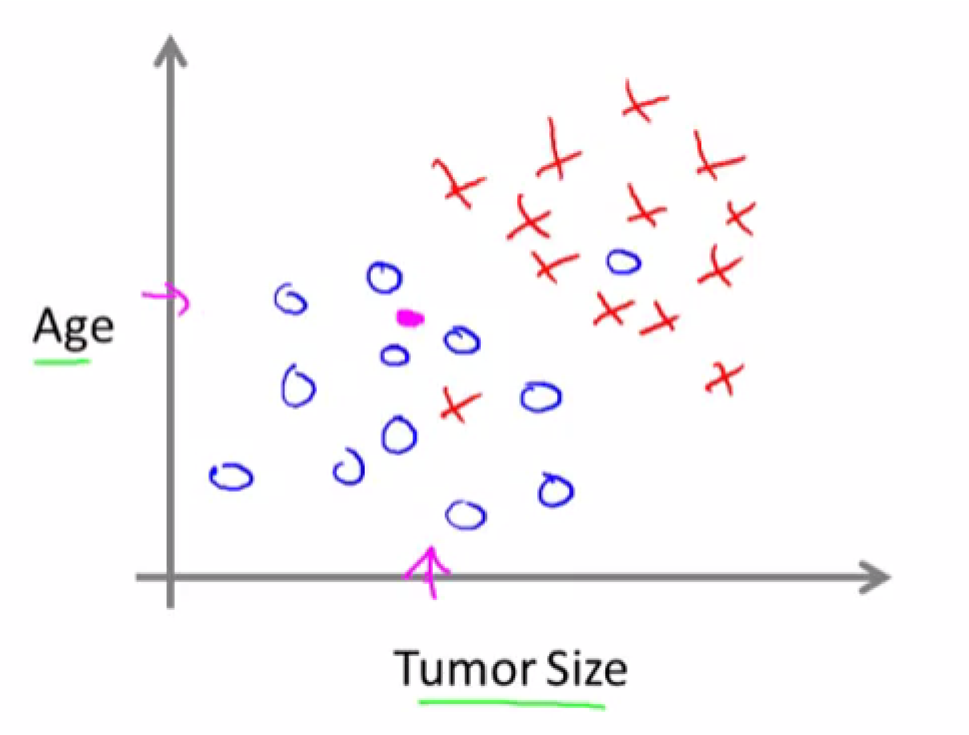
Now, in this example we use only one **feature** or one attribute, mainly, the *tumor size* in order to predict whether the tumor is malignant or benign.

In other machine learning problems we may have more than one feature.

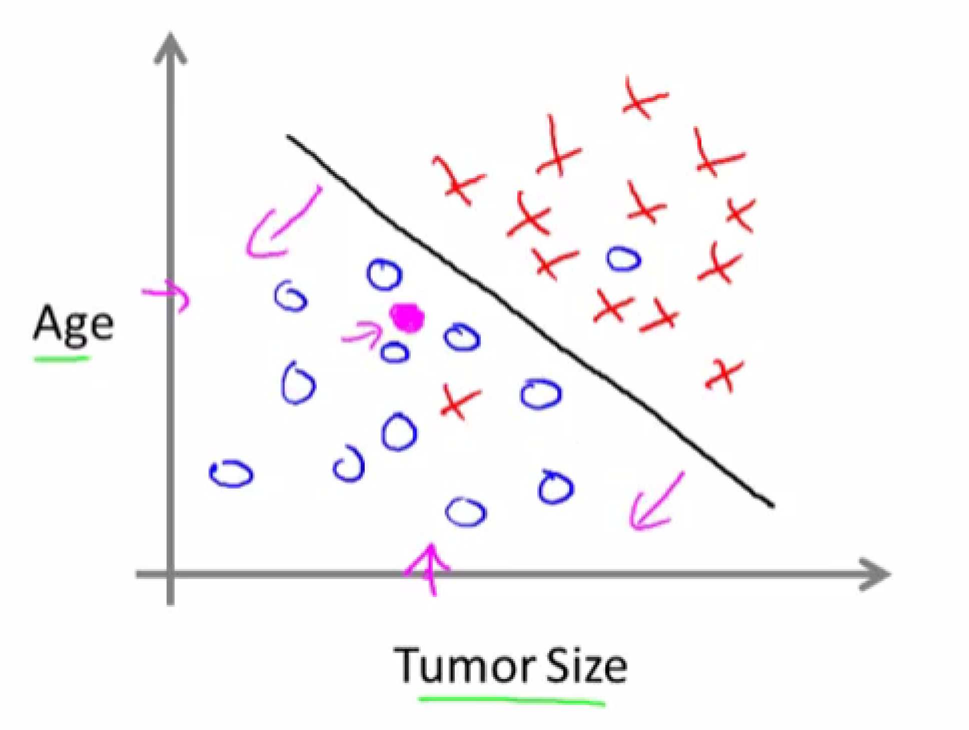
Here's an example. Let's say that instead of just knowing the tumor size, we know both the age of the patients and the tumor size. In that case maybe the data set will look like this.



So, let's say a person who tragically has a tumor. And maybe, their tumor size and age falls around there (rose point):



So given a data set like this, what the learning algorithm might do is throw a straight line through the data to try to separate out the malignant tumors from the benign ones. And with this, hopefully we can decide that the person's tumor falls on this benign side and is therefore more likely to be benign than malignant.



In this example we had **two features**, namely, the age of the patient and the size of the tumor. In other machine learning problems we will often have more features.

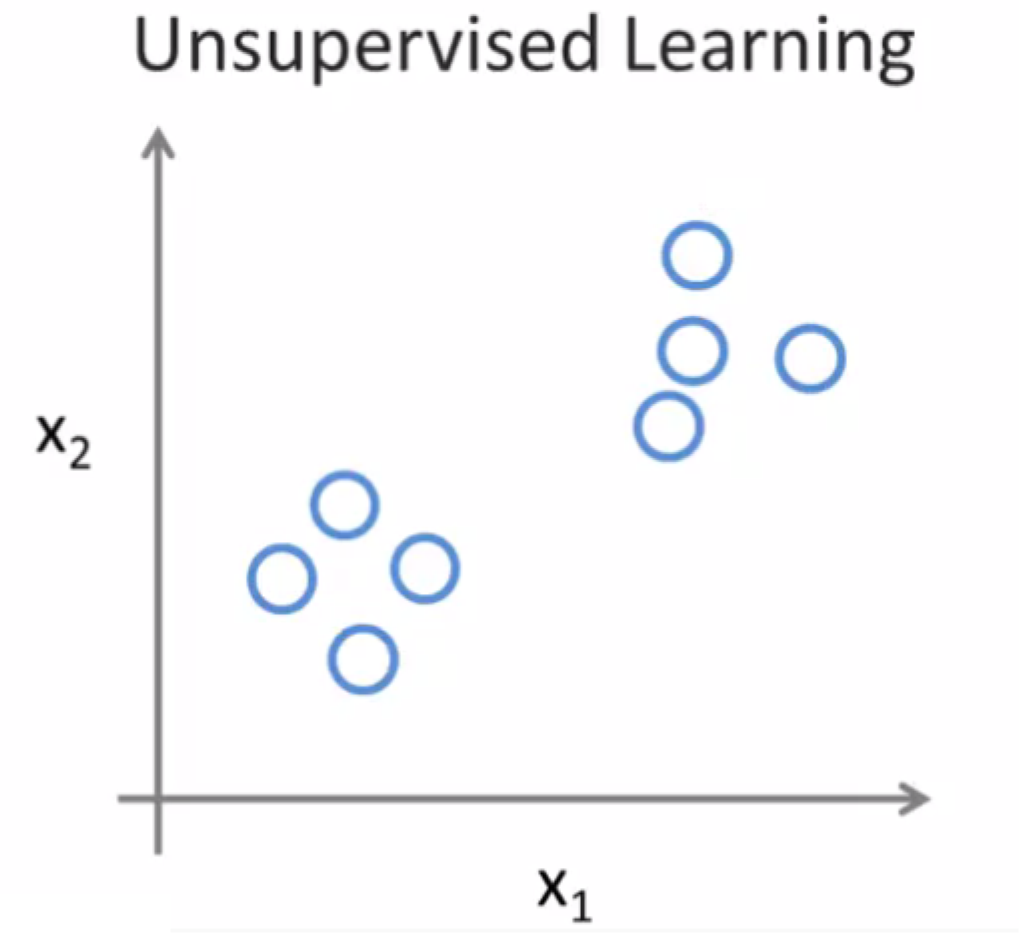
Most interesting learning algorithms is a learning algorithm that can deal with, not just two or three or five features, but an **infinite number of features**. So how do you deal with an infinite number of features. How do you even store an infinite number of things on the computer when your computer is gonna run out of memory.

Unsupervised Learning

The second major type of machine learning problem is called Unsupervised Learning.

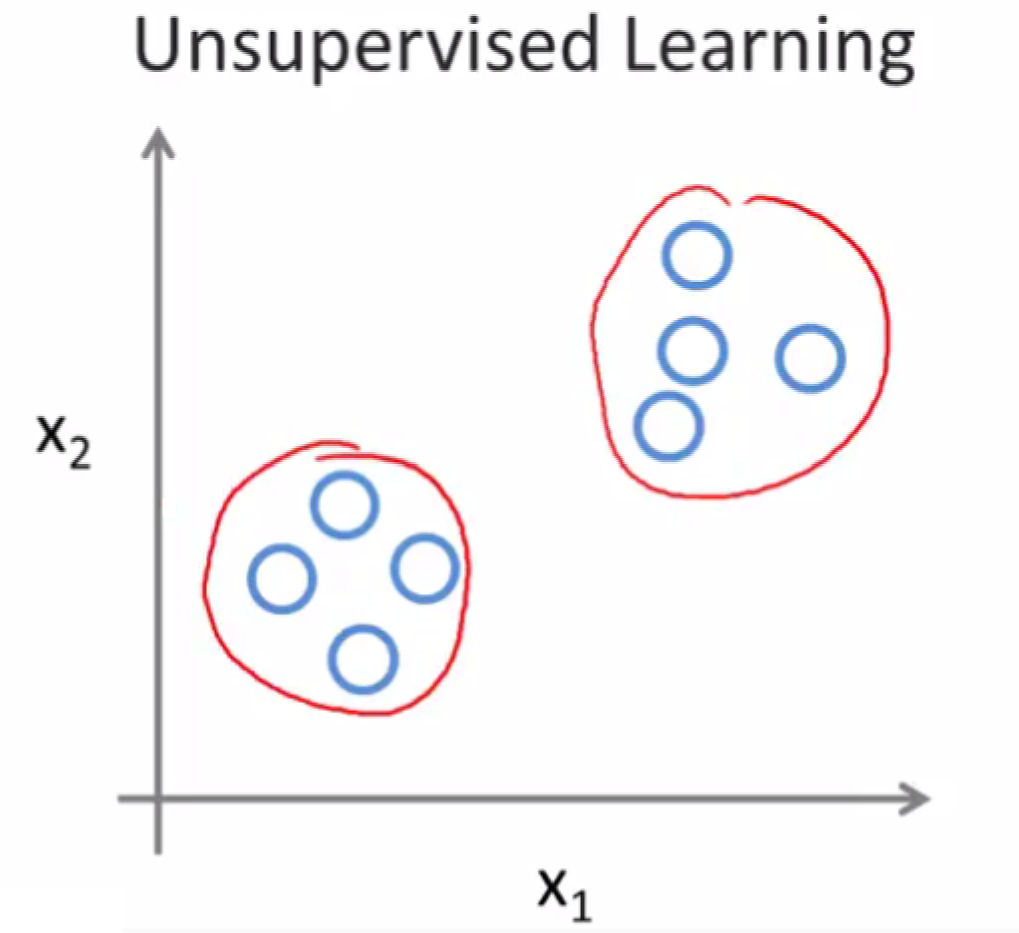
The difference between Unsupervised Learning and Supervised Learning is that in Supervised Learning we are told explicitly what is the so-called right answers (data are labeled).

In Unsupervised Learning, we're given data that doesn't have any labels or that all has the same label or really no labels. Like in this example:



So we're given the data set and we're not told what to do with it and we're not told what each data point is. Instead we're just told, here is a data set. Can you find some structure in the data?

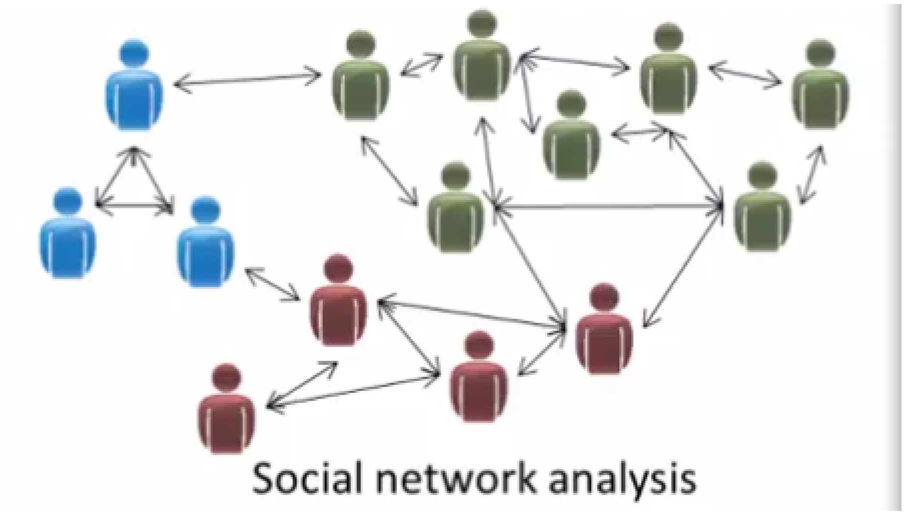
Given this data set, an Unsupervised Learning algorithm might decide that the data lives in two different clusters.



This is called a **clustering** algorithm.

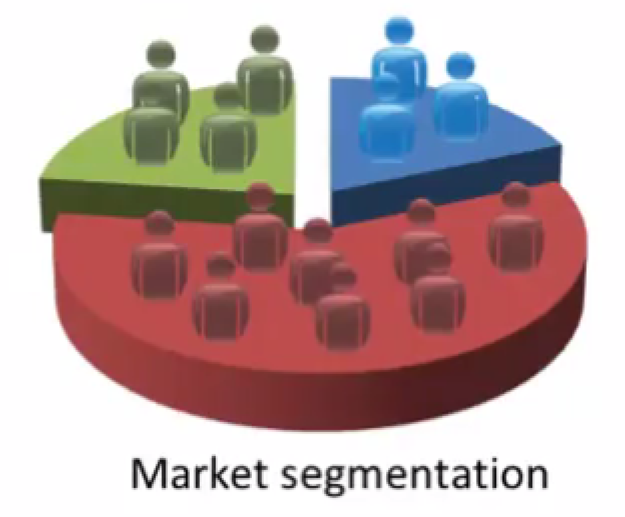
Here are two examples where Unsupervised Learning or clustering is used.

Social network analysis:



So given knowledge about which friends you email the most or given your Facebook friends or your Google+ circles, can we automatically identify which are cohesive groups of friends, also which are groups of people that all know each other?

Market segmentation:



Many companies have huge databases of customer information. So, can you look at this customer data set and automatically discover market segments and automatically group your customers into different market segments so that you can automatically and more efficiently sell or market your different market segments together?

This is Unsupervised Learning because we have all this customer data, but we don't know in advance what are the market segments and for the customers in our data set, we don't know in advance who is in market segment one, who is in market segment two, and so on. But we have to let the algorithm discover all this just from the data.

Part I

Supervised Learning

Part II

Regression

Chapter 1

Linear Regression

1.1 Notation

In general, we will let x_{ij} represent the value of the jth variable for the ith observation, where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, p$. We will use i to index the samples or observations (from 1 tp n) and j will be used to index the variables (or features) (from 1 to p). We let $\mathbf{X}$ denote a $n \times p$ matrix whose (i, j)th element is x_{ij}. That is,

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & x_{13} & \ldots & x_{1p} \\ x_{21} & x_{22} & x_{23} & \ldots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \ldots & x_{np} \end{pmatrix}$$

Note that it is useful to visualize $\mathbf{X}$ as a spreadsheet of numbers with n rows and p columns. We will write the rows of $\mathbf{X}$ as $x_1, x_2, \ldots, x_n$. Here x_i is a vector of length p, containing the p variable measurements for the ith observation. That is,

$$x_i = \begin{pmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{pmatrix}$$

(Vectors are by default represented as columns.)

We will write the columns of $\mathbf{X}$ as $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p$. Each is a vector of length n. That is,

$$\mathbf{x}_j = \begin{pmatrix} \mathbf{x}_{1j} \\ \mathbf{x}_{2j} \\ \vdots \\ \mathbf{x}_{nj} \end{pmatrix}$$

Using this notation, the matrix $\mathbf{X}$ can be written as

$$\mathbf{X} = (\mathbf{x}_1 \mathbf{x}_2 \ldots \mathbf{x}_p)$$

or

$$\mathbf{X} = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix}$$

The ^{T} notation denotes the transpose of a matrix or vector.

We use y_i to denote the ith observation of the variable on which we wish to make predictions. We write the set of all n observations in vector form as

$$\mathbf{y} = \begin{pmatrix} y_1^T \\ y_2^T \\ \vdots \\ y_n^T \end{pmatrix}$$

Then the observed data consists of $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where each x_i is a vector of length p. (If $p = 1$, then x_i is simply a scalar).

1.2 Model Representation

Let's consider the example about predicting housing prices. We're going to use this data set as an example,

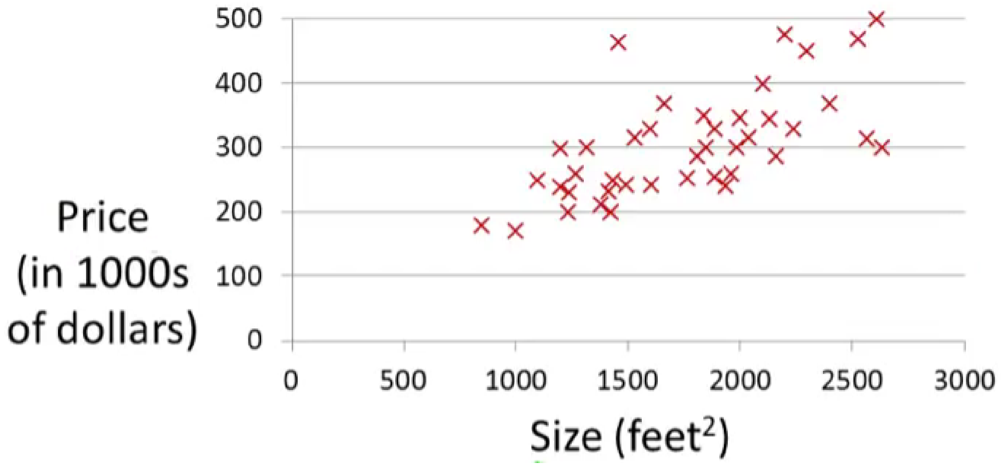


Figure 1.1:

Suppose that there is a person trying to sell a house of size 1250 square feet and he wants to know how much he might be able to sell the house for. One thing we could do is fit a model. Maybe fit a straight line to this data. Looks something like this,

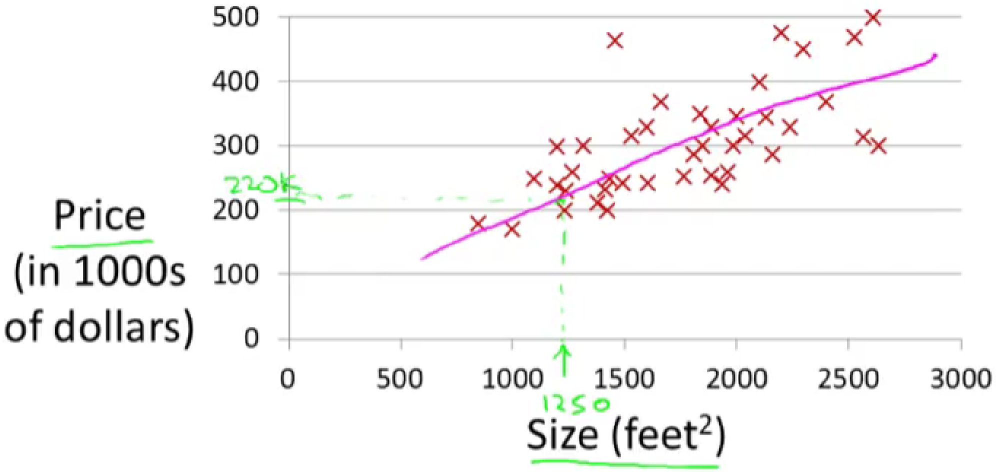


Figure 1.2:

and based on that, maybe he can sell the house for around $220,000. Recall that this is an example of a supervised learning algorithm. And it's supervised learning because we're given the "right answer" for each of our examples. More precisely, this is an example of a regression problem where the term regression refers to the fact that we are predicting a real-valued output namely the price.

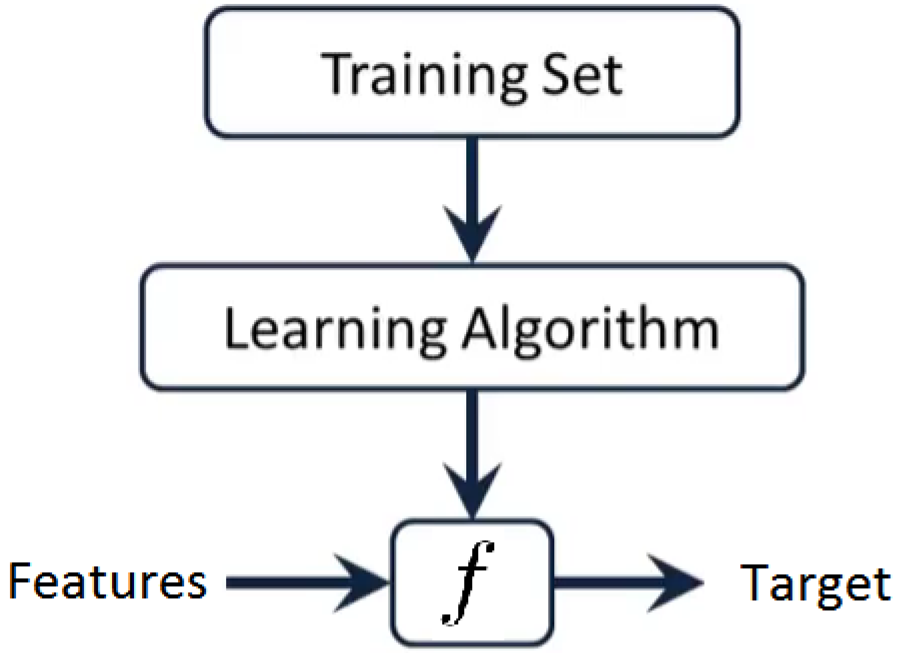
More formally, in supervised learning, we have a data set and this data set is called a **training set**. So for housing prices example, we have a training set of different housing prices and our job is to learn from this data how to predict prices of the houses.

Let's define some notation from this data set:

- The size of the house is the input variable.
- The house price is the output variable.
- The input variables are typically denoted using the variable symbol X,
- The inputs go by different names, such as *predictors*, *independent variables*, *features*, *predictor* or sometimes just *variables*.
- The output variable is often called the *response*, *dependent variable* or *target*, and is typically denoted using the symbol Y.
- (x_i, y_i) is the ith training example.
- The set of $\{(x_i, y_i)\}$ is the training set.
- n is the number of training examples.

So here's how this supervised learning algorithm works. Suppose that we observe a quantitative response Y and p different predictors, $X_1, X_2, \ldots, X_p$. We assume that there is some relationship between Y and $X = (X_1, X_2, \ldots, X_p)$, which can be written in the very general form

$$Y = f(X) + \epsilon$$



Here f is some fixed but unknown function of $X_1, X_2, \ldots, X_p$, and ϵ is a random error term, which is independent of X and has mean zero. The f function is also called *hypothesis* in Machine Learning. In general, the function f may involve more than one input variable. In essence, Supervised Learning refers to a set of approaches for estimating f.

1.3 Why Estimate f ?

There are two main reasons that we may wish to estimate f: *prediction* and *inference.*

Prediction

In many situations, a set of inputs X are readily available, but the output Y cannot be easily obtained. In this setting, since the error term averages to zero, we can predict Y using

$$\hat{Y} = \hat{f}(X)$$

where $\hat{f}$ represents our estimate for f, and $\hat{Y}$ represents the resulting prediction for Y. Like in the example above about predicting housing prices.

We can measure the accuracy of $\hat{Y}$ by using a **cost function**. In the regression models, the most commonly-used measure is the *mean squared error* (MSE), given by

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

Inference

We are often interested in understanding the way that Y is affected as $X_1, X_2, \ldots, X_p$ change. In this situation we wish to estimate f , but our goal is not necessarily to make predictions for Y. We instead want to understand the relationship between X and Y, or more specifically, to understand how Y changes as a function of $X_1, X_2, \ldots, X_p$. In this case, one may be interested in answering the following questions:

- Which predictors are associated with the response?
- What is the relationship between the response and each predictor?
- Can the relationship between Y and each predictor be adequately summarized using a linear equation, or is the relationship more complicated?

1.4 Simple Linear Regression Model

Simple linear regression is a very straightforward approach for predicting a quantitative response Y on the basis of a single predictor variable X. It assumes that there is approximately a linear relationship between X and Y. Mathematically, we can write this linear relationship as

$$\begin{gathered} Y = \beta_0 + \beta_1 X + \epsilon \\ Y \approx \beta_0 + \beta_1 X \end{gathered}$$

where β_0 and β_1 are two unknown constants that represent the *intercept* and *slope*, also known as ***coefficients*** or *parameters*, and ϵ is the error term.

Given some estimates $\hat{\beta_0}$ and $\hat{\beta_1}$ for the model coefficients, we predict future inputs x using

$$\hat{y} = \hat{\beta_0} + \hat{\beta_1} x$$

where $\hat{y}$ indicates a prediction of Y on the basis of $X = x$. The *hat* symbol, $\hat{}$, denotes an estimated value.

1.5 Estimating the Coefficients

Let $\hat{y}_i = \hat{\beta_0} + \hat{\beta_1} x_i$ be the prediction for Y based on the ith value of X. Then $e_i = y_i - \hat{y}_i$ represents the ith ***residual***.

We define the ***residual sum of squares*** (**RSS**) as

$$\begin{aligned} RSS &= e_1^2 + e_2^2 + \ldots + e_n^2 \\ &= \sum_{i=1}^{n} e_i^2 \end{aligned}$$

or equivantly as

$$\begin{aligned} RSS &= (y_1 - \hat{\beta_0} - \hat{\beta_1} x_1)^2 + (y_2 - \hat{\beta_0} - \hat{\beta_1} x_2)^2 + \ldots + (y_n - \hat{\beta_0} - \hat{\beta_1} x_n)^2 \\ &= \sum_{i=1}^{n} (y_i - \hat{\beta_0} - \hat{\beta_1} x_i)^2 \end{aligned}$$

The *least squares* approach chooses $\hat{\beta_0}$ and $\hat{\beta_1}$ to minimize the RSS. The minimizing values can be show to be

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{s_{xy}}{s_x^2}$$

and

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

where:

- $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the *sample mean.*
- $s_x^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$ is the *sample variance.* The sample standard deviation is $s_x = \sqrt{s_x^2}$.
- $s_{xy} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$ is the *sample covariance.* It measures the degree of linear association between $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$. Once scaled by $s_x s_y$, it gives the *sample correlation coefficient*, $r_{xy} = \frac{s_{xy}}{s_x s_y}$.

Click here to see the influence of the distance employed in the sum of squares. Try to minimize the sum of squares for the different datasets. The choices of intercept and slope that minimize the sum of squared distances for a kind of distance are not the optimal for a different kind of distance.

1.6 Assessing the Accuracy of the Coefficient Estimates

The standard error of an estimator reflects how it varies under repeated sampling. We have

$$\mathrm{SE}(\hat{\beta}_1)^2 = \frac{\sigma^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\mathrm{SE}(\hat{\beta}_0)^2 = \sigma^2\left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}\right]$$

where $\sigma^2 = Var(\epsilon)$

In general, σ^2 is know known, but can be estimated from the data. The estimate of σ is known as the *residual standard error*, and is given by

$$\mathrm{RSE} = \sqrt{\frac{\mathrm{RSS}}{(n-2)}}$$

These standard errors can be used to compute *confidence intervals.* A 95% confidence interval is defined as a range of values such that with 95% probability, the range will contain the true unknown value of the parameter. It has the form

$$\hat{\beta}_1 \pm 2 \cdot \mathrm{SE}(\hat{\beta}_1)$$

That is, there is approximately a 95% chance that the interval

$$\left[\hat{\beta}_1 - 2 \cdot \mathrm{SE}(\hat{\beta}_1), \hat{\beta}_1 + 2 \cdot \mathrm{SE}(\hat{\beta}_1)\right]$$

will contain the true value of β_1. Similarly, a confidence interval for β_0 approximately takes the form

$$\hat{\beta}_0 \pm 2 \cdot \mathrm{SE}(\hat{\beta}_0)$$

Hypothesis testing

Standard errors can also be used to perform *hypothesis tests* on the coefficients. The most common hypothesis test involves testing the *null hypothesis* of

$$H_0 : \text{There is no relationship between} \, X \, \text{and} \, Y$$

versus the *alternative hypothesis*

$$H_1 : \text{There is some relationship between} \, X \, \text{and} \, Y$$

Mathematically, this corresponds to testing

$$H_0 : \beta_1 = 0$$

versus

$$H_1 : \beta_1 \neq 0$$

since if $\beta_1 = 0$ then the simple linear regression model reduces to $Y = \beta_0 + \epsilon$, and X is not associated with Y.

To test the null hypothesis H_0, we compute a ***t-statistic***, given by

$$t = \frac{\hat{\beta}_1 - 0}{\mathrm{SE}(\hat{\beta}_1)}$$

This will have a t-distribution (*Student*) with $n-2$ degrees of freedom, assuming $\beta_1 = 0$.

Using statistical software, it is easy to compute the probability of observing any value equal to $|t|$ or larger. We call this probability the ***p-value***.

If p-value is small enough (typically under 0.01 (1% error) or 0.05 (5% error)) we reject the null hypothesis, that is we declare a relationship to exist between X and Y.

1.7 ANOVA and model fit

1.7.1 ANOVA

In this section we will see how the variance of Y is decomposed into two parts, each one corresponding to the regression and to the error, respectively. This decomposition is called the *ANalysis Of VAriance* (ANOVA).

Before explaining ANOVA, it is important to recall an interesting result: *the mean of the fitted values* $\hat{Y}_1, \ldots, \hat{Y}_n$ *is the mean of* $Y_1, \ldots, Y_n$. This is easily seen if we plug-in the expression of $\hat{\beta}_0$:

$$\frac{1}{n}\sum_{i=1}^{n} \hat{Y}_i = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{\beta}_0 + \hat{\beta}_1 X_i\right) = \hat{\beta}_0 + \hat{\beta}_1 \bar{X} = \left(\bar{Y} - \hat{\beta}_1 \bar{X}\right) + \hat{\beta}_1 \bar{X} = \bar{Y}.$$

The ANOVA decomposition considers the following measures of variation related with the response:

- SST $= \sum_{i=1}^n \left(Y_i - \bar{Y}\right)^2$, the **total sum of squares**. This is the *total variation* of $Y_1, \ldots, Y_n$, since SST $= ns_y^2$, where s_y^2 is the sample variance of $Y_1, \ldots, Y_n$.
- SSR $= \sum_{i=1}^n \left(\hat{Y}_i - \bar{Y}\right)^2$, the **regression sum of squares**[1]. This is the variation explained by the regression line, that is, *the variation from $\bar{Y}$ that is explained by the estimated conditional mean* $\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i$. SSR $= ns_{\hat{y}}^2$, where $s_{\hat{y}}^2$ is the sample variance of $\hat{Y}_1, \ldots, \hat{Y}_n$.
- SSE $= \sum_{i=1}^n \left(Y_i - \hat{Y}_i\right)^2$, the **sum of squared errors**[2]. Is the variation around the conditional mean. Recall that SSE $= \sum_{i=1}^n \hat{\varepsilon}_i^2 = (n-2)\hat{\sigma}^2$, where $\hat{\sigma}^2$ is the sample variance of $\hat{\varepsilon}_1, \ldots, \hat{\varepsilon}_n$.

The ANOVA decomposition is

$$\underbrace{\text{SST}}_{\text{Variation of } Y_i' s} = \underbrace{\text{SSR}}_{\text{Variation of } \hat{Y}_i' s} + \underbrace{\text{SSE}}_{\text{Variation of } \hat{\varepsilon}_i' s}$$

The graphical interpretation of this equation is shown in the following figures.

[1] Recall that SSR is different from RSS (Residual Sum of Squares

[2] Recall that SSE and RSS (for $(\hat{\beta}_0, \hat{\beta}_1)$) are just different names for referring to the same quantity: SSE $= \sum_{i=1}^n \left(Y_i - \hat{Y}_i\right)^2 = \sum_{i=1}^n \left(Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i\right)^2 = \text{RSS}\left(\hat{\beta}_0, \hat{\beta}_1\right)$.

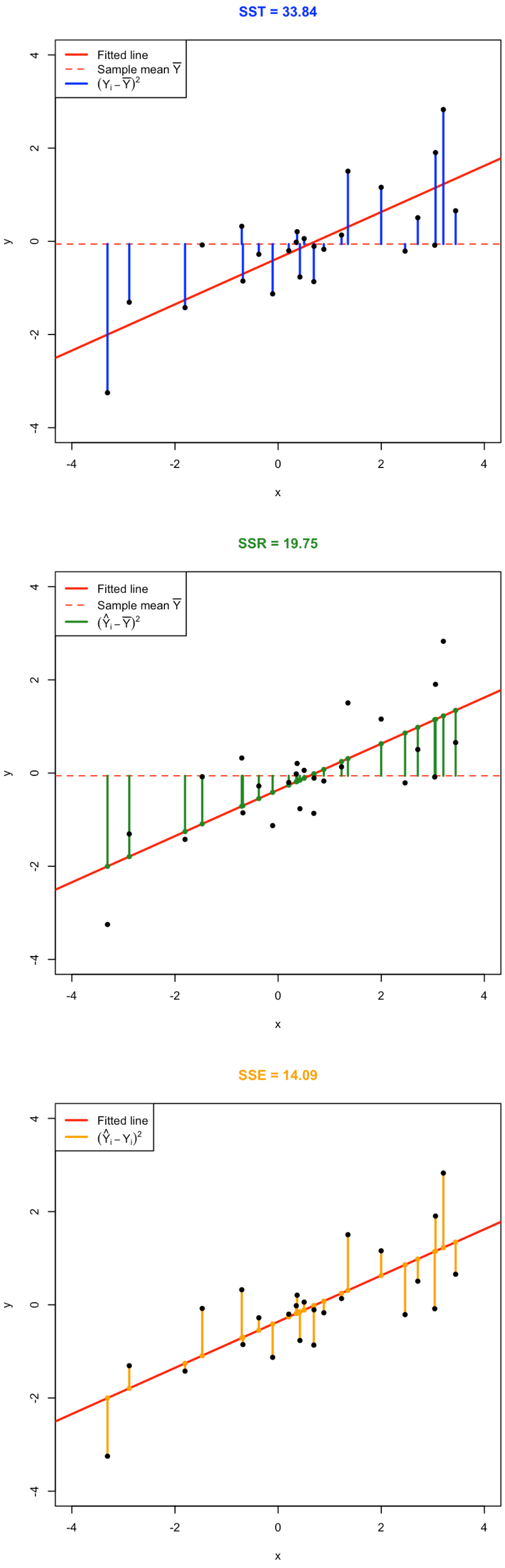


Figure 1.3: Visualization of the ANOVA decomposition. SST measures the variation of $Y_1, \ldots, Y_n$ with respect to $\bar{Y}$. SST measures the variation with respect to the conditional means, $\hat{\beta}_0 + \hat{\beta}_1 X_i$. SSE collects the variation of the residuals.

Click here to see the ANOVA decomposition and its dependence on σ^2 and $\hat{\sigma}^2$.'

The ANOVA table summarizes the decomposition of the variance. Here is given in the layout employed by `R`.

	Degrees of freedom	Sum Squares	Mean Squares	F-value	p-value
Predictor	1	SSR	$\frac{\text{SSR}}{1}$	$\frac{\text{SSR}/1}{\text{SSE}/(n-2)}$	p
Residuals	$n-2$	SSE	$\frac{\text{SSE}}{n-2}$		

The `anova` function in `R` takes a model as an input and returns the ANOVA table.

The "F-value" of the ANOVA table represents the value of the F-statistic $\frac{\text{SSR}/1}{\text{SSE}/(n-2)}$. This statistic is employed to test

$$H_0 : \beta_1 = 0 \quad \text{vs.} \quad H_1 : \beta_1 \neq 0,$$

that is, the hypothesis of no linear dependence of Y on X. The result of this test is completely equivalent to the t-test for β_1 that we saw previously in the Hypothesis testing (this is something *specific for simple linear regression* – the F-test will not be equivalent to the t-test for β_1 in the Mulitple Linear Regression).

It happens that

$$F = \frac{\text{SSR}/1}{\text{SSE}/(n-2)} \overset{H_0}{\sim} F_{1,n-2},$$

where $F_{1,n-2}$ is the *Snedecor's F distribution*[3] with 1 and $n-2$ degrees of freedom.

If H_0 is true, then F is expected to be *small* since SSR will be close to zero. The p-value of this test is the same as the p-value of the t-test for $H_0 : \beta_1 = 0$.

1.7.2 The R^2 Statistic

To calculate R^2, we use the formula

$$R^2 = \frac{\text{TSS} - \text{RSS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}}$$

where $\text{TSS} = \sum(y_i - \bar{y})^2$ is the *total sum of squared.*

R^2 measures the *proportion of variability in Y that can be explained using X*. An R^2 statistic that is close to 1 indicates that a large proportion of the variability in the response has been explained by the regression. A number near 0 indicates that the regression did not explain much of the variability in the response; this might occur because the linear model is wrong, or the inherent error σ^2 is high, or both.

It can be shown that in this simple linear linear regression setting that $R^2 = r^2$, where r is the correlation between X and Y:

[3] The $F_{n,m}$ distribution arises as the quotient of two independent random variables χ^2_n and χ^2_m, $\frac{\chi^2_n/n}{\chi^2_m/m}$.

$$r = \frac{cov(X, Y)}{\sigma_X \sigma_Y}$$

R^2 does not measure the correctness of a linear model but its **usefulness** (for prediction, for *explaining the variance* of Y), assuming the model is correct.

Trusting blindly the R^2 can lead to catastrophic conclusions, since the model may not be correct.

So remember:

A large R^2 means *nothing* if the **assumptions of the model do not hold**. R^2 is the proportion of variance of Y explained by X, but, of course, *only when the linear model is correct.*

PW 1

1.8 Some R basics

1.8.1 Basic Commands

R uses functions to perform operations. To run a function called `funcname`,we type `funcname(input1, input2)` , where the inputs (or arguments) `input1` and `input2` tell R how to run the function. A function can have any number of inputs. For example, to create a vector of numbers, we use the function `c()` (for *concatenate*).

```
x <- c(1,3,2,5)
x
#ans> [1] 1 3 2 5
```

Note that the `>` is not part of the command; rather, it is printed by R to indicate that it is ready for another command to be entered. We can also save things using `=` rather than `<-`. Note that the answer in the code above is followed by `#ans>` while in the R console it is not.

```
x = c(1,6,2)
x
#ans> [1] 1 6 2
y = c(1,4,3)
length(x)
#ans> [1] 3
length(y)
#ans> [1] 3
x+y
#ans> [1]  2 10  5
```

Hitting the *up arrow* multiple times will display the previous commands, which can then be edited. This is useful since one often wishes to repeat a similar command.

The `ls()` function allows us to look at a list of all of the objects, such `ls()` as data and functions, that we have saved so far. The `rm()` function can be used to delete any object that we don't want.

```
ls()
#ans> [1] "x" "y"
rm(x)
ls()
#ans> [1] "y"
```

1.8.2 Vectors

```
# A handy way of creating sequences is the operator :
# Sequence from 1 to 5
1:5
#ans> [1] 1 2 3 4 5

# Storing some vectors
vec <- c(-4.12, 0, 1.1, 1, 3, 4)
vec
#ans> [1] -4.12  0.00  1.10  1.00  3.00  4.00

# Entry-wise operations
vec + 1
#ans> [1] -3.12  1.00  2.10  2.00  4.00  5.00
vec^2
#ans> [1] 16.97  0.00  1.21  1.00  9.00 16.00

# If you want to access a position of a vector, use [position]
vec[6]
#ans> [1] 4

# You also can change elements
vec[2] <- -1
vec
#ans> [1] -4.12 -1.00  1.10  1.00  3.00  4.00

# If you want to access all the elements except a position, use [-position]
vec[-2]
#ans> [1] -4.12  1.10  1.00  3.00  4.00

# Also with vectors as indexes
vec[1:2]
#ans> [1] -4.12 -1.00

# And also
vec[-c(1, 2)]
#ans> [1] 1.1 1.0 3.0 4.0
```

Do the following:

- Create the vector $x = (1, 7, 3, 4)$.
- Create the vector $y = (100, 99, 98, ..., 2, 1)$.
- Compute $x_3 + y_4$ and $\cos(x_3) + \sin(x_2)e^{-y_2}$. (Answers: `100`, `-0.9899925`)
- Set $x_3 = 0$ and $y_2 = -1$. Recompute the previous expressions. (Answers: `97`, `2.785875`)
- Index y by $x + 1$ and store it as `z`. What is the output? (Answer: `z` is `c(-1, 93, 100, 96)`)

1.8.3 Matrices, data frames and lists

```
# A matrix is an array of vectors
A <- matrix(1:4, nrow = 2, ncol = 2)
```

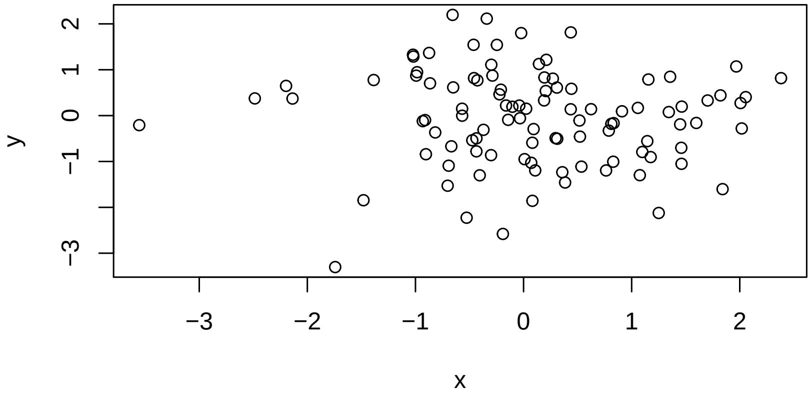
```
A
#ans>      [,1] [,2]
#ans> [1,]    1    3
#ans> [2,]    2    4

# Another matrix
B <- matrix(1:4, nrow = 2, ncol = 2, byrow = TRUE)
B
#ans>      [,1] [,2]
#ans> [1,]    1    2
#ans> [2,]    3    4

# Binding by rows or columns
rbind(1:3, 4:6)
#ans>      [,1] [,2] [,3]
#ans> [1,]    1    2    3
#ans> [2,]    4    5    6
cbind(1:3, 4:6)
#ans>      [,1] [,2]
#ans> [1,]    1    4
#ans> [2,]    2    5
#ans> [3,]    3    6

# Entry-wise operations
A + 1
#ans>      [,1] [,2]
#ans> [1,]    2    4
#ans> [2,]    3    5
A * B
#ans>      [,1] [,2]
#ans> [1,]    1    6
#ans> [2,]    6   16

# Accessing elements
A[2, 1] # Element (2, 1)
#ans> [1] 2
A[1, ] # First row
#ans> [1] 1 3
A[, 2] # First column
#ans> [1] 3 4

# A data frame is a matrix with column names
# Useful when you have multiple variables
myDf <- data.frame(var1 = 1:2, var2 = 3:4)
myDf
#ans>   var1 var2
#ans> 1    1    3
#ans> 2    2    4

# You can change names
names(myDf) <- c("newname1", "newname2")
myDf
#ans>   newname1 newname2
```

```
#ans> 1        1        3
#ans> 2        2        4

# The nice thing is that you can access variables by its name with the $ operator
myDf$newname1
#ans> [1] 1 2

# And create new variables also (it has to be of the same
# length as the rest of variables)
myDf$myNewVariable <- c(0, 1)
myDf
#ans>   newname1 newname2 myNewVariable
#ans> 1        1        3             0
#ans> 2        2        4             1

# A list is a collection of arbitrary variables
myList <- list(vec = vec, A = A, myDf = myDf)

# Access elements by names
myList$vec
#ans> [1] -4.12 -1.00  1.10  1.00  3.00  4.00
myList$A
#ans>      [,1] [,2]
#ans> [1,]    1    3
#ans> [2,]    2    4
myList$myDf
#ans>   newname1 newname2 myNewVariable
#ans> 1        1        3             0
#ans> 2        2        4             1

# Reveal the structure of an object
str(myList)
#ans> List of 3
#ans>  $ vec : num [1:6] -4.12 -1 1.1 1 3 4
#ans>  $ A   : int [1:2, 1:2] 1 2 3 4
#ans>  $ myDf:'data.frame': 2 obs. of  3 variables:
#ans>   ..$ newname1     : int [1:2] 1 2
#ans>   ..$ newname2     : int [1:2] 3 4
#ans>   ..$ myNewVariable: num [1:2] 0 1
str(myDf)
#ans> 'data.frame': 2 obs. of  3 variables:
#ans>  $ newname1     : int  1 2
#ans>  $ newname2     : int  3 4
#ans>  $ myNewVariable: num  0 1

# A less lengthy output
names(myList)
#ans> [1] "vec"  "A"    "myDf"
```

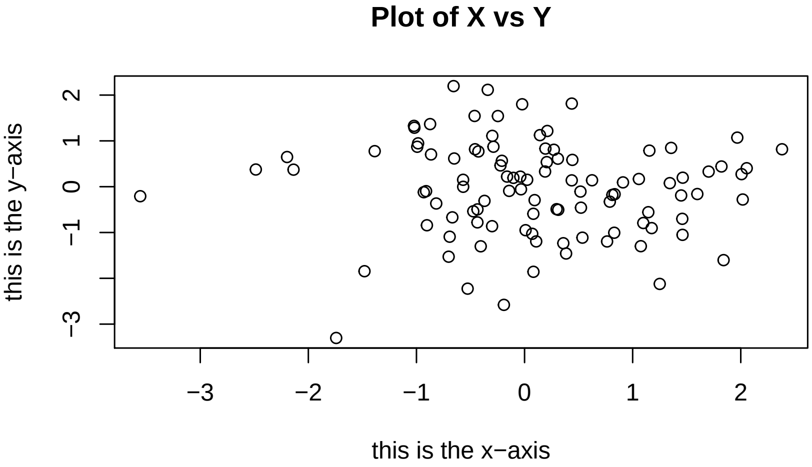
1.8.4 Graphics

The `plot()` function is the primary way to plot data in R . For instance, `plot(x,y)` produces a scatterplot of the numbers in `x` versus the numbers in `y`. There are many additional options that can be passed in to the `plot()` function. For example, passing in the argument `xlab` will result in a label on the `x-axis`. To find out more information about the `plot()` function, type `?plot`.

```
x=rnorm(100)
# The rnorm() function generates a vector of random normal variables,
# rnorm() with first argument n the sample size. Each time we call this
# function, we will get a different answer.
y=rnorm(100)
plot(x,y)
```

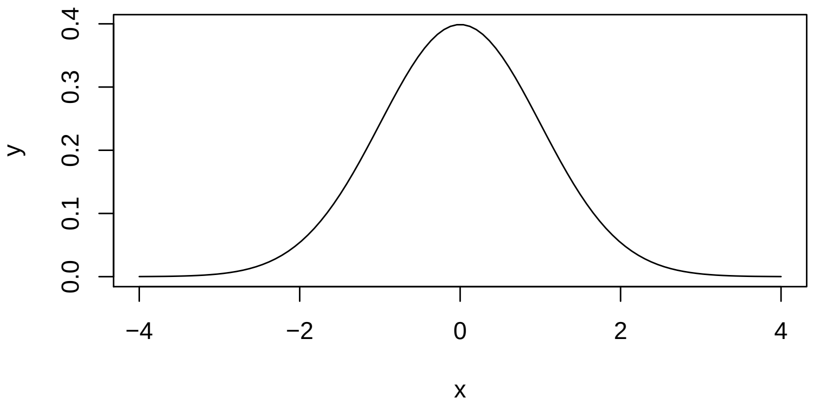


```
# with titles
plot(x,y,xlab="this is the x-axis",ylab="this is the y-axis",
main="Plot of X vs Y")
```



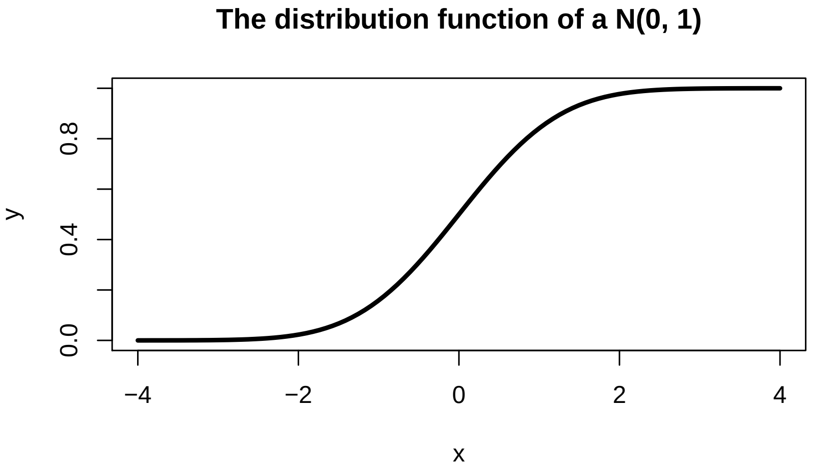
1.8.5 Distributions

```
# R allows to sample [r], compute density/probability mass [d],
# compute distribution function [p] and compute quantiles [q] for several
# continuous and discrete distributions. The format employed is [rdpq]name,
# where name stands for:
# - norm -> Normal
# - unif -> Uniform
# - exp -> Exponential
# - t -> Student's t
# - f -> Snedecor's F (Fisher)
# - chisq -> Chi squared
# - pois -> Poisson
# - binom -> Binomial
# More distributions: ?Distributions


# Sampling from a Normal - 100 random points from a N(0, 1)
rnorm(n = 10, mean = 0, sd = 1)
#ans>  [1]  0.2067  0.0958 -1.2583 -1.0032  1.3780  1.3026 -1.1420  0.4348
#ans>  [9]  1.0992 -0.9037

# If you want to have always the same result, set the seed of the random number
# generator
set.seed(45678)
rnorm(n = 10, mean = 0, sd = 1)
#ans>  [1]  1.440 -0.720  0.671 -0.422  0.378 -1.667 -0.508  0.443 -1.799 -0.618

# Plotting the density of a N(0, 1) - the Gauss bell
x <- seq(-4, 4, l = 100)
y <- dnorm(x = x, mean = 0, sd = 1)
plot(x, y, type = "l")
```

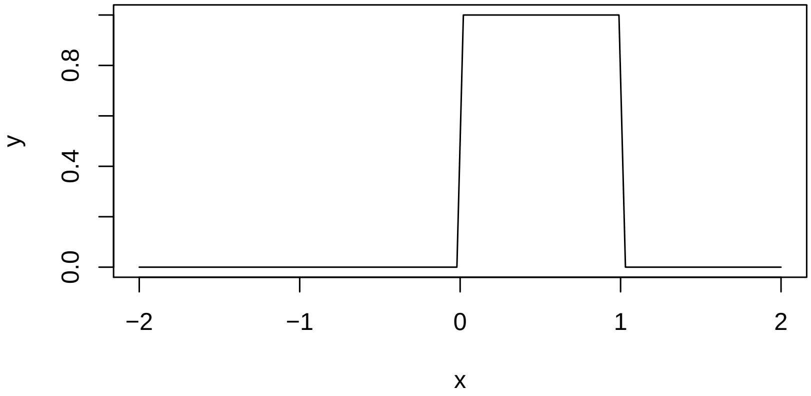


```
# Plotting the distribution function of a N(0, 1)
x <- seq(-4, 4, l = 100)
y <- pnorm(q = x, mean = 0, sd = 1)
```

```
plot(x, y, type = "l", lwd = 3, main="The distribution function of a N(0, 1)")
```



```
# Computing the 95% quantile for a N(0, 1)
qnorm(p = 0.95, mean = 0, sd = 1)
#ans> [1] 1.64

# All distributions have the same syntax: rname(n,...), dname(x,...), dname(p,...)
# and qname(p,...), but the parameters in ... change. Look them in ?Distributions
# For example, here is que same for the uniform distribution

# Sampling from a U(0, 1)
set.seed(45678)
runif(n = 10, min = 0, max = 1)
#ans>  [1] 0.9251 0.3340 0.2359 0.3366 0.7489 0.9327 0.3365 0.2246 0.6474 0.0808

# Plotting the density of a U(0, 1)
x <- seq(-2, 2, l = 100)
y <- dunif(x = x, min = 0, max = 1)
plot(x, y, type = "l")
```



```
# Computing the 95% quantile for a U(0, 1)
qunif(p = 0.95, min = 0, max = 1)
#ans> [1] 0.95
```

Do the following:

- Compute the 90%, 95% and 99% quantiles of a F distribution with `df1 = 1` and `df2 = 5`. (Answer: `c(4.060420, 6.607891, 16.258177)`)
- Sample 100 points from a Poisson with `lambda = 5`.
- Plot the density of a t distribution with `df = 1` (use a sequence spanning from `-4` to `4`). Add lines of different colors with the densities for `df = 5`, `df = 10`, `df = 50` and `df = 100`.

1.8.6 Working directory

Your *working directory* is the folder on your computer in which you are currently working. When you ask R to open a certain file, it will look in the working directory for this file, and when you tell R to save a data file or figure, it will save it in the working directory.

To set your working directory within RStudio you can go to `Tools / Set working directory`, or use the command `setwd()`, we put the complete path of the directory between the brackets, do not forget to put the path into quotation marks `""`.

To know the actual working directory we use `getwd()`.

1.8.7 Loading Data

The `read.table()` function is one of the primary ways to import a data set into `R`. The help file `?read.table()` contains details about how to use this function. We can use the function `write.table()` to export data.

Next we will show how to load the data set `Auto.data`.

```
Auto=read.table("Auto.data",header=T,na.strings ="?")
# For this file we needed to tell R that the first row is the
# names of the variables.
# na.strings tells R that any time it sees a particular character
# or set of characters (such as a question mark), it should be
# treated as a missing element of the data matrix.
```

- If the file is of csv format, we use `read.csv`.
- Always try to look to the file before importing it to `R` (Open it in a text editor. See for example if the first row containes the variables names, if the columns are separated by `,` or `;` or ..
- For text editors, I suggest `Sublime Text` or `Atom`.

```
dim(Auto) # To see the dimensions of the data set
#ans> [1] 397   9
nrow(Auto) # To see the number of rows
#ans> [1] 397
ncol(Auto) # To see the number of columns
#ans> [1] 9
Auto[1:4,] # The first 4 rows of the data set
#ans>   mpg cylinders displacement horsepower weight acceleration year origin
#ans> 1  18         8          307        130   3504         12.0   70      1
#ans> 2  15         8          350        165   3693         11.5   70      1
#ans> 3  18         8          318        150   3436         11.0   70      1
#ans> 4  16         8          304        150   3433         12.0   70      1
#ans>                         name
#ans> 1 chevrolet chevelle malibu
#ans> 2         buick skylark 320
#ans> 3        plymouth satellite
#ans> 4             amc rebel sst
```

```
# Once the data are loaded correctly, we can use names()
# to check the variable names.
names(Auto)
#ans> [1] "mpg"          "cylinders"    "displacement" "horsepower"
#ans> [5] "weight"       "acceleration" "year"         "origin"
#ans> [9] "name"
```

Take a look at this (very) short introduction to R. It can be useful.

1.9 Regression

1.9.1 The `lm` function

We are going to employ the `EU` dataset. The `EU` dataset contains 28 rows with the member states of the European Union (Country), the number of seats assigned under different years (Seats2011, Seats2014), the Cambridge Compromise apportionment (CamCom2011), and the countries population (Population2010,Population2013).

```
# Load the dataset, when we load an .RData using load()
# function we do not attribute it to a name like we did
# when we used read.table() or when we use read.csv()
```

```
load("EU.RData")
```

There is two ways to tell R where is the file you want to load/use/import or where to save a file when you write/export/save :

1. write the complete path of the files.
2. set a working directory and put the files in it.

```
# lm (for linear model) has the syntax:
# lm(formula = response ~ predictor, data = data)
# The response is the y in the model. The predictor is x.
# For example (after loading the EU dataset)
mod <- lm(formula = Seats2011 ~ Population2010, data = EU)

# We have saved the linear model into mod, which now contains all the output of lm
# You can see it by typing
mod
#ans>
#ans> Call:
#ans> lm(formula = Seats2011 ~ Population2010, data = EU)
#ans>
#ans> Coefficients:
#ans>    (Intercept)  Population2010
#ans>       7.91e+00        1.08e-06

# mod is indeed a list of objects whose names are
names(mod)
#ans>  [1] "coefficients"  "residuals"     "effects"       "rank"
#ans>  [5] "fitted.values" "assign"        "qr"            "df.residual"
#ans>  [9] "na.action"     "xlevels"       "call"          "terms"
#ans> [13] "model"

# We can access these elements by $
# For example
mod$coefficients
#ans>    (Intercept) Population2010
#ans>       7.91e+00       1.08e-06

# The residuals
mod$residuals
#ans>        Germany         France United Kingdom          Italy          Spain
#ans>         2.8675        -3.7031        -1.7847         0.0139        -3.5084
#ans>         Poland        Romania    Netherlands         Greece        Belgium
#ans>         1.9272         1.9434         0.2142         1.8977         2.3994
#ans>       Portugal Czech Republic        Hungary         Sweden        Austria
#ans>         2.6175         2.7587         3.2898         2.0163         2.0575
#ans>       Bulgaria        Denmark       Slovakia        Finland        Ireland
#ans>         1.9328        -0.8790        -0.7606        -0.6813        -0.7284
#ans>      Lithuania         Latvia       Slovenia        Estonia         Cyprus
#ans>         0.4998        -1.3347        -2.1175        -3.3552        -2.7761
#ans>     Luxembourg          Malta
#ans>        -2.4514        -2.3553
```

```
# The fitted values
mod$fitted.values
#ans>        Germany         France United Kingdom          Italy          Spain
#ans>          96.13          77.70          74.78          72.99          57.51
#ans>         Poland        Romania    Netherlands         Greece        Belgium
#ans>          49.07          31.06          25.79          20.10          19.60
#ans>       Portugal Czech Republic        Hungary         Sweden        Austria
#ans>          19.38          19.24          18.71          17.98          16.94
#ans>       Bulgaria        Denmark       Slovakia        Finland        Ireland
#ans>          16.07          13.88          13.76          13.68          12.73
#ans>      Lithuania         Latvia       Slovenia        Estonia         Cyprus
#ans>          11.50          10.33          10.12           9.36           8.78
#ans>     Luxembourg          Malta
#ans>           8.45           8.36

# Summary of the model
sumMod <- summary(mod)
sumMod
#ans> 
#ans> Call:
#ans> lm(formula = Seats2011 ~ Population2010, data = EU)
#ans> 
#ans> Residuals:
#ans>    Min     1Q Median     3Q    Max 
#ans> -3.703 -1.951  0.014  1.980  3.290 
#ans> 
#ans> Coefficients:
#ans>                Estimate Std. Error t value Pr(>|t|)    
#ans> (Intercept)    7.91e+00   5.66e-01    14.0  2.6e-13 ***
#ans> Population2010 1.08e-06   1.92e-08    56.3  < 2e-16 ***
#ans> ---
#ans> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#ans> 
#ans> Residual standard error: 2.29 on 25 degrees of freedom
#ans>   (1 observation deleted due to missingness)
#ans> Multiple R-squared:  0.992,  Adjusted R-squared:  0.992 
#ans> F-statistic: 3.17e+03 on 1 and 25 DF,  p-value: <2e-16
```

The following table contains a handy cheat sheet of equivalences between `R` code and some of the statistical concepts associated to linear regression.

R	Statistical concept
`x`	Predictor $X_1, \ldots, X_n$
`y`	Response $Y_1, \ldots, Y_n$
`data <- data.frame(x = x, y = y)`	Sample $(X_1, Y_1), \ldots, (X_n, Y_n)$
`model <- lm(y ~ x, data = data)`	Fitted linear model
`model$coefficients`	Fitted coefficients $\hat\beta_0, \hat\beta_1$
`model$residuals`	Fitted residuals $\hat\varepsilon_1, \ldots, \hat\varepsilon_n$
`model$fitted.values`	Fitted values $\hat Y_1, \ldots, \hat Y_n$
`model$df.residual`	Degrees of freedom $n - 2$
`summaryModel <- summary(model)`	Summary of the fitted linear model
`summaryModel$sigma`	Fitted residual standard deviation $\hat\sigma$
`summaryModel$r.squared`	Coefficient of determination R^2

R	Statistical concept
`summaryModel$fstatistic`	F-test
`anova(model)`	ANOVA table

Do the following:

- Download The 'EU' dataset from here as an `.RData` file and load it using the function `load`.
- Compute the regression of `CamCom2011` into `Population2010`. Save that model as the variable `myModel`.
- Access the objects `residuals` and `coefficients` of `myModel`.
- Compute the summary of `myModel` and store it as the variable `summaryMyModel`.
- Access the object `sigma` of `myModel`.

1.9.2 Predicting House Value: Boston dataset

We are going to use a dataset called Boston which is part of the `MASS` package. It recordes the median value of houses for 506 neighborhoods around Boston. Our task is to predict the median house value (`medv`) using only one predictor (`lstat`: percent of households with low socioeconomic status).

```r
# First, install the MASS package using the command: install.packages("MASS")

# load MASS package
library(MASS)

# Check the dimensions of the Boston dataset
dim(Boston)
#ans> [1] 506  14
```

STEP 1: Split the dataset

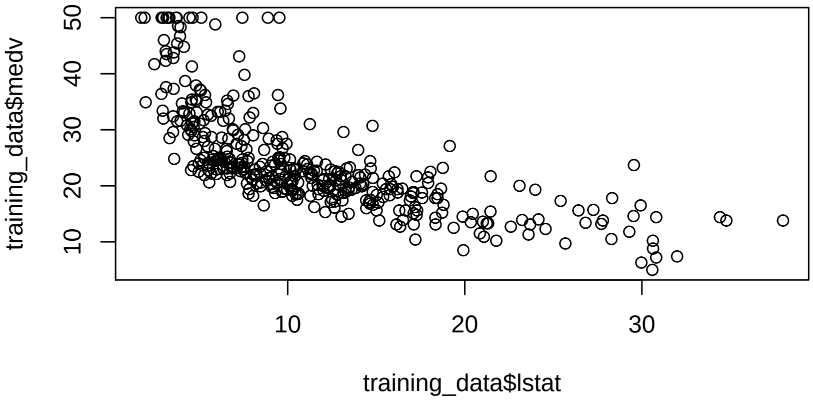
```r
# Split the data by using the first 400 observations as the training
# data and the remaining as the testing data
train = 1:400
test = -train

# Speficy that we are going to use only two variables (lstat and medv)
variables = which(names(Boston) ==c("lstat", "medv"))
training_data = Boston[train, variables]
testing_data = Boston[test, variables]

# Check the dimensions of the new dataset
dim(training_data)
#ans> [1] 400   2
```

STEP 2: Check for Linearity

In order to perfom linear regression in `R`, we will use the function `lm()`to fit a simple linear regression with `medv` as the response (dependent variable) and `lstat` as the predictor or independent variable, and then save it in `model`.

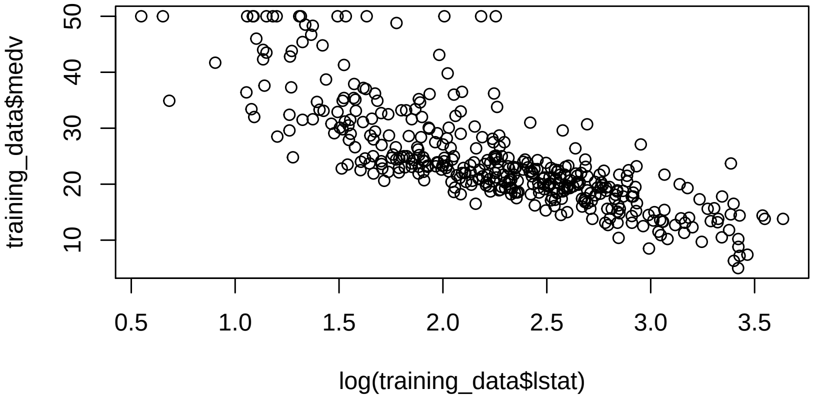
But before we run our model, let's visually check if the relationship between x and y is linear.

```
# Scatterplot of lstat vs. medv
plot(training_data$lstat, training_data$medv)
```



According to the plot, we see that the relationship is not linear. Let's try a transformation of our explanatory variable `lstat`.

```
# Scatterplot of log(lstat) vs. medv
plot(log(training_data$lstat), training_data$medv)
```



Look at the plot, it is more linear, so we can proceed and perform `lm()`:

STEP 3: Run the linear regression model

```
model = lm(medv ~ log(lstat), data = training_data)
model
#ans>
#ans> Call:
#ans> lm(formula = medv ~ log(lstat), data = training_data)
#ans>
#ans> Coefficients:
```

```
#ans> (Intercept)   log(lstat)
#ans>        51.8        -12.2
```

Notice that basic information when we print `model`. This only give us the slope (-12.2) and the intercept (51.8) of the linear model. Note that here we are looking at `log(lstat)` and not `lstat` anymore. So for every one unit increase in `lstat`, the median value of the house will decrease by $e^{12.2}$. For more detailed information, we can use the `summary()` function:

```
summary(model)
#ans>
#ans> Call:
#ans> lm(formula = medv ~ log(lstat), data = training_data)
#ans>
#ans> Residuals:
#ans>     Min      1Q  Median      3Q     Max
#ans> -11.385  -3.908  -0.779   2.245  25.728
#ans>
#ans> Coefficients:
#ans>             Estimate Std. Error t value Pr(>|t|)
#ans> (Intercept)   51.783      1.097    47.2   <2e-16 ***
#ans> log(lstat)   -12.203      0.472   -25.9   <2e-16 ***
#ans> ---
#ans> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#ans>
#ans> Residual standard error: 5.6 on 398 degrees of freedom
#ans> Multiple R-squared:  0.627,   Adjusted R-squared:  0.626
#ans> F-statistic:  669 on 1 and 398 DF,  p-value: <2e-16
```

Now, we have access to p-values and standard errors for the coefficients, as well as the R^2.

- The output states that the slope is statistically significant and different from 0 and with a t-value$= -25.9$ (p-value < 0.05), which means that there is a significant relationship between the percentage of households with low socioeconomic income and the median house value.
- This relationship is negative. That is as the percantage of household with low socioeconomic income increases, the median house value decreases.
- Looking at R^2, we can deduce that 62.7% of the model variation is being explained by the predictor `log(lstat)`. This is probably low, but indeed it would increase if we had more independent (explanatory) variables. We can use the `names()` function to see what other pieces of information are stored in our linear model (`model`).

```
names(model)
#ans>  [1] "coefficients"  "residuals"     "effects"       "rank"
#ans>  [5] "fitted.values" "assign"        "qr"            "df.residual"
#ans>  [9] "xlevels"       "call"          "terms"         "model"
```

```
model$coefficients
#ans> (Intercept)  log(lstat)
#ans>        51.8       -12.2
```

To obtain the confidence intervel for the linear model (`model`), we can use the `confint()` function:

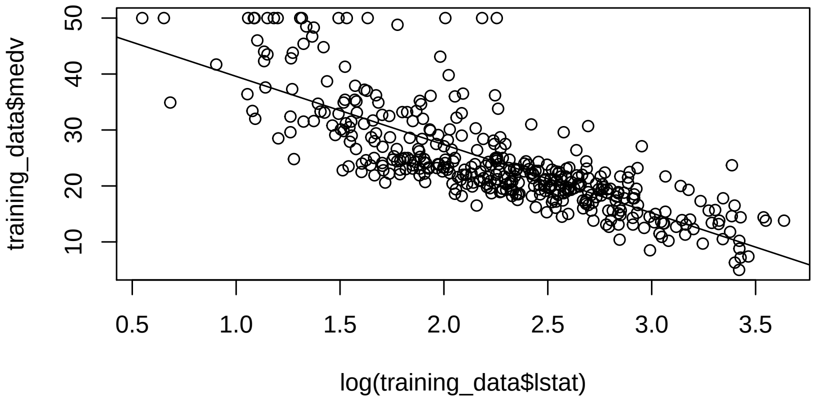
```
confint(model, level = 0.95)
#ans>             2.5 % 97.5 %
#ans> (Intercept)  49.6   53.9
#ans> log(lstat)  -13.1  -11.3
```

So, a 95% confidence interval for the slope of `log(lstat)` is $(-13.13, -11.28)$. Notice that this confidence interval gives us the same result as the hypothesis test performed earlier, by stating that we are 95% confident that the slope of `lstat` is not zero (in fact it is less than zero, which means that the relationship is negative.)

STEP 4: Plot the regression model

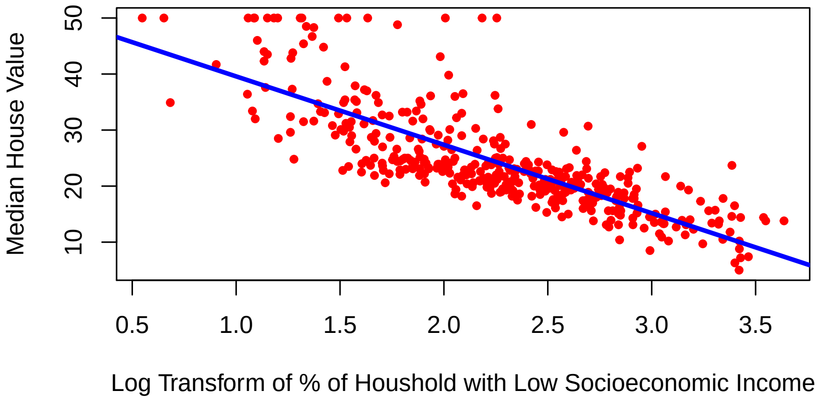
Now, let's plot our regression line on top of our data.

```
# Scatterplot of lstat vs. medv
plot(log(training_data$lstat), training_data$medv)

# Add the regression line to the existing scatterplot
abline(model)
```



Let's play with the look of the plot, and makes it perttier!

```
# Scatterplot of lstat vs. medv
plot(log(training_data$lstat), training_data$medv,
xlab = "Log Transform of % of Houshold with Low Socioeconomic Income",
ylab = "Median House Value",
col = "red",
pch = 20)

# Make the line color blue, and the line's width =3 (play with the width!)
abline(model, col = "blue", lwd =3)
```



STEP 5: Assess the model

Final thing we will do is to predict using our fitted model. We can use the `predict()` function for this purpose:

```
# Predict what is the median value of the house with lstat= 5%
predict(model, data.frame(lstat = c(5)))
#ans>    1
#ans> 32.1
```

```
# Predict what is the median values of houses with lstat= 5%, 10%, and 15%
predict(model, data.frame(lstat = c(5,10,15), interval = "prediction"))
#ans>    1    2    3
#ans> 32.1 23.7 18.7
```

Now let's assess our model, by computing th mean squared error (MSE). To assess the model we created, then we will be using the test data!

```
# Save the testing median values for houses (testing y) in y
y = testing_data$medv

# Compute the predicted value for this y (y hat)
y_hat = predict(model, data.frame(lstat = testing_data$lstat))

# Now we have both y and y_hat for our testing data.
# let's find the mean square error
error = y-y_hat
error_squared = error^2
MSE = mean(error_squared)
MSE
#ans> [1] 17.7
```

Chapter 2

Multiple Linear Regression

Simple linear regression is a useful approach for predicting a response on the basis of a single predictor variable. However, in practice we often have more than one predictor. In the previous chapter, we took for example the prediction of housing prices considering we had the size of each house. We had a single feature X, the size of the house. But now imagine if we had not only the size of the house as a feature but we also knew the number of bedrooms, the number of flours and the age of the house in years. It seems like this would give us a lot more information with which to predict the price.

2.1 The Model

In general, suppose that we have p distinct predictors. Then the multiple linear regression model takes the form

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon$$

where X_j represents the jth predictor and β_j quantifies the association between that variable and the response. We interpret β_j as the average effect on Y of a one unit increase in X_j, *holding all other predictors fixed.*

In matrix terms, supposing we have n observations and p variables, we need to define the following matrices:

$$\mathbf{Y}_{n\times 1} = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix} \qquad \mathbf{X}_{n\times(p+1)} = \begin{pmatrix} 1 & X_{11} & X_{12} & \dots & X_{1p} \\ 1 & X_{21} & X_{22} & \dots & X_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_{n1} & X_{n2} & \dots & X_{np} \end{pmatrix} \tag{2.1}$$

$$\beta_{(p+1)\times 1} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix} \qquad \epsilon_{n\times 1} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix} \tag{2.2}$$

In matrix terms, the general linear regression model is

$$\mathbf{Y}_{n\times 1} = \mathbf{X}_{n\times(p+1)} \beta_{(p+1)\times 1} + \epsilon_{n\times 1}$$

where,

- $\mathbf{Y}$ is a vector of responses.
- β is a vector of parameters.
- $\mathbf{X}$ is a matrix of constants.
- ϵ is a vector of independent *normal* (Gaussian) random variables.

2.2 Estimating the Regression Coefficients

As was the case in the simple linear regression setting, the regression coefficients $\beta_0, \beta_1, \ldots, \beta_p$ are unknown, and must be estimated. Given estimates $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$, we can make predictions using the formula

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \ldots, \hat{\beta}_p x_p$$

We choose $\beta_0, \beta_1, \ldots, \beta_p$ to minimize the sum of squared residuals

$$\begin{aligned} RSS &= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^{n} (y_1 - \hat{\beta}_0 - \hat{\beta}_1 \hat{x}_{i1} - \hat{\beta}_2 \hat{x}_{i2} - \ldots - \hat{\beta}_p \hat{x}_{ip})^2 \end{aligned}$$

The values $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$ that minimize the RSS are the multiple least squares regression coefficient estimates, they are calculated using this formula (in matrix terms):

$$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$$

Note 1:

> It is a remarkable property of matrix algebra that the results for the general linear regression model in matrix notation appear exactly as those for the simple linear regression model. Only the degrees of freedom and other constants related to the number of X variables and the dimensions of some matrices are different.

Note 2:

> If $\mathbf{X}^T \mathbf{X}$ is noninvertible, the common causes might be having:
>
> – Redundant features, where two features are very closely related (i.e. they are linearly dependent)
> – Too many features (e.g. $p \geq n$). In this case, we delete some features or we use “regularization” (to be, maybe, explained in a later lesson).

2.3 Some important questions

When we perform multiple linear regression, we usually are interested in answering a few important questions.

1. Is at least one of the predictors $X_1, X_2, \ldots, X_p$ useful in predicting the response?
2. Do all the predictors help to explain Y, or is only a subset of the predictors useful?
3. How well does the model fit the data?

4. Given a set of predictor values, what response value should we predict, and how accurate is our prediction?

Relationship Between the Response and Predictors?

F-Statistic

Recall that in the simple linear regression setting, in order to determine whether there is a relationship between the response and the predictor we can simply check whether $\beta_1 = 0$. In the multiple regression setting with p predictors, we need to ask whether all of the regression coefficients are zero, i.e. whether $\beta_1 = \beta_2 = \ldots = \beta_p = 0$. As in the simple linear regression setting, we use a hypothesis test to answer this question. We test the null hypothesis,

$$H_0 : \beta_1 = \beta_2 = \ldots = \beta_p = 0$$

versus the alternative hypothesis

$$H_1 : \text{at least one}\, \beta_j \text{ is non-zero}$$

This hypothesis test is performed by computing the F-statistic (*Fisher*):

$$F = \frac{(\text{TSS} - \text{RSS})/p}{\text{RSS}/(n-p-1)} \sim F_{p,n-p-1}$$

where, as with simple linear regression, $\text{TSS} = \sum(y_i - \bar{y})^2$ and $\text{RSS} = \sum(y_i - \hat{y}_i)^2$.

When the F-statistic value is close to 1, then H_0 is true, which means there is no relationship between the response and predictors. On the other hand, if H_1 is true, so we expect F to be greater than 1.

So the question we ask here: *Is the whole regression explaining anything at all?* The answer comes from the F-test in the ANOVA (ANalysis Of VAriance) table. This is what we get in an ANOVA table:

Source	df	SS	MS	F	p-value
Factor (Explained)	$p-1$	SST	SST/$(k-1)$	MST/MSE	p-value
Error (Unexplained)	$n-p$	SSE	SSE/$(n-k)$		
Total	$n-1$	SS			

The ANOVA table has many pieces of information. What we care about is the F Ratio and the corresponding p-value. We compare the F Ratio with $F_{(p-1,n-p)}$ and a corresponding α value (error).

p-values

The p-values provide information about whether each individual predictor is related to the response, after adjusting for the other predictors. Let's look at the following table we obtain in general using a statistical software for example

	Coefficient	Std. error	t-statistic	p-value
Constant	2.939	0.3119	9.42	<0.0001
X_1	0.046	0.0014	32.81	<0.0001
X_2	0.189	0.0086	21.89	<0.0001
X_3	-0.001	0.0059	-0.18	0.8599

In this tablewe the following model

$$Y = 2.939 + 0.046X_1 + 0.189X_2 - 0.001X_3$$

Note that for each individual predictor a *t*-statistic and a p-value were reported. These p-values indicate that X_1 and X_2 are related to Y, but that there is no evidence that X_3 is associated with Y, in the presence of these two.

Deciding on Important Variables

The most direct approach is called *all subsets* or *best subsets* regression: we compute the least squares fit for all possible subsets and then choose between them based on some criterion that balances training error with model size.

However we often can't examine all possible models, since they are 2^p of them; for example when $p = 40$ there are over a billion models! Instead we need an automated approach that searches through a subset of them. Here are two commonly use approaches:

Forward selection:

- Begin with the *null model* — a model that contains an intercept (constant) but no predictors.
- Fit p simple linear regressions and add to the null model the variable that results in the lowest RSS.
- Add to that model the variable that results in the lowest RSS amongst all two-variable models.
- Continue until some stopping rule is satisfied, for example when all remaining variables have a p-value above some threshold.

Backward selection:

- Start with all variables in the model.
- Remove the variable with the largest p-value — that is, the variable that is the least statistically significant.
- The new $(p1)$-variable model is fit, and the variable with the largest p-value is removed.
- Continue until a stopping rule is reached. For instance, we may stop when all remaining variables have a significant p-value defined by some significance threshold.

> There are more systematic criteria for choosing an "optimal" member in the path of models produced by forward or backward stepwise selection. These include *Mallow's C_p* , *Akaike information criterion (AIC)*, *Bayesian information criterion (BIC)*, *adjusted R^2* and *Cross-validation (CV).*

Model Fit

Two of the most common numerical measures of model fit are the RSE and R^2, the fraction of variance explained. These quantities are computed and interpreted in the same fashion as for simple linear regression. Recall that in simple regression, R^2 is the square of the correlation of the response and the variable. In multiple linear regression, it turns out that it equals $Cor(Y, \hat{Y})^2$, the square of the correlation between the response and the fitted linear model; in fact one property of the fitted linear model is that it maximizes this correlation among all possible linear models. An R^2 value close to 1 indicates that the model explains a large portion of the variance in the response variable.

In general RSE is defined as

$$\text{RSE} = \sqrt{\frac{1}{n-p-1}\text{RSS}}$$

2.3.1 Other Considerations in Regression Model

Qualitative Predictors

- If we have a categorial (qualitative) variable (feature), how do we fit into a regression equation?
- For example, if X_1 is the gender (male or female).
- We can code, for example, male = 0 and female = 1.
- Suppose X_2 is a quantitative variable, the regression equation becomes:

$$Y_i \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 = \begin{cases} \beta_0 + \beta_2 X_2 & \text{if male} \\ \beta_0 + \beta_1 X_1 + \beta_2 X_2 & \text{if female} \end{cases}$$

- Another possible coding scheme is to let male = -1 and female = 1, the regression equation is then:

$$Y_i \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 = \begin{cases} \beta_0 - \beta_1 X_1 + \beta_2 X_2 & \text{if male} \\ \beta_0 + \beta_1 X_1 + \beta_2 X_2 & \text{if female} \end{cases}$$

Interaction Terms

- When the effect on Y of increasing X_1 depends on another X_2.
- We may in this case try the model

$$Y_i = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2$$

- $X_1 X_2$ is the Interaction term.

PW 2

2.4 Reporting

Markdown

Markdown is a lightweight markup language with plain text formatting syntax designed so that it can be converted to HTML and many other formats (pdf, docx, etc..).

Click here to see an example of a markdown (`.md`) syntaxes and the result in HTML. The markdown syntaxes are on right and their HTML result is on left. You can modify the source text to see the result.

Extra: There is some markdown online editors you can use, like dillinger.io/. See the Markdown source file and the HTML preview. Play with the source text to see the result in the preview.

R Markdown

R Markdown is a variant of Markdown that has embedded `R` code chunks, to be used with the `knitr` package to make it **easy to create reproducible web-based reports**.

First, in `Rstudio` create a new `R Markdown` file. A default template will be opened. There is some `R` code in `R chunks`. Clink on `knit`, save your file and see the produced output. The output is a html report containing the results of the `R` codes. If your file is named `report.Rmd`, your report is named `report.html`.

- Make sure to have the latest version of `Rstudio`.
- If you have problems creating a `R Markdown` file (problem in installing packages, etc..) close your `Rstudio` and reopen it with administrative tools and retry.
- If it doesn't work and in order to not lose more time, write your script in an `.R` file with your comments and submit it.

- Be ready to submit your report (your `.html` file) at the end of each class.
- You report must be named:

`YouLastName_YourFirstName_WeekNumber.html`

You can find all the informations about R Markdown on this site: rmarkdown.rstudio.com.

You may also find the following resources helpful:

- The R Markdown Reference Guide
- The R Markdown Cheatsheet

The report to be submitted

In `Rstudio`, start by creating a R Markdown file. When you create it a default template will be opened with the following first lines:

```
---
title: "Untitled"
output: html_document
---
```

These lines are the YAML header in which you choose the settings of your report (title, author, date, appearance, etc..)

For your submitted report, use the following YAML header:

```
---
title: "Week 2"
subtitle: "Multiple Linear Regression"
author: LastName FirstName
date: "2017-01-25 17:00:43"
output:
  html_document:
    toc: true
    toc_depth: 2
    theme: flatly
---
```

In the core of your report:

- Put every exercise in a section, name the section `Exercise i` (i is the exercise's number).
- Paste the exercise content.
- Write the code of the exercise in R chunks.
- Run the chunk to make sure it works.
- If there is a need, explain the results.
- Click on `knit`

2.5 Multiple Linear Regression

Submit a report following the instructions above. In the first exercise you must continue the analysis of the Boston data set.

The exercises

Continue what we started last week with the Boston dataset which is part of the `MASS` package. Recall that this dataset records the median value of houses for 506 neighborhoods around Boston.

1. Load the Boston dataset from `MASS` package.

2. Split the dataset into traning set and testing set. (keep all the variables of the Boston data set)

3. Check if there is a linear relationship between the variables `medv` and `age`. (use `cor()` function).

4. Plot `medv` in function of `age`.

5. Train a regression model using both `lstat` and `age` as predictors of median house value. (Remember that we transformed `lstat`, use the same transformation here). What is the obtained model?

6. Print the summary of the obtained regression model.
7. Are the predictors significant?
8. Is the model as a whole significant ?
9. Train a new model using all the variables of the dataset. (We can use `.` as a short cut instead of writing down all the variables names)
10. When using all the variables as predictors, we didn't transform `lstat`. Re train the model using `log(lstat)` instead of `lstat`.
11. Did R^2 improve ?
12. To see if there is correlated variables print the correlation matrix using the `cor()` function (round the correlations with 2 digits).
13. Visualize the correlations using the `corrplot` package. To do so, install the `corrplot` package, load it, then use the function `corrplot.mixed()`. See this link for examples and to understand how to use it.
14. What is the correlation between `tax` and `rad`?
15. Run the model again without `rad`. What happens to the R^2 ? and for the F-statistic?

> Of course R^2 should go a little lower because we deleted one of the variables. But check for the model significance (F-statistic) gets higher, which means the p-values gets lower and thus the model is more significant without `rad`.

16. Calculate the mean squared error (MSE) for the last model.

ANOVA

17. In the Boston data set there is a categorical variable `chas` which corresponds to Charles River (= 1 if a suburb bounds the river; 0 otherwise). How many of the suburbs in this data set bound the Charles river?
18. Create Boxplots of the median value of houses with respect to the variable `chas`. Do we observe some difference between the median value of houses with respect to the neighborhood to Charles River?
19. Next we will apply an analysis of variances (ANOVA) in order to test if there is a significant difference of means between two groups i and j (Consider group i is the suburbs bounding the river and j the suburbs which not). The hypotheses are

$$H_0 : \mu_i = \mu_j$$

$$H_1 : \mu_i \neq \mu_j$$

Where μ_i is the mean of `medv` in group i.

Calculate μ_i and μ_j (in one line using the function `aggregate()`).

20. Apply an ANOVA test of `medv` whith respect to `chas` (use the function `aov()`). Print the result and the summary of it. what do you conclude ?

Part III

Classification

Chapter 3

Logistic Regression

3.1 Introduction

In the previous chapters we discussed the linear regression model, which assumes that the response variable Y is quantitative. But in many situations, the response variable is instead qualitative (categorical). For example, eye color is qualitative, taking on values blue, brown, or green.

The process for predicting qualitative responses is known as ***classification***.

Given a feature vector X and a qualitative response Y taking values in the set $\mathcal{C}$, the classification task is to build a function $C(X)$ that takes as input the feature vector X and predicts its value for Y; i.e. $C(X) \in \mathcal{C}$. We are often more interested in estimating the probabilities that X belongs to each category in $\mathcal{C}$.

If c is a category ($c \in \mathcal{C}$), by the probability that X belongs to c we mean $p(X \in c) = \mathbb{P}(Y = c|X)$.

In the binomial or binary logistic regression, the outcome can have only two possible types of values (e.g. "Yes" or "No", "Success" or "Failure"). Multinomial logistic refers to cases where the outcome can have three or more possible types of values (e.g., "good" vs. "very good" vs. "best"). Generally outcome is coded as "0" and "1" in binary logistic regression.

3.2 Logistic Regression

Consider a data set where the response falls into one of two categories, Yes or No. Rather than modeling the response Y directly, logistic regression models the *probability* that Y belongs to a particular category.

3.2.1 The Logistic Model

Let us suppose the response has two categories and we use the generic 0/1 coding for the response. How should we model the relationship between $p(X) = \mathbb{P}(Y = 1|X)$ and X?

The simplest situation is when Y is *binary*: it can only take two values, codified for convenience as 0 (success) and 1 (failure).

More formally, a binary variable is known as a *Bernoulli variable*, which is the simplest non-trivial random variable. We say that $Y \sim \text{Ber}(p)$, $0 \leq p \leq 1$, if

$$Y = \begin{cases} 1, & \text{with probability } p, \\ 0, & \text{with probability } 1 - p, \end{cases}$$

or, equivalently, if $\mathbb{P}[Y = 1] = p$ and $\mathbb{P}[Y = 0] = 1 - p$, which can be written compactly as

$$\mathbb{P}[Y = y] = p^y(1 - p)^{1-y}, \quad y = 0, 1.$$

Recall that a *binomial variable with size* n *and probability* p, $\mathrm{Bi}(n, p)$, was obtained by adding n independent $\mathrm{Ber}(p)$ (so $\mathrm{Ber}(p)$ is the same as $\mathrm{Bi}(1, p)$).

A Bernoulli variable Y is completely determined by p. **So its mean and variance**:

- $\mathbb{E}[Y] = p \times 1 + (1 - p) \times 0 = p$
- $\mathbb{V}\mathrm{ar}[Y] = p(1 - p)$.

In particular, recall that $\mathbb{P}[Y = 1] = \mathbb{E}[Y] = p$.

Assume then that Y is a binary/Bernoulli variable and that X are predictors associated to them (no particular assumptions on them). The purpose in *logistic regression* is to estimate

$$p(x) = \mathbb{P}[Y = 1|X = x] = \mathbb{E}[Y|X = x],$$

this is, how the probability of $Y = 1$ is changing according to particular values, denoted by x, of the random variables X.

Why not linear regression? A tempting possibility is to consider the model

$$p(x) = \beta_0 + \beta_1 x.$$

However, such a model will run into problems inevitably: negative probabilities and probabilities larger than one ($p(x) < 0$ for some values of X and $p(X) > 1$ for others). To avoid this problem, the solution is to consider a function to encapsulate the value of $z = \beta_0 + \beta_1 x$, in $\mathbb{R}$, and map it to $[0, 1]$. There are several alternatives to do so, based on distribution functions $F : \mathbb{R} \longrightarrow [0, 1]$ that deliver $y = F(z) \in [0, 1]$. Many functions meet this description. In logistic regression, we use the *logistic function*,

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

- No matter what values β_0, β_1 or X take, $p(X)$ will have values between 0 and 1.
- The logistic function will always produce an *S-shaped* curve.
- The logistic *distribution* function is:

$$F(z) = \mathrm{logistic}(z) = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}}.$$

After a bit of manipulation of the previous equation, we find that

$$\frac{p(X)}{1 - p(X)} = e^{\beta_0 + \beta_1 X}$$

The quantity $p(X)/[1p(X)]$ is called the *odds*, and can take on any value between 0 and ∞.

By taking the logarithm of both sides of the equation, we arrive at

$$\log(\frac{p(X)}{1 - p(X)}) = \beta_0 + \beta_1 X$$

The left-hand side is called the *log-odds* or *logit*. We see that the logistic regression model has a logit that is linear in X.

3.2.2 Estimating the Regression Coefficients

We estimate β_0 and β_1 using the *Maximum Likelihood Estimation* method (MLE). The basic intuition behind using maximum likelihood to fit a logistic regression model is as follows: we seek estimates for β_0 and β_1 such that the predicted probability $\hat{p}(x_i)$ of the response for each individual, corresponds as closely as possible to the individual's observed response status (recall that the response Y is categorical). The *likelihood function* is

$$l(\beta_0, \beta_1) = \prod_{i=1}^{n} p(x_i)^{Y_i}(1 - p(x_i))^{1-Y_i}.$$

This likelihood is **the probability of the data based on the model**. It gives the probability of the observed zeros and ones in the data. The estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are chosen to *maximize* this likelihood function. The interpretation of the likelihood function is the following:

- $\prod_{i=1}^{n}$ appears because the sample elements are assumed to be independent and we are computing the probability of observing the whole sample $(x_1, y_1), \dots, (x_n, y_n)$. This probability is equal to the product of the *probabilities of observing each* (x_i, y_i).
- $p(x_i)^{Y_i}(1 - p(x_i))^{1-Y_i}$ is the probability of observing (x_i, Y_i).

In the linear regression setting, the least squares approach is a special case of maximum likelihood.

We will not give mathematical details about the maximum likelihood and how to estimate the parameters. We will use R to fit the logistic regression models (using `glm` function).

Click here to see how the log-likelihood changes with respect to the values for (β_0, β_1) in three data patterns. The logistic regression fit and its dependence on β_0 (horizontal displacement) and β_1 (steepness of the curve). Recall the effect of the sign of β_1 in the curve: if positive, the logistic curve has an *s* form; if negative, the form is a reflected *s*.’

3.2.3 Prediction

Example

	Coefficient	Std. error	Z-statistic	p-value
Constant	-10.6513	0.3612	-29.5	<0.0001
X	0.0055	0.0002	24.9	<0.0001

In this example, $\hat{\beta}_0 = -10.6513$ and $\hat{\beta}_1 = 0.0055$. It produces the blue curve that separates that data in the following figure,

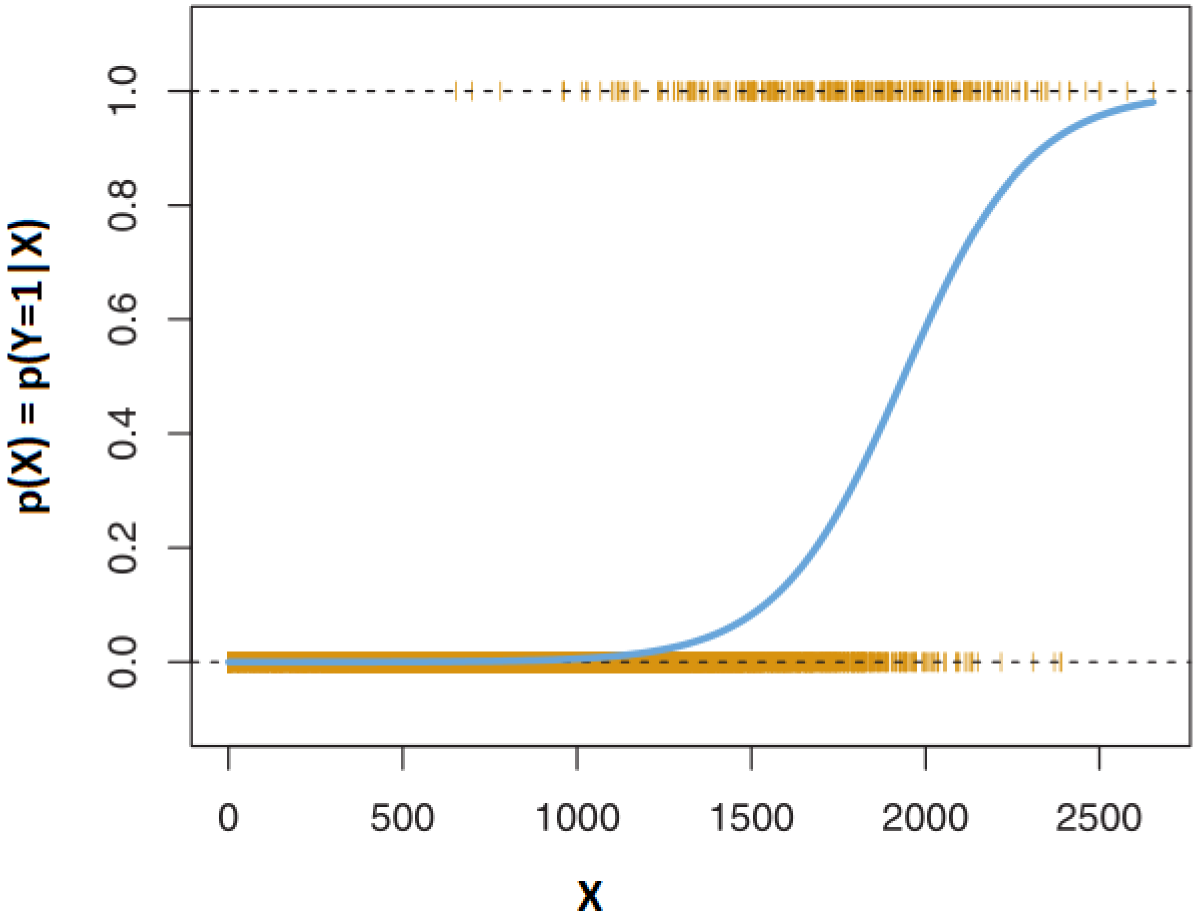


Figure 3.1:

As for prediction, we use the model built with the estimated parameters to predict probabilities. For example, If $X = 1000$,

$$\hat{p}(X) = \frac{e^{\hat{\beta_0}+\hat{\beta_1}X}}{1+e^{\hat{\beta_0}+\hat{\beta_1}X}} = \frac{e^{-10.6513+0.0055\times1000}}{1+e^{-10.6513+0.0055\times1000}} = 0.006$$

If $X = 2000$,

$$\hat{p}(X) = \frac{e^{\hat{\beta_0}+\hat{\beta_1}X}}{1+e^{\hat{\beta_0}+\hat{\beta_1}X}} = \frac{e^{-10.6513+0.0055\times2000}}{1+e^{-10.6513+0.0055\times2000}} = 0.586$$

3.3 Multiple Logistic Regression

We now consider the problem of predicting a binary response using multiple predictors. By analogy with the extension from simple to multiple linear regression in the previous chapters, we can generalize the simple logistic regression equation as follows:

$$\log(\frac{p(X)}{1-p(X)}) = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p$$

where $X = (X_1, \ldots, X_p)$ are p predictors. The equation above can be rewritten as

$$p(X) = \frac{e^{\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p}}$$

Just as in the simple logistic regression we use the maximum likelihood method to estimate $\beta_0, \beta_1, \ldots, \beta_p$.

PW 3

Report template

Create a new RMarkdown file and replace the YAML header with these lines:

```
---
title: "Week 3"
subtitle: "Logistic Regression"
author: LastName FirstName
date: "23/01/2017"
output:
  html_document:
    toc: true
    toc_depth: 2
    toc_float: true
    theme: united
    highlight: zenburn
---
```

Name the file `LastName_FirstName_Week3` and submit the html output.

Social Networks Ads

In this PW we are going to analyse the `Social_Network_Ads` dataset. This dataset contains informations of users of a social network. The social network has several business clients and its business clients put ads on the social network for marketing compaigns purposes. For this dataset, a company has put ads for one of its new products and the social network gathered some informations about wich users responded positively to the ad by buying the product and those who responded negatively by not buying the product.

1. Download the `Social_Network_Ads` dataset from here and import it into `R`.

2. Describe the dataset (you can use `str()` and `summary()` functions).

We will consider the variables `Age` and `EstimatedSalary` as input variables (features) to see the correlations between them and the decision of the user to buy (or not) the product.

3. Now we are going to split the dataset into training set and test set. Last week we did it manually. From now on we will split it randomly with this code,

```r
library(caTools) # install it first in the console
set.seed(123)
# we use this function with the same number
# to randomly generate the same values
split = sample.split(dataset$Purchased, SplitRatio = 0.75)
```

```r
# here we chose the SplitRatio to 75% of the dataset,
# and 25% for the test set.
training_set = subset(dataset, split == TRUE)
# we use subset to split the dataset
test_set = subset(dataset, split == FALSE)
```

4. Scale the input variables in both training set and test set.

First let us fit a simple logistic regression model of `Purchased` in function of `Age`. We do it with this line of code,

```r
classifier <- glm(Purchased ~ Age , family = binomial, data=training_set)
# glm goes for generalized linear model
```

5. In the argument `family` of the function `glm` we chose `binomial`. Why ?

6. What is the equation of the obtained model in the question **4** ?

7. Is the feature `Age` significant?

The **AIC** is the **Akaike Information Criterion**. You will use this while comparing multiple models. The model with lower value of AIC is better. Suppose that we have a statistical model of some data. Let $\hat{L}$ be the maximum value of the likelihood function for the model; let k be the number of estimated parameters in the model. Then the AIC value of the model is the following.

$$\text{AIC} = 2k - 2\ln(\hat{L})$$

where

- $\hat{L}$ = the maximized value of the likelihood function of the model M, i.e. $\hat{L} = p(x|\hat{\beta}, M)$, where $\hat{\beta}$ are the parameter values that maximize the likelihood function.
- x = the observed data.
- k = the number of free parameters to be estimated. If the model under consideration is a linear regression, k is the number of regressors, including the intercept.

8. Plot `Purchased` in function of `Age` and add the curve of the obtained logistic regression model.

(**Hints**: First plot the point, then use the `curve()` function with oprtion `add=TRUE` to add the curve to the plot. The argument "type" of the function `predit()` must be "reponse")

You must obtain something like

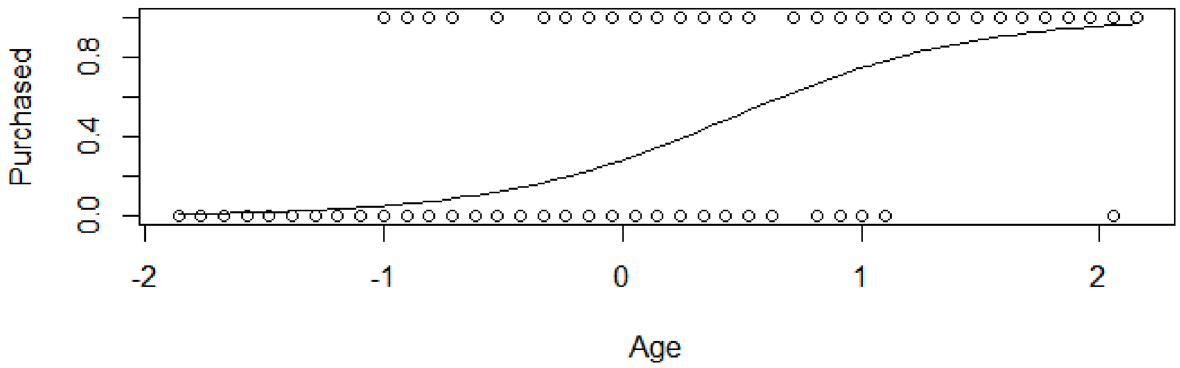


Figure 3.2:

Extra: A great R package for visualization if `ggplot2`. Take a look on this link for some examples. With this library we obtain the following plot for our model,

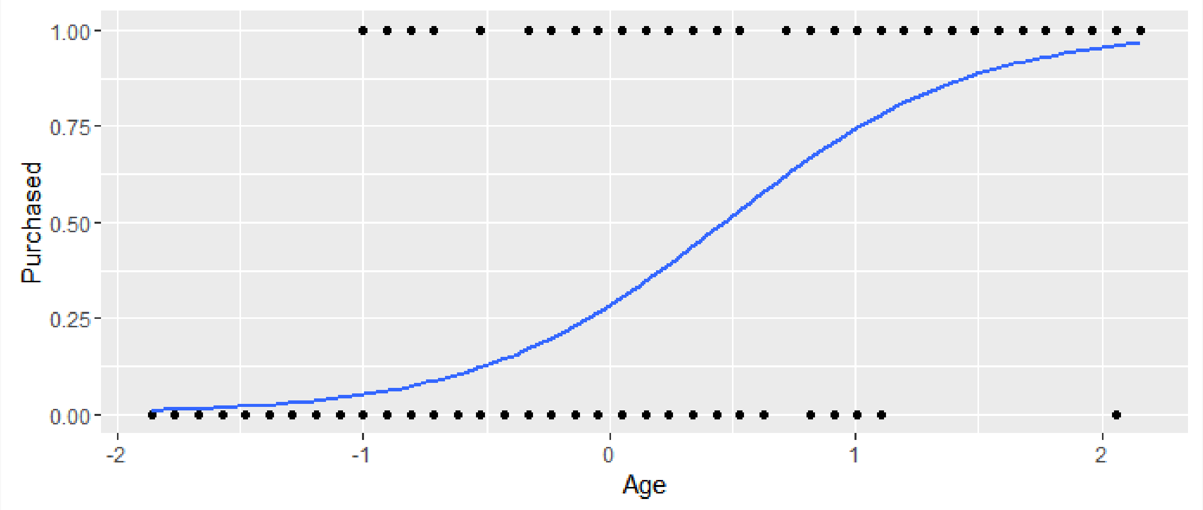


Figure 3.3:

I obtained the last figure with these lines of code,

```
library(ggplot2)
ggplot(training_set, aes(x=Age, y=Purchased)) +
  geom_point() +
  stat_smooth(method="glm", method.args=list(family="binomial"), se=FALSE)
```

9. Now let us take another feature into account in the model. Fit a logistic regression model of purchasing the product in function of the age of the user and its salary.

10. Are the predictors significant? Did the model get better by adding t he estimated salary?

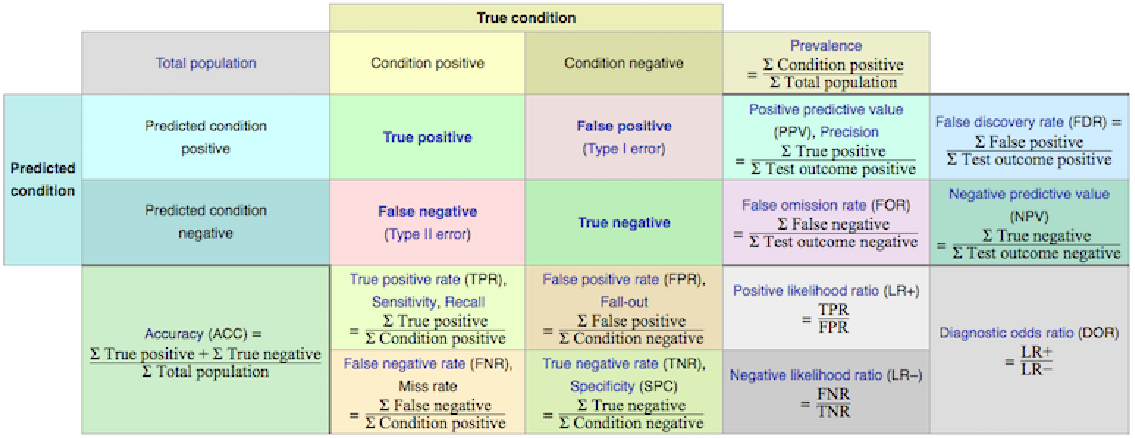
11. On the test set, predict the probability of purchasing the product by the users using the obtained model.

12. Take a look on your predicted values for the variable `Purchased`. We predicted the probability that the user will purchase the product right? Now in order to compare your results with the real answers, transform the predicted values to 0 or 1 (1 if >0.5).

Hint: You can easily do it with the `ifelse()` function.

Now read the following about the evaluation of a classifier (of a classification model).

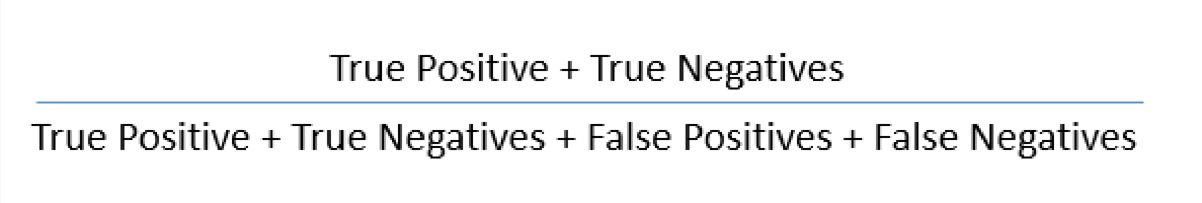
Confusion matrix: is a tabular representation of Actual vs Predicted values. This helps us to find the accuracy of the model. The different results from a binary classifier are true positives, true negatives, false positives, and false negatives. This is how the confusion matrix looks like:



		True condition			
	Total population	Condition positive	Condition negative	Prevalence $= \frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**	**False positive** (Type I error)	Positive predictive value (PPV), Precision $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Test outcome positive}}$	False discovery rate (FDR) $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Test outcome positive}}$
	Predicted condition negative	**False negative** (Type II error)	**True negative**	False omission rate (FOR) $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Test outcome negative}}$	Negative predictive value (NPV) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Test outcome negative}}$
	Accuracy (ACC) $= \frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	True positive rate (TPR), Sensitivity, Recall $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR−}}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	True negative rate (TNR), Specificity (SPC) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$	

(Image source: Wikipedia)

You can calculate the **accuracy** of your model with:



$$\frac{\text{True Positive + True Negatives}}{\text{True Positive + True Negatives + False Positives + False Negatives}}$$

Figure 3.4:

Accuracy is a key measure of performance, and is more specifically the rate at which the model is able to predict the correct value (classification or regression) for a given data point or observation. In other words, accuracy is the proportion of correct predictions out of all predictions made.

The other two metrics from the confusion matrix worth discussing are **precision** and **recall**. Precision (positive predictive value) is the ratio of true positives to the total amount of positive predictions made (i.e., true or false). Said another way, precision measures the proportion of accurate positive predictions out of all positive predictions made.

Recall on the other hand, or true positive rate, is the ratio of true positives to the total amount of actual positives, whether predicted correctly or not. So in other words, recall measures the proportion of accurate positive predictions out of all actual positive observations.

A metric that is associated with precision and recall is called the F-score (also called F1 score), which combines them mathematically, and somewhat like a weighted average, in order to produce a single measure of performance based on the simultaneous values of both. Its values range from 0 (worst) to 1 (best).

Another important concept to know about is the ***Receiver Operating Characteristic***, which when plotted, results in what's known as an ROC curve.

ROC Curve: An ROC curve is a two-dimensional plot of *sensitivity* (recall, or true positive rate) vs *specificity* (false positive rate). The area under the curve is referred to as the ***AUC***, and is a numeric metric used to represent the quality and performance of the classifier (model).

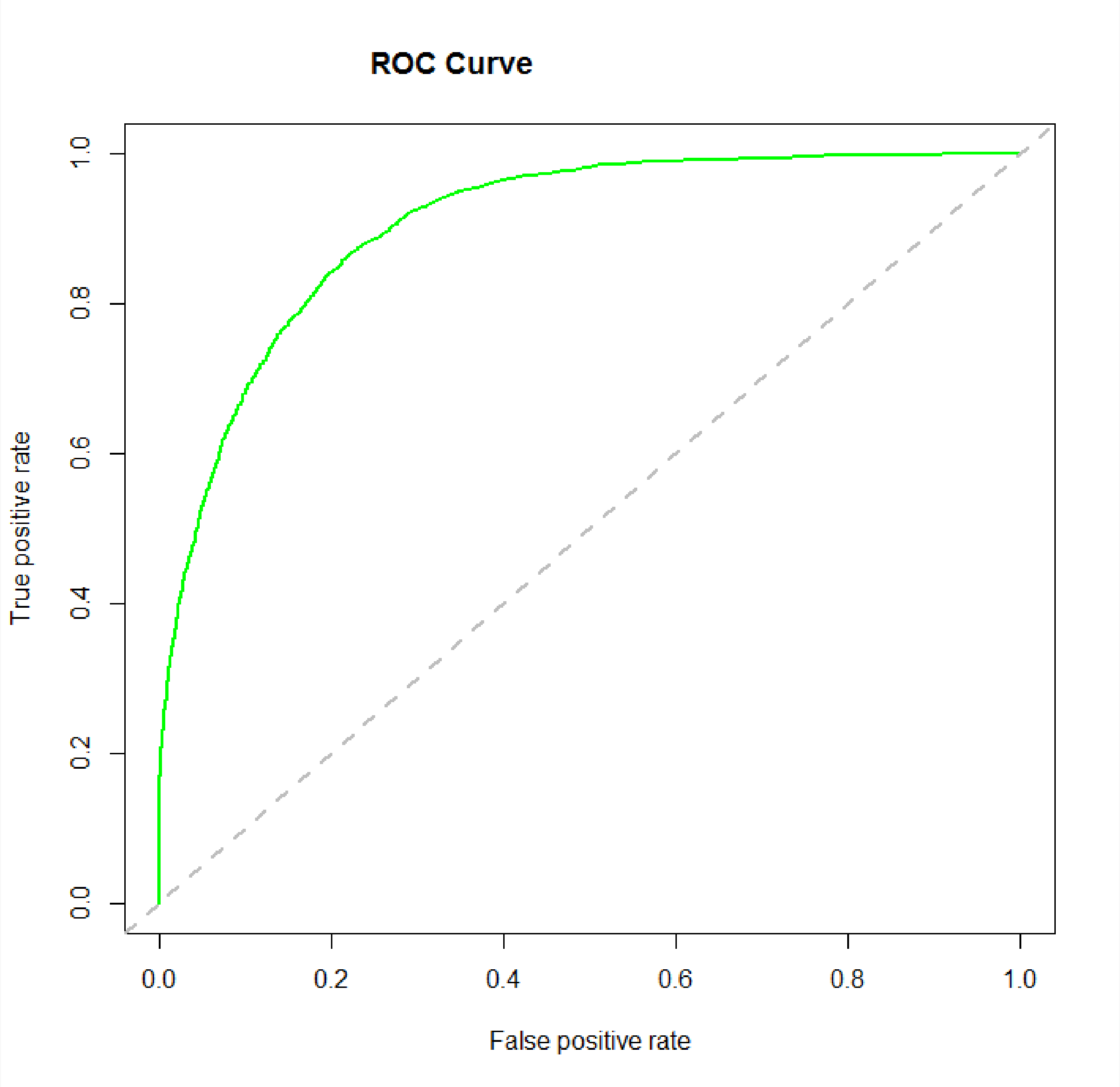


Figure 3.5:

An AUC of 0.5 is essentially the same as random guessing without a model, whereas an AUC of 1.0 is considered a perfect classifier. Generally, the higher the AUC value the better, and an AUC above 0.8 is considered quite good.

The higher the AUC value, the closer the curve gets to the upper left corner of the plot. One can easily see from the ROC curves then that the goal is to find and tune a model that maximizes the true positive rate, while simultaneously minimizing the false positive rate. Said another way, the goal as shown by the ROC curve is to correctly predict as many of the actual positives as possible, while also predicting as many of the actual negatives as possible, and therefore minimize errors (incorrect classifications) for both.

13. Now to evaluate the predictions, compute the confusion matrix. What do you obtain ?

(**Hint**: you can use the `table()` function).

14. Calculate the accuracy, specificity, sensitivity and the precision of the model.

15. Plot the ROC curve and calculate AUC value.

> **Hints**: to plot it, install the `ROCR` package. Load and use the functions:
>
> - `prediction()` to calculate the elements of the confusion matrix.
> - `performance()` to calculate the AUC.
> - `plot()` to plot the ROC curve, you can plot the performance calculated before.
> - `abline()` to plot a line of equation `y=x`.

16. Compare the AUC of the two models you fitted (one with only age and one with age and estimated salary) and plot their ROC curves in the same figure.

Chapter 4

Linear Discriminant Analysis

Discriminant analysis is a popular method for multiple-class classification. We will start first by the *Linear Discriminant Analysis (LDA).*

4.1 Introduction

As we saw in the previous chapter, Logistic regression involves directly modeling $\mathbb{P}(Y = k|X = x)$ using the *logistic function*, for the case of two response classes. In logistic regression, we model the conditional distribution of the response Y, given the predictor(s) X. We now consider an alternative and less direct approach to estimating these probabilities. In this alternative approach, ***we model the distribution*** of the predictors X ***separately*** in each of the response classes (i.e. given Y), and then use **Bayes' theorem** to flip these around into estimates for $\mathbb{P}(Y = k|X = x)$. When these distributions are assumed to be *Normal*, it turns out that the model is very similar in form to logistic regression.

Why not logistic regression? Why do we need another method, when we have logistic regression? There are several reasons:

- When the classes are well-separated, the parameter estimates for the logistic regression model are surprisingly unstable. Linear discriminant analysis does not suffer from this problem.
- If n is small and the distribution of the predictors X is approximately normal in each of the classes, the linear discriminant model is again more stable than the logistic regression model.
- Linear discriminant analysis is popular when we have *more than two response classes.*

4.2 Bayes' Theorem

Bayes' theorem is stated mathematically as the following equation:

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

where A and B are events and $P(B) \neq 0$.

- $P(A|B)$, a conditional probability, is the probability of observing event A given that B is true. It is called the ***posterior*** probability.
- $P(A)$, is called the ***prior***, is the initial degree of belief in A.
- $P(B)$ is the ***likelihood***.

The posterior probability can be written in the memorable form as :

Posterior probability $\propto$ Likelihood $\times$ Prior probability.

Extended form:

Suppose we have a partition $\{A_i\}$ of the sample space, the even space is given or conceptualized in terms of $P(A_j)$ and $P(B|A_j)$. It is then useful to compute $P(B)$ using the law of total probability:

$$P(B) = \sum_j P(B|A_j)P(A_j)$$

$$\Rightarrow P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_j P(B|A_j)P(A_j)}$$

Bayes' Theorem for Classification:

Suppose that we wish to classify an observation into one of K classes, where $K \geq 2$. In other words, the qualitative response variable Y can take on K possible distinct and unordered values.

Let π_k represent the overall or *prior* probability that a randomly chosen observation comes from the k-th class; this is the probability that a given observation is associated with the k-th category of the response variable Y.

Let $f_k(X) \equiv P(X = x|Y = k)$ denote the *density function* of X for an observation that comes from the k-th class. In other words, $f_k(x)$ is relatively large if there is a high probability that an observation in the k-th class has $X \approx x$, and $f_k(x)$ is small if it is very unlikely that an observation in the k-th class has $X \approx x$. Then *Bayes' theorem* states that

$$P(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{c=1}^{K} \pi_c f_c(x)} \tag{4.1}$$

As we did in the last chapter, we will use the abbreviation $p_k(X) = P(Y = k|X)$.

The equation above stated by *Bayes' theorem* suggests that instead of directly computing $p_k(X)$ as we did in the logistic regression, we can simply plug in estimates of π_k and $f_k(X)$ into the equation. In general, estimating π_k is easy (the fraction of the training observations that belong to the k-th class). But estimating $f_k(X)$ tends to be more challenging.

> Recall that $p_k(x)$ is the *posterior* probability that an observation $X = x$ belongs to k-th class.
>
> If we can find a way to estimate $f_k(X)$, we can develop a classifier with the lowest possibe error rate out of all classifiers.

4.3 LDA for $p = 1$

Assume that $p = 1$, which mean we have only one predictor. We would like to obtain an estimate for $f_k(x)$ that we can plug into the Equation (4.1) in order to estimate $p_k(x)$. **We will then classify an observation to the class for which $p_k(x)$ is greatest**.

In order to estimate $f_k(x)$, we will first make some assumptions about its form.

Suppose we assume that $f_k(x)$ is *normal* (*Gaussian*). In the one-dimensional setting, the normal density take the form

$$f_k(x) = \frac{1}{\sigma_k\sqrt{2\pi}} \exp\big(-\frac{1}{2\sigma_k^2}(x-\mu_k)^2\big) \tag{4.2}$$

where μ_k and σ_k^2 are the mean and variance parameters for k-th class. Let us assume that $\sigma_1^2 = \ldots = \sigma_K^2 = \sigma$ (which means there is a shared variance term across all K classes). Plugging Eq. (4.2) into the Bayes formula in Eq. (4.1) we get,

$$p_k(x) = \frac{\pi_k \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2}}{\sum_{c=1}^{K} \pi_c \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_c}{\sigma}\right)^2}} \tag{4.3}$$

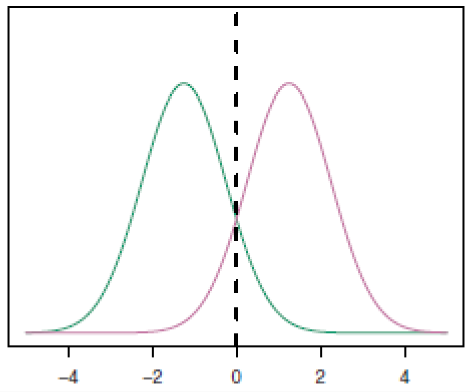
Note that π_k and π_c denote the prior probabilities. And π is the mathematical constant $\pi \approx 3.14159$.

To classify at the value $X = x$, we need to see which of the $p_k(x)$ is largest. Taking logs, and discarding terms that do not depend on k, we see that this is equivalent to assigning x to the class with the largest ***discriminant score***:

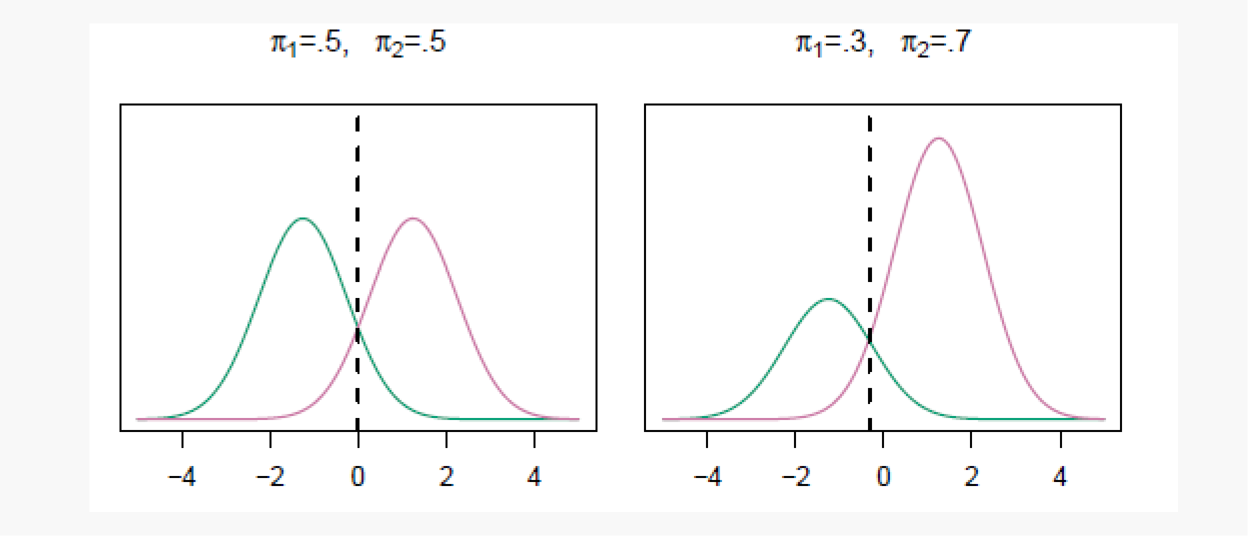
$$\delta_k(x) = x.\frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k) \tag{4.4}$$

Note that $\delta_k(x)$ is a *linear* function of x.

- The decision surfaces for a linear discriminant classifiers are defined by the linear equations $\delta_k(x) = \delta_c(x)$.
- Example: If $K = 2$ and $\pi_1 = \pi_2$, then the **desicion boundary** is at $x = \frac{\mu_1 + \mu 2}{2}$ (**Prove it!**).
- An example where $\mu_1 = -1.5$, $\mu_2 = 1.5$, $\mu_1 = \mu_2 = 0.5$ and $\sigma^2 = 1$ is shown in this following figure



- See this video to understand more about *decision boundary.*
- As we classify a new point according to which density is highest, when the priors are different we take them into account as well, and compare $\pi_k f_k(x)$. On the right of the following figure, we favor the pink class (remark that the decision boundary has shifted to the left).



4.4 Estimating the parameters

Typically we don't know these parameters; we just have the training data. In that case we simply estimate the parameters and plug them into the rule.

Let n the total number of training observations, and n_k the number of training observations in the k-th class. The following estimates are used:

$$\begin{aligned}\hat{\pi}_k &= \frac{n_k}{n}\\ \hat{\mu}_k &= \frac{1}{n_k}\sum_{i:y_i=k} x_i\\ \hat{\sigma}^2 &= \frac{1}{n-K}\sum_{k=1}^{K}\sum_{i:y_i=k}(x_i-\hat{\mu}_k)^2\\ &= \sum_{k=1}^{K}\frac{n_k-1}{n-K}.\hat{\sigma}_k^2\end{aligned}$$

where $\hat{\sigma}_k^2 = \frac{1}{n_k-1}\sum_{i:y_i=k}(x_i-\hat{\mu}_k)^2$ is the usual formula for the estimated variance in the -k-th class.

The linear discriminant analysis (LDA) classifier plugs these estimates in Eq. (4.4) and assigns an observation $X = x$ to the class for which

$$\hat{\delta}_k(x) = x.\frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k) \tag{4.5}$$

is largest.

The *discriminant functions* in Eq. (4.5) are linear functions of x.

> Recall that we assumed that the observations come from a normal distribution with a common variance σ^2.

4.5 LDA for $p > 1$

Let us now suppose that we have multiple predictors. We assume that $X = (X_1, X_2, \ldots, X_p)$ is drawn from *multivariate Gaussian* distribution (assuming they have a common covariance matrix, e.g. same variances as in the case of $p = 1$). The multivariate Gaussian distribution assumes that each individual predictor follows a one-dimensional normal distribution as in Eq. (4.2), with some correlation between each pair of predictors.

To indicate that a p-dimensional random variable X has a multivariate Gaussian distribution, we write $X \sim \mathcal{N}(\mu, \Sigma)$. Where

$$\mu = E(X) = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_p \end{pmatrix}$$

and,

$$\Sigma = Cov(X) = \begin{pmatrix} \sigma_1^2 & Cov[X_1, X_2] & \ldots & Cov[X_1, X_p] \\ Cov[X_2, X_1] & \sigma_2^2 & \ldots & Cov[X_2, X_p] \\ \vdots & \vdots & \ddots & \vdots \\ Cov[X_p, X_1] & Cov[X_p, X_2] & \ldots & \sigma_p^2 \end{pmatrix}$$

Σ is the $p \times p$ covariance matrix of X.

Formally, the multivariate Gaussian density is defined as

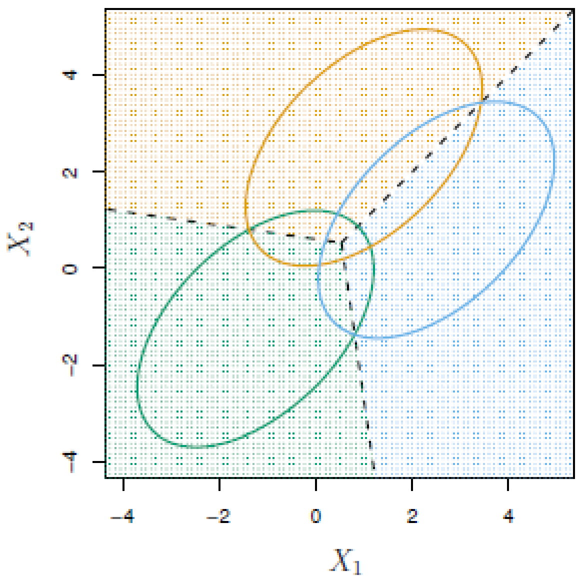
$$f(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\Big(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\Big)$$

Plugging the density function for the k-th class, $f_k(X = x)$, into Eq. (4.1) reveals that the Bayes classifier assigns an observation $X = x$ to the class for which

$$\delta_k(x) = x^T \Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T \Sigma^{-1}\mu_k + \log \pi_k \tag{4.6}$$

is largest. This is the vector/matrix version of (4.4).

An example is shown in the following figure. Three equally-sized Gaussian classes are shown with class-specific mean vectors and a common covariance matrix ($\pi_1 = \pi_2 = \pi_3 = 1/3$). The three ellipses represent regions that contain 95% of the probability for each of the three classes. The dashed lines are the Bayes decision boundaries.



Recall that the decision boundaries represent the set of values x for which $\delta_k(x) = \delta_c(x)$; i.e. for $k \neq c$

$$x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k = x^T\Sigma^{-1}\mu_c - \frac{1}{2}\mu_c^T\Sigma^{-1}\mu_c$$

Note that there are three lines representing the Bayes decision boundaries because there are three pairs of classes among the three classes. That is, one Bayes decision boundary separates class 1 from class 2, one separates class 1 from class 3, and one separates class 2 from class 3. These three Bayes decision boundaries divide the predictor space into three regions. The Bayes classifier will classify an observation according to the region in which it is located.

Once again, we need to estimate the unknown parameters $\mu_1, \ldots, \mu_k$, and $\pi_1, \ldots, \pi_k$, and Σ; the formulas are similar to those used in the one-dimensional case. To assign a new observation $X = x$, LDA plugs these estimates into Eq. (4.6) and classifies to the class for which $\delta_k(x)$ is largest.

Note that in Eq. (4.6) $\delta_k(x)$ is a *linear* function of x; that is, the LDA decision rule depends on x only through a linear combination of its elements. This is the reason for the word linear in LDA.

4.6 Making predictions

Once we have estimates $\hat{\delta}_k(x)$, we can turn these into estimates for class probabilities:

$$\hat{P}(Y = k|X = x) = \frac{e^{\hat{\delta}_k(x)}}{\sum_{c=1}^{K} e^{\hat{\delta}_c(x)}}$$

So classifying to the largest $\hat{\delta}_k(x)$ amounts to classifying to the class for which $\hat{P}(Y = k|X = x)$ is largest.

When $K = 2$, we classify to class 2 if $\hat{P}(Y = 2|X = x) \geq 0.5$, else to class 1.

4.7 Other forms of Discriminant Analysis

$$P(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{c=1}^{K} \pi_c f_c(x)}$$

We saw before that when $f_k(x)$ are Gaussian densities, whith the same covariance matrix Σ in each class, this leads to Linear Discriminant Analysis (LDA).

By altering the forms for $f_k(x)$, we get different classifiers.

- With Gaussians but different Σ_k in each class, we get *Quadratic Discriminant Analysis (QDA).*
- With $f_k(x) = \prod_{j=1}^{p} f_{jk}(x_j)$ (conditional independence model) in each class we get *Naive Bayes.* (For Gaussian, this mean the Σ_k are diagonal, e.g. $Cov(X_i, X_j) = 0 \; \forall \; 1 \leq i, j \leq p$).
- Many other forms by proposing specific density models for $f_k(x)$, including *nonparametric approaches.*

4.7.1 Quadratic Discriminant Analysis (QDA)

Like LDA, the QDA classifier results from assuming that the observations from each class are drawn from a Gaussian distribution, and plugging estimates for the parameters into Bayes' theorem in order to perform prediction.

However, unlike LDA, QDA assumes that each class has its own covariance matrix. Under this assumption, the Bayes classifier assigns an observation $X = x$ to the class for which

$$\begin{aligned}
\delta_k(x) &= -\frac{1}{2}(x-\mu)^T \Sigma_k^{-1}(x-\mu) - \frac{1}{2}\log|\Sigma_k| + \log \pi_k \\
&= -\frac{1}{2}x^T \Sigma_k^{-1} x + \frac{1}{2}x^T \Sigma_k^{-1}\mu_k - \frac{1}{2}\mu_k^T \Sigma_k^{-1}\mu_k - \frac{1}{2}\log|\Sigma_k| + \log \pi_k
\end{aligned}$$

is largest.

Unlike in LDA, the quantity x appears as a quadratic function in QDA. This is where QDA gets its name.

> The decision boundary in QDA is non-linear. It is quadratic (a curve).

4.7.2 Naive Bayes

We use Naive Bayes classifier if the features are independant in each class. It is useful when p is large (unklike LDA and QDA).

Naive Bayes assumes that each Σ_k is diagonal, so

$$\begin{aligned}
\delta_k(x) &\propto \log\left[\pi_k \prod_{j=1}^{p} f_{kj}(x_j)\right] \\
&= -\frac{1}{2}\sum_{j=1}^{p}\frac{(x_j-\mu_{kj})^2}{\sigma_{kj}^2} + \log \pi_k
\end{aligned}$$

It can used for mixed feature vectors (qualitative and quantitative). If X_j is qualitative, we replace $f_{kj}(x_j)$ by probability mass function (histogram) over discrete categories.

4.8 LDA vs Logistic Regression

the logistic regression and LDA methods are closely connected. Consider the two-class setting with $p = 1$ predictor, and let $p_1(x)$ and $p_2(x) = 1p_1(x)$ be the probabilities that the observation $X = x$ belongs to class

1 and class 2, respectively. In the LDA framework, we can see from Eq. (4.4) (and a bit of simple algebra) that the log odds is given by

$$\log\left(\frac{p_1(x)}{1-p_1(x)}\right) = \log\left(\frac{p_1(x)}{p_2(x)}\right) = c_0 + c_1 x$$

where c_0 and c_1 are functions of μ_1, μ_2, and σ^2.

On the other hand, we know that in logistic regression

$$\log\left(\frac{p_1}{1-p_1}\right) = \beta_0 + \beta_1 x$$

Both of the equations above are linear functions of x. Hence both logistic regression and LDA produce linear decision boundaries. The only difference between the two approaches lies in the fact that β_0 and β_1 are estimated using *maximum likelihood*, whereas c_0 and c_1 are computed using the estimated mean and variance from a *normal distribution*. This same connection between LDA and logistic regression also holds for multidimensional data with $p > 1$.

- Logistic regression uses the conditional likelihood based on $P(Y|X)$ (known as *discriminative learning*).
- LDA uses the full likelihood based on $P(X, Y)$ (known as *generative learning*).
- Despite these differences, in practice the results are often very similar.

Remark: Logistic regression can also

fit quadratic boundaries like QDA, by explicitly including quadratic terms in the model.