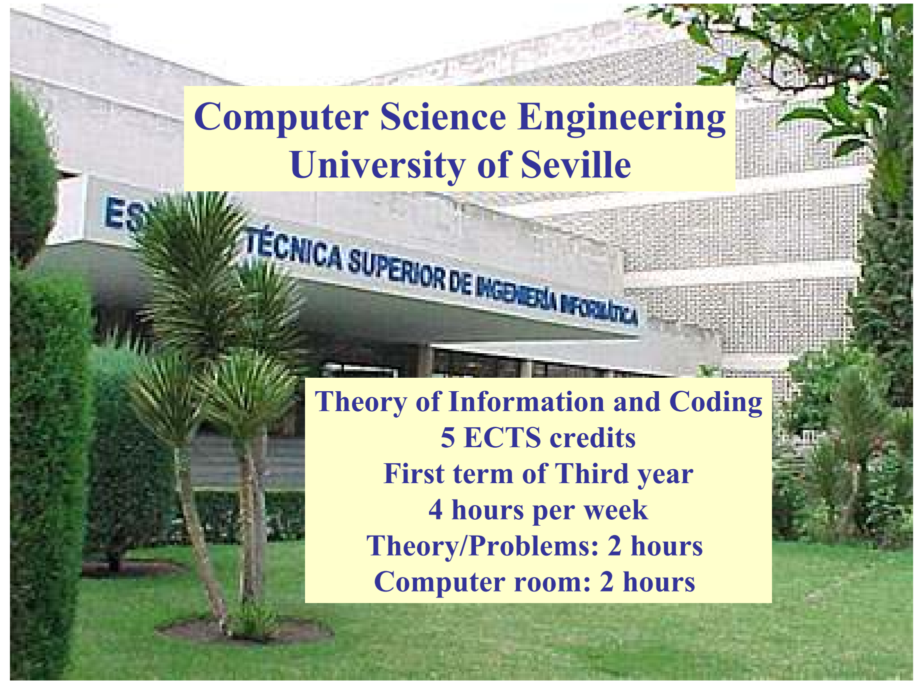
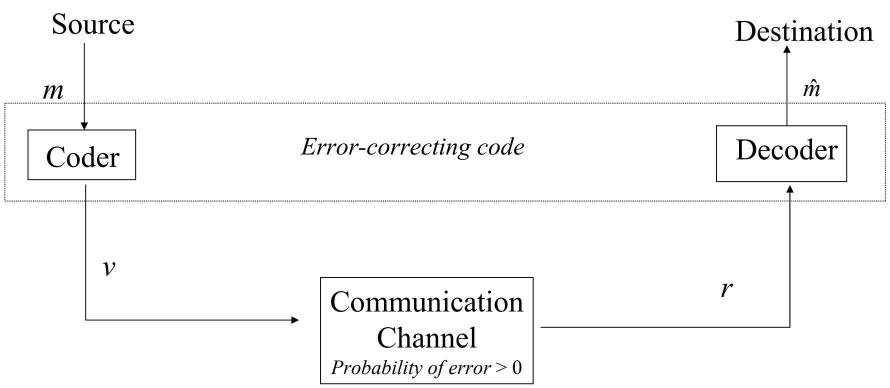
Teaching the Theory of Information and Coding with R

Rafael Pino Mejías, Mª Dolores Cubiles de la Vega
Statistics Department, University of Seville, SPAIN



The Theory of Information and Coding was developed to deal with the fundamental problem of comunication, that of reproducing at one point, either exactly or approximately, a message sent from another point.



Communication rate= size(m)/size(v)

Information Theory:
Theoretical capabilities of these communication systems considering the communication rate and the probability of error of the codes.
Coding Theory:
This is concerned with the design of effective error-correction codes. When the code is designed to reduce the requirement of memory resources for storing data, it is a compressor code.

Computer classes based on the R system (syllabus)

Part one: Information Theory
1. Introduction to Theory of Information
2. Discrete memoryless channels

Part two: Coding Theory
3. Linear codes
4. Cyclic codes

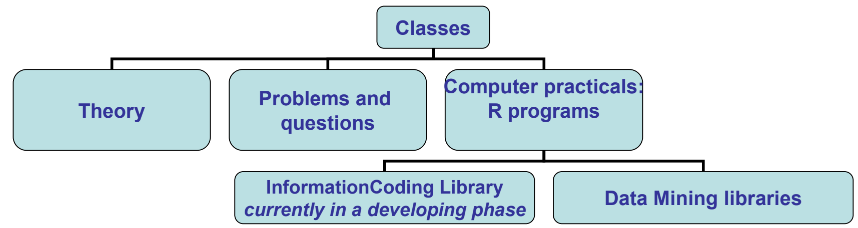
Part three: Compression
5. Data compression
6. Image compression

Part four:
7. Introduction to Data Mining

Benefits of the R system:

- Computer science students know the art of programming
- They understand a theme better when they program it
- It interfaces with other languages (C)
- It is a free system
- It has a powerful programming language
- It contains extensive and powerful graphics abilities
- The R system is continuously being developed

Teaching methodology



The practicals include the design, use and programming with the R system. On one hand, the "InformationCoding Library" is still being developed, and on the other hand, the last chapter is available in R.

InformationCoding Library

This library is still in the construction stage and can be found in four main blocks:

1. Entropy functions

These functions compute entropy (univariate, joint, conditional) and mutual information.

2. Simulation of communication channels

This block lets the transmission of a message over a digital communication channel be simulated. Some numerical and graphical summaries are produced.

3. Run-length codes

Coding and decoding of some classic algorithms used to compress data. Some of these codes are associated to the pioneer work of Claude Shannon.

4. Fixed-length codes

These functions code and decode messages with some of the most important algorithms used in coding practice.

InformationCoding library:
Block 1: Entropy functions

Some of these functions are:

entropyone(p): computes the entropy function given a probability p defining a two-result vector probability (p,1-p)

entropytwo(x,y): computes the entropy function given two probabilities x,y which define a three-result vector probability (x,y,1-x-y)

entropy(p): computes the entropy function from a probability vector p

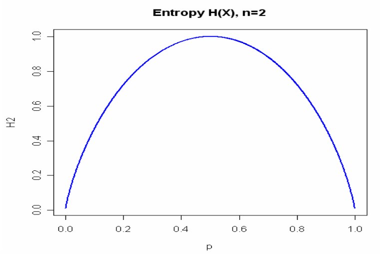
jointentropy(P): computes the joint entropy function from a probability matrix P

condientropy(P,margin): computes the conditional entropy function from a probability matrix by conditioning on the rows (margin=1) or the columns (margin=2). It also obtains the conditional entropy for each of the rows or columns.

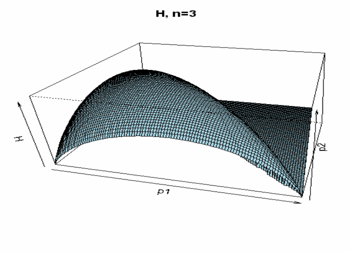
mutinf(P): computes the mutual information given a probability matrix P.

Some of the previous functions can be utilized to graphically represent the entropy:

```
entropyone<- function(p)
{
p*log2(1/p) + (1-p)*log2(1/(1-p))
}
curve(entropyone,0,1,1000, col="blue", lwd=2,
xlab="p",ylab="H2", main="Entropy, H(X),
n=2", type="l" )
```




```
entropytwo<-function(x,y)
{z<- 1-x-y
 ifelse(z>0,-x*log2(x)-y*log2(y)-z*log2(z),0)
}
x<-(0:100)/100
y<-(0:100)/100
z<-outer(x,y,entropytwo)
persp(x, y, z, theta = 15, phi = 30, expand =
0.5, col = "lightblue", xlab="p1", ylab="p2",
zlab="H", main="H, n=3")
```



InformationCoding library:
Block 2: Simulation of Communication Channels

simulate.channel(n, a, p, prober, mis): this simulates the transmission of a message formed by *n* symbols of the alphabet *a*, with vector probability *p* and a probability of error *prober*. *mis*=TRUE allows missing symbols (coded as -1) in the transmission.

simulate.bsc(n, l, prober): this simulates the transmission over a binary symmetric channel of a message formed by *n* binary vectors of size l, and a probability of error *prober*.

An example to illustrate the information produced by the function:
simulate.channel(100,c(0,1,2),c(1/3,1/3,1/3),0.15,TRUE)

```
IC 95% probability of error= ( 0.1495564  ,  0.1964436 )
Distribution of sent symbols (%):
   0   1    2
34.1 33.5 32.4
Distribution of received symbols (%):
 -1    0    1   2
 5.3 31.8 32.3 30.6

Distribution of source symbols presenting errors:
0:  17.59%
1:  17.31%
2:  16.97%
```

InformationCoding library:
Block 3: Run-Length Codes

The Run-Length codes currently implemented are Shannon, Shannon-Fano, and Arithmetic code, while Huffman code is in developing phase.

For example:
Shannon.code(*m*, *a*, *p*): this obtains the Shannon codes for the message *m* from an alphabet *a* with associated probability vector *p*.

Shannon.decode(*rec*, *a*, *p*): decoding the Shannon code.

STEP 1: Sort the messages m_i by sorting their probabilities into decreasing order.

```
messsort<-sort(p, decreasing=TRUE, index.return=TRUE)
m<-m[messort$ix]
p<-messort$x
```

STEP 2: Compute α_i: $\alpha_1 = 0$, $\alpha_2 = P(m_1)$, $\alpha_3 = P(m_1) + P(m_2), \ldots, \alpha_M = 1 - P(m_M)$ (accumulated probabilities)

```
computealpha<-function(p)
{    long<-length(p)
  c(0,cumsum(p[-long])         }
```

STEP 3: Determine n_i: $2^{n_i} \geq 1/p_i \geq 2^{n_i - 1}$ (determinig n_i such that is fulfilled)

```
seekN<-function(p)
{ ceiling(-log2(p)) }
```

STEP 4: The code of m_i is the binary expression of α_i up to the n_i^{th} binary digit

```
binarycode<-function(a,n)
{    auxi<-c()
     for(i in 1:n)
     {
     auxi[i]<-floor(a*2)
     a<-a*2-floor(a*2)
                     }
     auxi
}
```

```
> #Example
> p<-c(0.25,0.30,0.10,0.20,0.10,0.05)
> m<-c(1,2,3,4,5,6); a<- c(1,2,3,4,5,6)
> shannon(m,a,p)
m  p
2    0.30  0  0
1    0.25  0  1
4    0.20  1  0    0
3    0.10  1  1    0    0
5    0.10  1  1    0    1
6    0.05  1  1    1    1  0
```

InformationCoding library:
Block 4: Fixed-Length Codes

The last block of this library includes the following fixed-length codes:

Repetition codes
Linear codes:
- Hamming
- Golay
- Reed-Muller

Cyclic codes:
- Polynomial codes
- Reed-Solomon

InformationCoding library:
Fixed-Length Codes
Repetition codes

rep.code(s, N): this codes the message s with a repetition code, therefore each symbol in s is repeated N times.
rep.decode(*rec, N*): this decodes the received message *rec* by taking the majority vote of each N consecutive bits.
proberror(prober, N): this function computes the bit probability error of a repetition code in a symmetric binary channel with error probability *prober*.

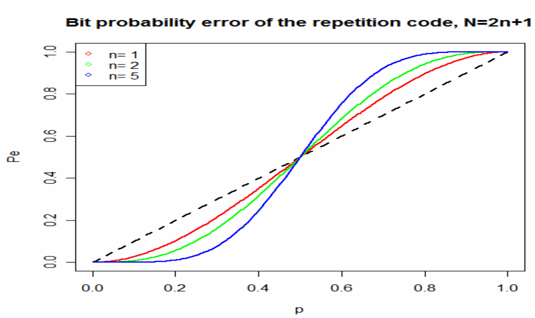
It is very easy to write some of these functions in R:

```
rep.code<-function(s,N)
{rep(s, each=N)}
```

InformationCoding library:
Fixed-Length Codes
Repetition codes

The function "proberror" can be graphically represented by means of the "curve" function:

```
curve(x*1, 0,1,100,col="black",lty=2,main="Bit probability error of the repetition code,
N=2n+1",lwd=2,ylab="Pe",xlab="p")
curve(proberror(x,3),0,1,200,col="red",lwd=2,add=TRUE)
curve(proberror(x,5),0,1,200,col="green",lwd=2,add=TRUE)
curve(proberror(x,11),0,1,200,col="blue",lwd=2,add=TRUE)
corner<-par()$usr
legend(corner[3], corner[4], col=c("red","green","blue"), pch=1, legend=paste("n=",c(1,2,5)))
```



InformationCoding library
Fixed-Length Codes, Linear Codes:
Hamming Codes

controlmatrix.Ham(*r*): this builds the control matrix of the Hamming code with length 2^r- 1 and dimension 2^r-r-1.
code.Ham(*m*,*r*): this codes the message *m* with the Hamming code.
decode.Ham(*rec*,*r*): this decodes the message *m* with the Hamming code.
proberr.Ham(*r*): this function computes the block error probability of the Hamming code.
codewords.Ham(*r*): list of the codewords of the Hamming code.

```
> codewords.Ham(3)
 [1,]  0  0  0  0  0  0  0
 [2,]  0  0  0  1  1  1  1
 [3,]  0  0  1  0  0  1  1
 [4,]  0  0  1  1  1  0  0
 [5,]  0  1  0  0  1  0  1
 [6,]  0  1  0  1  0  1  0
 [7,]  0  1  1  0  1  1  0
 [8,]  0  1  1  1  0  0  1
 [9,]  1  0  0  0  1  1  0
[10,]  1  0  0  1  0  0  1
[11,]  1  0  1  0  1  0  1
[12,]  1  0  1  1  0  1  0
[13,]  1  1  0  0  0  1  1
[14,]  1  1  0  1  1  0  0
[15,]  1  1  1  0  0  0  0
[16,]  1  1  1  1  1  1  1
```

InformationCoding library
Fixed-Length Codes, Linear Codes: Golay Codes

genmatrix.Golay(*r*): this builds the generator matrix of the Golay-24 code (r=24) or the Golay-23 code (r=23).
code.Golay(*m, r*): this codes message m with the Golay code.
decode.Golay(*rec, r*): this decodes message *rec* with the Golay code.

Reed Muller Codes

genmatrix.RM(*r*): this builds the generator matrix of the Reed Muller code with length 2^r and dimension r+1.
code.RM(*m, r*): this codes message m with the Golay code.
decode.RM(*rec, r*): this decodes message *rec* with the Golay code.

Example of Reed Muller code

These are the matrices of Reed Muller generator of order 3 and 4 respectively:

```
> genmatrix.RM(3)
1  1  1  1  1  1  1  1
0  1  0  1  0  1  0  1
0  0  1  1  0  0  1  1
0  0  0  0  1  1  1  1
```

```
> genmatrix.RM(4)
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1
0  0  1  1  0  0  1  1  0  0  1  1  0  0  1  1
0  0  0  0  1  1  1  1  0  0  0  0  1  1  1  1
0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1
```

Example of Reed Muller code

We suppose the following message formed by 10 vectors of size 4:

```
> m
      [,1] [,2] [,3] [,4]
 [1,]    0    0    0    1
 [2,]    0    1    0    1
 [3,]    0    0    0    0
 [4,]    0    0    1    0
 [5,]    0    1    1    0
 [6,]    1    1    1    1
 [7,]    1    0    1    1
 [8,]    0    1    1    1
 [9,]    1    1    1    0
[10,]    0    1    0    0
```

Given the generator matrix G then the coding of a vector s becomes

(*s*%*%*G*)%%2

The resulting code would be:

```
>t<- code.RM(m,3)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
 [1,]    0    0    0    0    1    1    1    1
 [2,]    0    1    0    1    1    0    1    0
 [3,]    0    0    0    0    0    0    0    0
 [4,]    0    0    1    1    0    0    1    1
 [5,]    0    1    1    0    0    1    1    0
 [6,]    1    0    0    1    0    1    1    0
 [7,]    1    1    0    0    0    0    1    1
 [8,]    0    1    1    0    1    0    0    1
 [9,]    1    0    0    1    1    0    0    1
[10,]    0    1    0    1    0    1    0    1
```

We can simulate the transmission of the coded message with the function *simulate.csb:*

```
> r<- simulate.csb(t,0.1)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
 [1,]    1    0    0    0    1    1    1    1
 [2,]    0    1    0    1    1    1    1    0
 [3,]    0    0    0    0    0    0    0    0
 [4,]    0    0    1    1    0    0    1    1
 [5,]    0    1    1    0    0    1    1    0
 [6,]    1    0    1    1    0    0    1    0
 [7,]    0    1    1    0    0    0    1    1
 [8,]    0    1    1    0    1    0    0    1
 [9,]    1    0    0    1    1    0    0    1
[10,]    0    1    1    1    0    0    0    1
```

The decoding of the received message is obtained with the function decode.RM

```
> decode.RM(r,3)
      [,1] [,2] [,3] [,4]
 [1,]    0    0    0    1
 [2,]    0    1    0    1
 [3,]    0    0    0    0
 [4,]    0    0    1    0
 [5,]    0    1    1    0
 [6,]    0    0    0    0
 [7,]    0    0    1    0
 [8,]    0    1    1    1
 [9,]    1    1    1    0
[10,]    0    0    0    0
```

InformationCoding library
Fixed-Length Codes, Cyclic codes
Polynomial codes

These are based on the polynom R library
Systematic coding of a cyclic polynomial code
Syndrome decoding based on a reduced syndrome table

Reed Solomon Codes

These require the implementation of the elements of the Galois Field GF(2^n), including the sum and product functions

The Reed Solomon codes work with blocks of *n* symbols: whereas in the previous codes *K* bits are coded as a codeword of size *N* bits, now *K* blocks of *n* bits are coded by *N* blocks of n bits.

Data Mining

The subject also includes an introduction to the main machine-learning models. The brief theoretical presentation is accompanied by some examples.

Neural Networks: multilayer perceptron with the *nnet* library.

CART: Classification and regression trees with the *rpart* library.

SVM: Support Vector Machines with the *svm* function in the *e1071* library.

Example of Reed Solomon Codes
n=3, K=3, N=7

We want to send this message, 3 elements of GF(2^3):

110 110 111

which becomes a coded message, 7 elements of GF(2^3),:

001 000 111 000 110 110 111

Polynomgen.RS(N, K, n): generator polynomial for a Reed Solomon code with length N, dimension K, over a Galois Field GF(2^n).
code.RS(m, N, K, n): codes the message m with the Reed Solomon code.
decode.RS(rec, N, K ,n): decodes the message *rec* with the Reed Solomon code. It is based on the Berlekamp Massey algorithm.