Tutorial 1: Introduction to Machine Learning with Python¶

Jonathan Ish-Horowicz

12/01/2020

Tutorial 1: Introduction to Machine Learning with Python

The goal of this tutorial is to introduce a typical workflow in carrying out ML in R. Similarly to last week's Python tutorial, this includes:

1. accessing and organising data,
2. assessing the data,
3. visualising the data,
4. a) creating training, b) test datasets and c) learning a model using them and evaluating its performance.

1) Load Data

We load the same iris dataset as last week, which has 150 samples and 4 attributes. There are 3 classes (species).

In R we can load the Iris dataset from the `datasets` package:

```
# Load the iris dataset
library(datasets)
library(matlib)
data(iris)
iris$Species <- as.character(iris$Species)
```

2) Statistics of the dataset

Now compute the mean, standard deviation, minimum and maximum of each attribute.

Suggestion: use the the `group_by` and `summarise_all` functions from the `dplyr` library.

```
library(dplyr)

# Calculate the mean of each attribute
print(colMeans(select(iris, -Species)))
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##     5.843333     3.057333     3.758000     1.199333
```

```
# Calculate the standard deviation of each attribute
print(apply(select(iris, -Species), 2, sd))
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    0.8280661    0.4358663    1.7652982    0.7622377
```

```r
# Calculate the minimum of each attribute
print(apply(select(iris, -Species), 2, min))
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##          4.3          2.0          1.0          0.1
```

```r
# Calculate the maximum of each attribute
print(apply(select(iris, -Species), 2, max))
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##          7.9          4.4          6.9          2.5
```

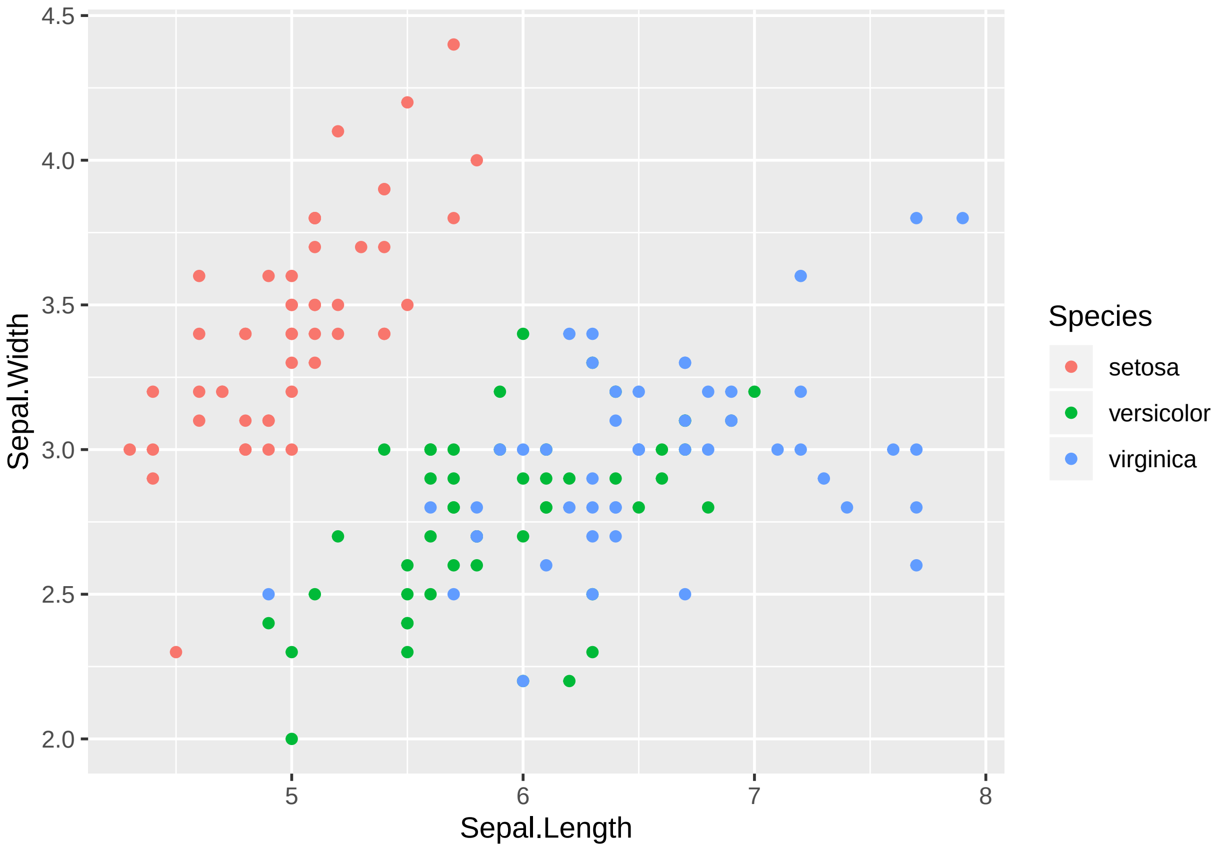
3) Visualise the dataset

Make some exploratory plots here.

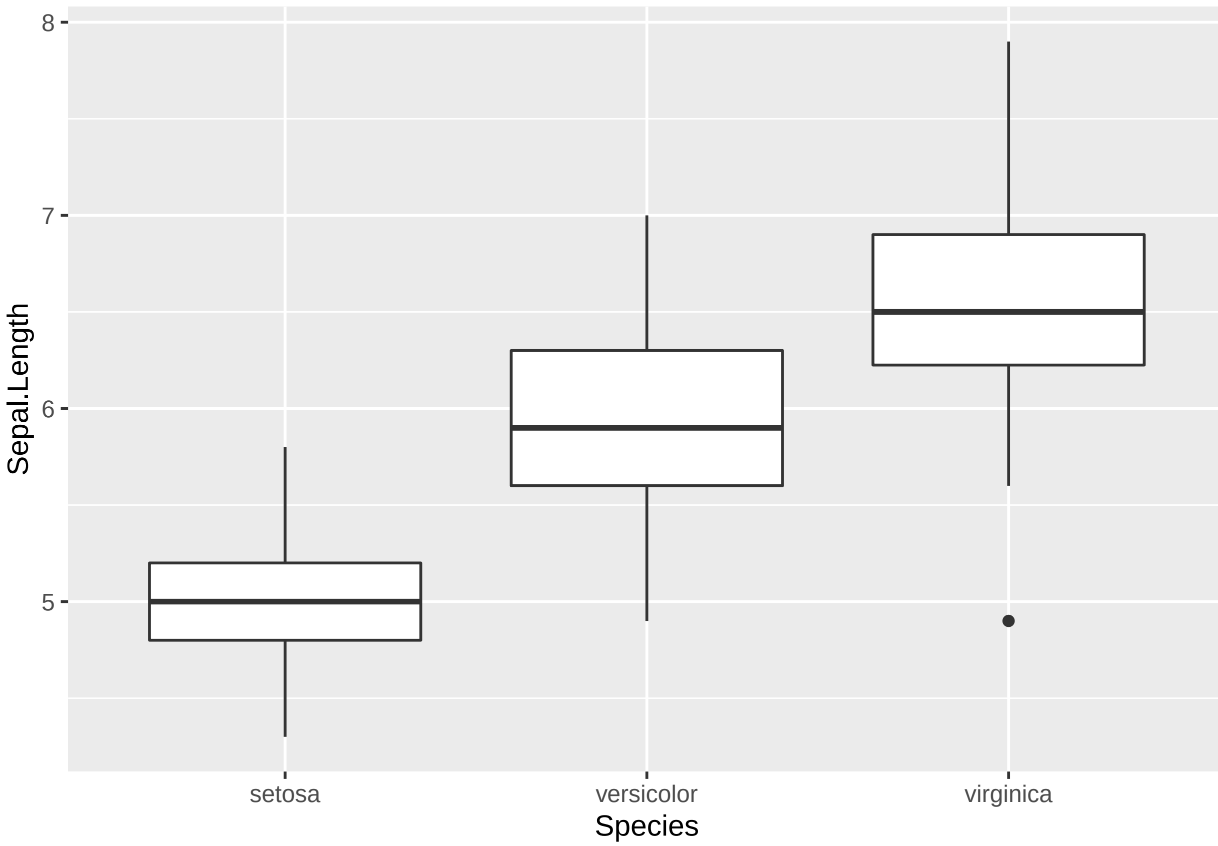
Suggestion: use the `ggplot2` library. For a nice pairs plot use the `ggpairs` function from `GGally` (a `ggplot2` extension library).

```r
library(ggplot2)
library(GGally)

ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species))+geom_point()
```




```r
ggplot(iris, aes(Species, Sepal.Length))+geom_boxplot()
```



4) Classification using Least Squares

Here we will be carrying out classification using the least squares formulation on 2 classes of the dataset.

a) Create separate datasets for the classes `setosa` and `versicolor`.

```
# your code here
setosa = iris %>% filter(Species == "setosa")
versicolor = iris %>% filter(Species == "versicolor")
# result should be two dataframes (one for each for setosa, versicolor classes),
# each with dim (50,5) - 4 attributes plus column for class
stopifnot(all(dim(setosa)==c(50,5)))
stopifnot(all(dim(versicolor)==c(50,5)))
```

b) add a column to each dataset where the column is 1 if the class is `setosa` and -1 otherwise.

```
# your code here
setosa$Output = 1
versicolor$Output = -1
# result should add a column to each of setosa and versicolor
stopifnot(all(dim(setosa)==c(50,6)))
stopifnot(all(dim(versicolor)==c(50,6)))
```

c) create training and test datasets, with 20% of the data for testing. This 80 training points and 20 testing points in total (half this per class).

```r
# your code here
dataset = rbind(setosa, versicolor)

tr.index = sample(100, 80)
tst.index = rownames(dataset)[!rownames(dataset) %in% tr.index]

training.data = dataset[tr.index,]
test.data = dataset[tst.index,]

# resulting dataframes (one each for training and test data) should have
# the appropriate sizes
stopifnot(all(dim(training.data)==c(80,6)))
stopifnot(all(dim(test.data)==c(20,6)))
```

d) apply the least squares solution to obtain an optimal solution for different combinations of the 4 available attributes. The code to create a list containing all the combinations of the attributes has been provided.

```r
# Creates all possible combinations of attributes
# attribute.combinations is a list whose elements are lists of attributes
attribute.names <- colnames(iris)[1:4]
attribute.combinations <- do.call(
  c,
  lapply(1:4, function(i) as.list(data.frame(combn(attribute.names, i))))
  )
names(attribute.combinations) <- 1:length(attribute.combinations)


return.predictions <- function(attribute.names, training.data, test.data) {

  # Format training and test data (as matrices)
  ### your code here
  X.train = as.matrix(training.data[,attribute.names])
  X.test = as.matrix(test.data[,attribute.names])

  X.train = cbind(matrix(1,nrow(X.train)), X.train)
  X.test = cbind(matrix(1,nrow(X.test)), X.test)

  y.train = as.matrix(training.data[,"Output"])

  # Calculate optimal weights
  ### your code here
  params = inv(t(X.train)%*%X.train)%*%t(X.train)%*%y.train

  # Make predictions
  ### your code here
  predictions = X.test %*% params

  predictions
}
```

e) evaluate which input attributes are the best.

```r
# Calculate the mean square error between some predictions and
# the corresponding testing data
return.mse <- function(predictions, testing.data) {
```

```r
  ### your code here

  mse = mean((predictions - testing.data[,'Output'])^2)
  mse
}

# Calculate the test MSE for each the elements of attribute.combinations
# by calling return.predictions and return.mse
results = as.data.frame(
  sapply(
    attribute.combinations,
    function(att) return.mse(
      return.predictions(
        att, training.data,test.data),test.data
      )
    )
  )

colnames(results)[1] = "MSE"
results$Attributes = sapply(attribute.combinations, function(x) paste(x, collapse = ","))
# FOR LOOPS MUST BE AVOIDED
# for (att in attribute.combinations){
#   #print("For params: ",att,"\n MSE: \t",
#               #return.mse(return.predictions(att, training.data,test.data),
#                #          test.data))
#   preds = return.predictions(att, training.data,test.data)
#   print(att)
#   print(paste("MSE: ", return.mse(preds, test.data)))
# }

print(results)
```

```
##           MSE                                       Attributes
## 1  0.64947002                                     Sepal.Length
## 2  0.64947002                                      Sepal.Width
## 3  0.64947002                                     Petal.Length
## 4  0.64947002                                      Petal.Width
## 5  0.21885976                         Sepal.Length,Sepal.Width
## 6  0.21885976                        Sepal.Length,Petal.Length
## 7  0.21885976                         Sepal.Length,Petal.Width
## 8  0.21885976                         Sepal.Width,Petal.Length
## 9  0.21885976                          Sepal.Width,Petal.Width
## 10 0.21885976                         Petal.Length,Petal.Width
## 11 0.03975984             Sepal.Length,Sepal.Width,Petal.Length
## 12 0.03975984              Sepal.Length,Sepal.Width,Petal.Width
## 13 0.03975984             Sepal.Length,Petal.Length,Petal.Width
## 14 0.03975984              Sepal.Width,Petal.Length,Petal.Width
## 15 0.03311679 Sepal.Length,Sepal.Width,Petal.Length,Petal.Width
```