R programming

José R. Valverde
jrvalverde@cnb.csic.es
CNB/CSIC

Classification and Machine Learning

In God we trust. All others must bring data.

Robert Hayden, Plymouth State College

Introduction to Deep Learning

library(caret)

If I have seen farther than others, it is because I was standing on the shoulder of giants.

== Isaac Newton

If I have not seen as far as others, it is because giants were standing on my shoulders.

== Hal Abelson

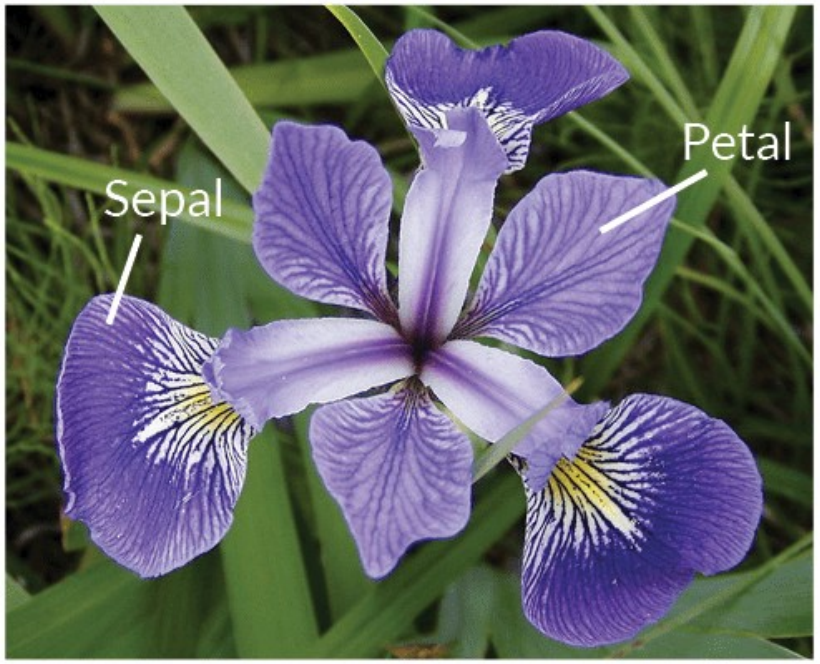
In computer science, we stand on each other's feet.

== Brian K. Reid

Getting started

- Collect the data that you will use to train the computer
- Classify the data yourself first
- This should give you a set of well known, well classified data points to use in training

```
data(iris)
head(iris)
str(iris)  # show structure of iris
```



Iris Versicolor

Iris Setosa

Iris Virginica

Inspect the data

- See if you can spot correlations between measures and groups

  `pairs(iris)`

- Don’t believe me, just watch!

```
# Overall correlation
print( cor(iris[ ,1:4]) )
# get 'iris' levels
species_names <- levels(iris$Species)
measures <- 1:4  # columns with measues
specie <- 5      # column with species
```

Search for correlations

```
# explore species correlations (e.g.)
print(species_names[1])
print( cor(iris[iris$Species==species_names[1],
           measures]) )
```

Search for correlations

```
# explore species correlations (e.g.)
print(species_names)
print( cor(iris[iris$Species=="setosa",
           measures]) )
```

- Compute the correlations for each species using function 'cor'

Search for correlations

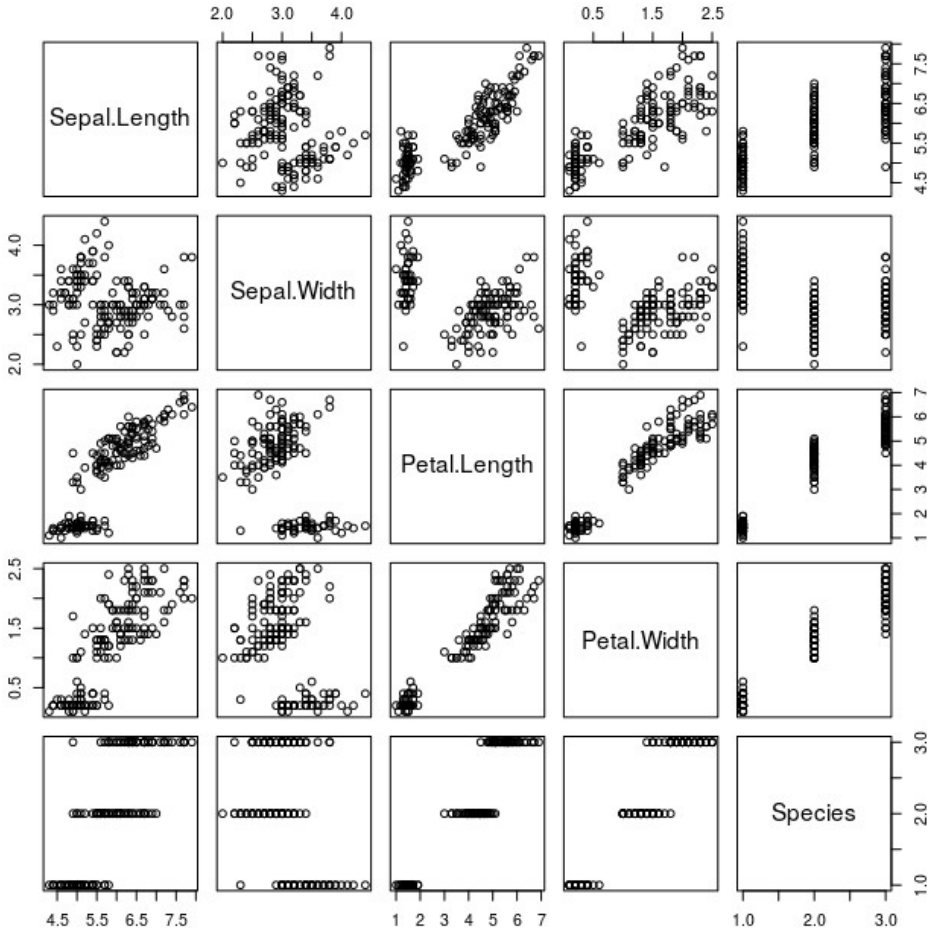
```
# explore species correlations (e.g.)
print(species_names)
print( cor(iris[iris$Species=="setosa",
           measures]) )
```

- Compute the correlations for each species

```
for (i in species_names)
    print(cor(
           iris[ iris$Species == i, measures] ))
```

Draw a scatterplot matrix

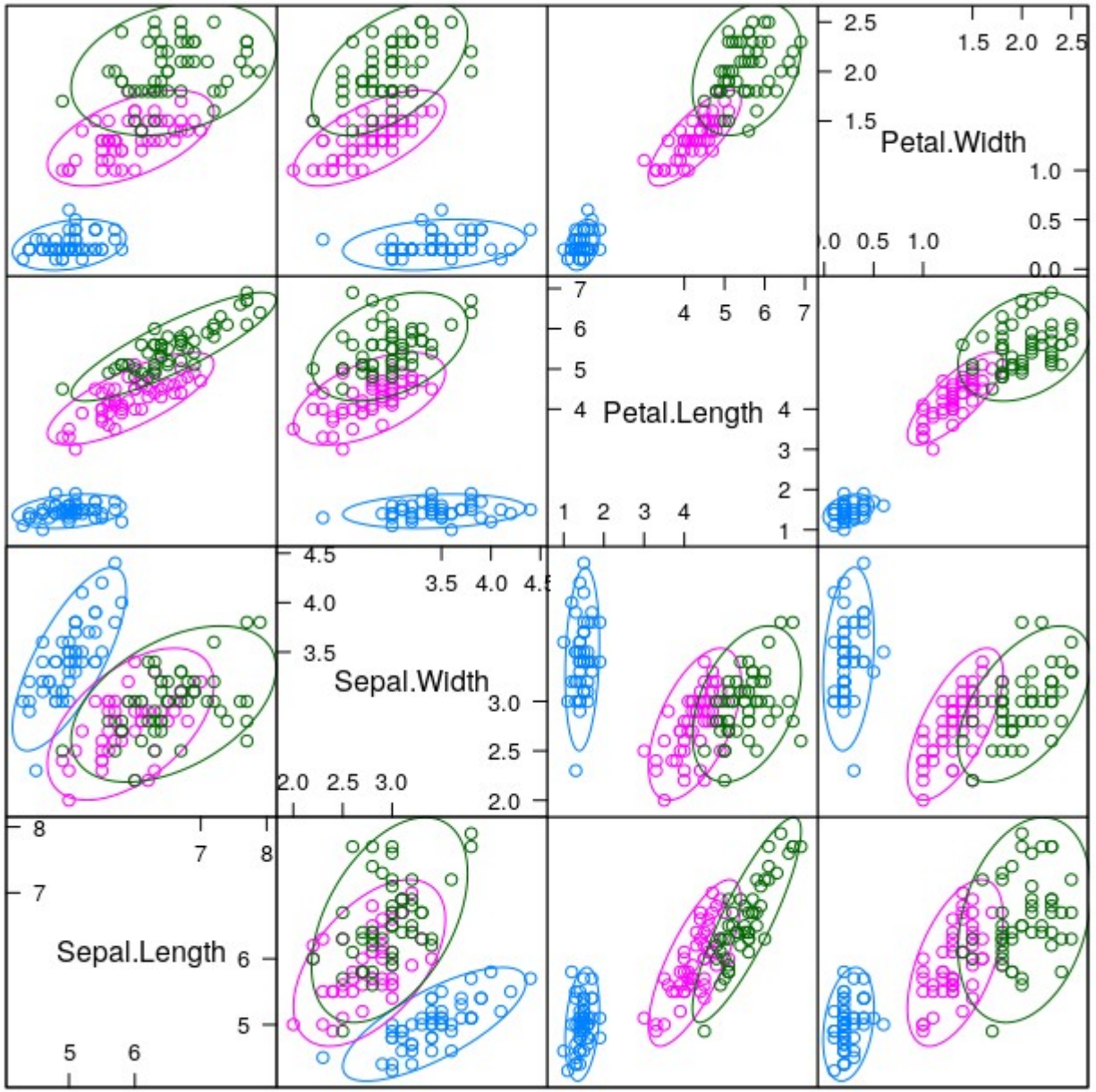
- **`plot(iris)`**



Inspect the data

```
# visualize the 'iris' data
print(featurePlot(x=iris[ , measures],   # 1:4
                  y=iris[ , specie],     # 5
                  plot="ellipse"))
# try using “box” or “density” instead of “ellipse”
# in feaurePlot
```

- With good correlations we could resort to linear models
- Here, when we combine all three species, the correlation is somewhat better than when we look at each species sparately.
- We would get better results if we analyze all of them simultaneously



Scatter Plot Matrix

Prepare your data

- Reverse psychology:
- If you have huge and tiny observables…
- Which ones would draw your attention?

- We deal with it *normalizing* the data

- Consists in adjusting the values within each group to fall between the range [0,1]

Data normalization

- Although we do not really need it, we'll still see how to do it:
  - You can make a "normalize()" function and then apply it to all values in a column:

we will get a single columns (and use it as a one-dimensional vector), say 'x'
first we need to "shift to zero" all values:
find the minimum with function min(): e.g. min(x)
subtract the minimum from all the values : e.g. x - min(x)
(you can make a function to do this)
then we need to find the range of values (maximum - minimum)
we can use "max(x) - min(x)"
or we can find the maximum of the "zero-shifted" values
(you may define also a function range() to find the range of a vector)
finally, we divide the zero-shifted values by the range.

normalized value = (shifted to zero values) /
range of all original values

a possible solution

```
shift.to.zero <- function(x)
    x - min(x)
range <- function(x)
    max(x) - min(x)
normalize <- function(x)
   shift.to.zero(x) / range(x)
```

apply it to your data

We can use "apply()" and save the result in a new dataset

```
by.column <- 2
iris_norm <- apply(iris[ ,measures],
                   by.column, normalize)
summary(iris_norm)
```

Getting ready

- Don’t put all your eggs on the same basket
- If you use up all the classified data you have to train the computer
  - How are you going to see if it has learnt to classify?
  - Classifying data that you don’t know?

- We will use a large part to train
- And keep a smaller part to check if it has learnt

Know your data

- *Species* is a categorical variable that can take a limited number of values

```
table(iris$Species)
# prop.table here will give x/sum(x) (proportions);
# by multiplying by 100 we get percentages;
# then, we round them to 1 decimal digit
prop.table( table( iris$Species ) )
round( prop.table(table(iris$Species)) * 100,
       digits=1 )
```

- The data is evenly distributed among species
  - 50-50-50 or 0.33-0.33-0.33 or 33.3-33.3-33.3

Splitting

- We will typically use ⅔ of the data to train
- And keep ⅓ to test
- If we use the first ⅔ of ‘iris’
  - We’d choose for training two species
  - And leave the other for testing
- How will the computer learn to recognise (classify) the third species if it has never seen it?

Splitting the data with 'caret'

We'll use a 3/4:1/4 ratio this time:

```
# Create an index to split based on it
chosen <- createDataPartition(
    iris$Species, p=0.75, list=FALSE)
print(chosen)


# Subset training set with the index
iris.training <- iris[chosen,]
# Subset test set with -chosen
iris.test <- iris[-chosen,]
```

The training process

- We will use some observations together with their correct classifications for training
  - iris[, measures] and iris[,specie]
- Which, after selecting 75%, will become
  - iris.training[, measures] and iris.training[, specie]
- Afterwards, we give *only the observations* (iris.test[, measures]) to the computer and ask for their classification, but not their solutions
- And contrast the computed results with the actual classification (iris.test[,specie] which we kept hidden)

Build an AI model

```
# Overview of algorithms supported by caret
names( getModelInfo() )
```

- Make a decision (we'll use KNN) and...

```
# Train a model
knn.model <- train(
    iris.training[, measures],
    iris.training[, specie],
    method='knn')
```

k-NN

k-Nearest-Neighbours is a classification system (much like clustering) based on previously stored, labeled data

You take some data that has been previously collected and classified (labeled)

You use this data to "train" the computer

And then you ask the computer to classify new data.

Use the model

- We can now use our new model to make predictions on new data

```
# Predict the labels of the test set
predicted.specie <- predict(object=knn.model,
    iris.test[, measures])
# Evaluate the predictions
table(predicted.specie)
# Compare with known correct results
print(confusionMatrix(predicted.specie,
      iris.test[, specie]))
```

Valverde's minimum

If you can do it, it can be improved

Si puedes hacerlo, es mejorable

Jose R. Valverde

Preprocess the data

```
# Train the model with preprocessing (scaling and
# centering)
knn.model <- train(iris.training[, measures],
    iris.training[, specie], method='knn',
    preProcess=c("center", "scale"))
# Predict values
predictions <- predict(object=knn.model,
    iris.test[, measures], type="raw")
# Confusion matrix
print(confusionMatrix(predictions,iris.test[,specie]))
```

Máxima de Calleja

In case of doubt, expect the worst

En caso de duda, ponte en lo peor

M. Calleja

NEVER trust yourself... or the computer!

- Learning depends on both exposure to *good* data and use of *good* systematics
- How can we be sure it did work?
  - *Really*
  - Not by chance

- R is a statistical package. We are using R. We proud ourselves on our statistics
- R is a programming language. We can introduce variations *ad nauseam*.

Repeat many times and check

- There are many ways, most depend on the learning method.

```
# repeat 10 times 10 rounds of cross-validation (not much, but...)
cv.control <- trainControl( method="repeatedcv",
    repeats=10, number=10)    # you can try 100 and 100 (or more)

metric <- "Accuracy"  # use accuracy as a metric for automatic
                      # evaluation of each model's learning
# Now do the training again, but this time repeating many times,
# evaluating all the mnodels produced for accuracy and selecting
# the best one
knn.model <- train(Species ~ ., data=iris.training,
    method="knn", metric=metric, tuneLength=50,
    trControl=cv.control )
knn.pred <- predict(knn.model,
    iris.test[ , measures], type="raw")
print(confusionMatrix(knn.pred, iris.test[ , specie]))
```

A short disgression

- Save your script
- Try variants of the next two slides until you feel that you understand and are comfortable with them.
- Then procceed to the subsequent exercise

A gentle reminder

```
# group some words into a vector
w <- c('January', 'February, 'March', 'April', 'May', 'June')
# group values {1..10} in a vector
n <- 1:10
# group commands as if they were one
{ print(w) ; print(n) }
# make 'i' sucessively take all the values in the vector 'n',
# from 1 to 10, and print 'i' each time
for (i in n) print(i)
# make 'name' sucessively take all the values in the vector 'w',
# and put them one after the other separated by tabulators ('\t')
for (name in w) {
    cat(name, '\t')     # cat() is like print+paste, without 'end of line'
}
cat('\n')               # '\n' means 'end-of-line' (i.e. start a new line)
```

...more reminiscences

```
# arrange values 1..12 in a 3x4 matrix
m <- matrix(data=1:12, nrow=3, ncol=4)
m[3,4]              # get value at row 3 and column 4

# make a data frame with four named columns
d <- data.frame(tom=1:3, dick=4:6, harry=7:9, joe=10:12)
d$joe               # get the contents of column named column 'joe'
d$joe[3]            # get the value at position 3 of column 'joe'

# make a list with three elements
l <- list(words=v, numbers=n, matrix=m)
l[[2]]              # see 2nd element
l$words             # see element named 'words'
l[['data']] <- d    # add a new element called 'data'
l$data              # see that we can access its contents
```

Common A/I models

```
# a) linear algorithms
"lda"        # Linear Discriminant Analysis

# b) nonlinear alorithms
"rpart"      # Classification And Regression Trees
"knn"        # k-Nearest Neighbors

# c) advanced algorithms
"svmRadial" # Support Vector Machines
"rf"         # Random Forest
```

Was KNN good enough?

- You may want to try different learning methods: define a vector with the methods' names

```
methods <- c('knn', 'lda', 'rf',
             'rpart', 'svmRadial')
```

- Try to repeat the last analysis with each of these methods

  - see next slide for some hints

Some tips

- Use a ‘for’ loop with a variable called 'meth' that will take all the values in the vector of methods' names
- Enclose the last training we did within { }
- Substitute the name of the training method in train() ('knn') with *meth*
- In each pass of the loop you create a new trained model, to keep it, you will need to save it in a list so we can use it later
  - start with an empty list before the for loop: models <- list()
  - after running train, add the model to the list
- You may need other packages:
  - lattice, MASS, rpart, kernlab, e1071, ranger, randomForest...
  - don't worry, you will find out as you try and get errors.

```
cv.control <- trainControl( method="repeatedcv",
                            repeats=10, number=10)
metric <- "Accuracy"

methods <- c('knn', 'lda', 'rpart', 'rf', 'svmRadial')
models <- list()    # start with an empty list of trained models
for (meth in methods) {
    print(meth)
    model <- train(Species ~ ., data=iris.training,
                   method=meth, metric=metric, tuneLength=50,
                   trControl=cv.control )
    pred <- predict(model, iris.test[ ,1:4], type="raw")
    print( confusionMatrix(pred, iris.test[ ,5]) )

    models[[meth]] <- model
}
```

Select the best model

- Summarize accuracy of the models
  - We need a list with the models

```
# summarize accuracy of models
results <- resamples(models)
summary(results)
dotplot(results)
bwplot(results)
```

- κ takes into account the expected error rate

Save all your work

- It is time we learn the main ways to save our work
  - **sink**(filename) will save the screen output to a file
  - **write.table**() will save 2D data into a file (useful for data exchange and publication
  - **saveRDS()** will save *one object* into an RDS (R specific) data file
  - **save**() will save *anything* into an RData (R specific) file

sink()

- sink(*filename*)
  - will redirect all output from your screen to a file named as indicated (you do not see anything on the screen anymore until you close the sink)
    - sink('myfile.txt')
- sink(*filename*, split=TRUE)
  - does the same but splits output: you save all output but you also see it in the terminal at the same time
    - sink('myfile.txt', split=TRUE)
- sink()
  - closes the last sink file opened (if you opened more than one, the others continue working and copying the output)

write.table()

- **`write.table(x, file='', append=F, sep=' ', row.names=T, col.names=T, etc...)`**
  - write table will try to save 'x' as a table, use it with vectors, a matrix or a data frame
  - x: the data to be saved
  - file: the name of the text file where the table will be saved (please, choose an extension (*.xxx*) that matches the separator (*.tab, .csv, .txt*))
  - append: T/F, whether you want to add the data in 'x' at the end of the contents of the file (useful to add new data to an existing table)
  - sep: how do you want to separate the values: commas ',', tabulators '\t', spaces ' ', etc...
  - row.names, col.names: T/F, whether you want to also save the row and/or column names in the file (note that row names can be problematic)
- **`some.variable <- read.table(file,...)`**
  - will read the table into some.variable

saveRDS()

- **`saveRDS(object, file, etc...)`**
  - saves a single R object (variable, vector, matrix, data.frame, list, whatever..., but only one) into a file.
  - object: the object to save
  - file: the name of the file to save it into (please, end the file name with the *extension* '*.rds*')
  - help(saveRDS) for more details
- **`some.variable <- readRDS(file)`**
  - will read the variable contained in the named file and save it into 'variable'

save()

- **`save(..., file, ascii=F, etc...)`**
- save() can save one or more objects in an external file of RData type
  - first list all the objects you want to save
  - file is the name of the file where you want to save the objects (please, use '.RData' as the final part or *extension* of the file name)
  - see help(save) for more details
- **`load(file)`** will read all the contents of an RData file and create all the variables contained in it in your workspace.

Save your work

- You will need what you did for later reference:
  - the training and testing subsets are tables (in this case): you can save them with write.table()
  - we saved all the training models in a list, we can save them all together with saveRDS()

```
write.table(iris.training,
            '2023-02-24-iris_training.txt')
write.table(iris.test,
            '2023-02-24-iris_test.txt')
saveRDS(results,
            '2023-02-24-list_of_models.rds')
```

Appendix

- One-off encoding
  - Used for categorical variables
  - Consists in a matrix with as many columns as the original data and as many rows as levels
  - Each column has a 1 in the row corresponding to its value and 0 elsewhere
  - E.g. sequences like ‘ATGCATGC’

	A	T	G	C	A	T	G	C
A	1	0	0	0	1	0	0	0
C	0	0	0	1	0	0	0	1
G	0	0	1	0	0	0	1	0
T	0	1	0	0	0	1	0	0

One-off encoding example

```
nt <- c('A', 'C', 'G', 'T')   # possible nucleotides
seq <- 'ATGCATGC'             # the sequence
seqlen <- nchar(seq)          # length of the sequence
seq_as_vector <- c()          # seq as an empty vector
for (i in 1:seqlen) {
    letter <- substring(seq, i, i)  # get nt at position i
    seq_as_vector <- c(seq_as_vector, letter) # and add it
}
# create a one-off-encoded matrix filled with zeroes
ooe_seq <- matrix(nrow=length(nt), ncol=seqlen,
                  data=rep(0, length(nt)*seqlen))
rownames(ooe_seq) <- nt    # use nt as row names (for indexing)

for (i in 1:seqlen) {         # for all the nt in the sequence
    letter <- seq_as_vector[i]   # this is the nt at position i
    ooe_seq[ letter, i] <- 1     # set the cell at column i that
}                             # corresponds to the letter to 1
colnames(ooe_seq) <- seq_as_vector
```

and so on...

- Welcome to the wonderful world of Artificial Intelligence and Machine Learning.
- You are now ready to delve into Deep Learning methods in more detail.
- But remember...

There is no use in Artificial Intellidence
in Natural's absence

J. R. Valverde

De nada sirve la Inteligencia Artificial
cuando falta la Natural

Thank you

Questions?

To learn more:

https://tanthiamhuat.files.wordpress.com/2018/04/deeplearningwithr.pdf