Lab on Galaxy NGC 7531

Introduction

In machine learning, one common task is regression analysis: using the input data, we want to predict the output. This is a typical machine learning task: Given students' first year score, I want to predict their final year grades. Given patients' physical condition, doctors want to predict their life expectancy etc…

Today, we will play with an astronomical dataset. It records the relative positions of stars in Galaxy NGC 7531. We would like to **use these location information** to predict the **velocity** of stars.

We will use this data set to practice using data frames on a real-world application.

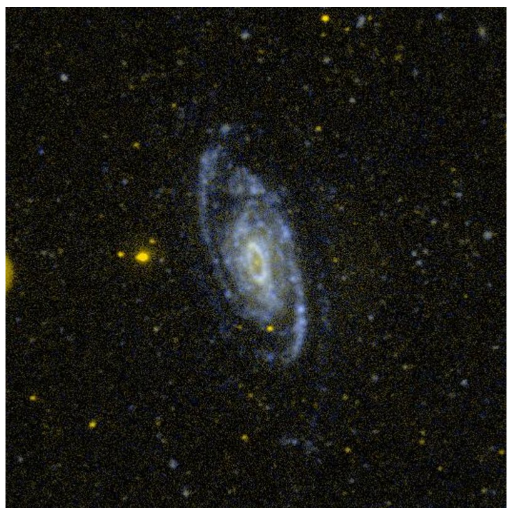


Figure 1: Galaxy NGC 7531, from Wikipedia.

Loading the data

The data frame has been downloaded and processed for you. Run the following command to load it:

```
set.seed(1)
galaxy <- read.table("galaxy")
```

Let us look at a summary of our dataset:

```
dim(galaxy)
```

```
## [1] 323   5
```

```
head(galaxy)
```

```
##   east.west north.south angle radial.position velocity
## 1  8.462789   -38.17317 102.5            39.1     1769
## 2  7.964978   -35.92769 102.5            36.8     1749
```

```
## 3  7.467167   -33.68221 102.5             34.5     1749
## 4  6.969356   -31.43673 102.5             32.2     1758
## 5  6.471544   -29.19125 102.5             29.9     1750
## 6  5.973733   -26.94577 102.5             27.6     1745
```

```r
summary(galaxy)
```

```
##    east.west          north.south          angle        radial.position
##  Min.    :-29.66693   Min.    :-49.108   Min.    : 12.50   Min.    :-52.4000
##  1st Qu.: -7.91688   1st Qu.:-13.554   1st Qu.: 63.50   1st Qu.:-21.3500
##  Median : -0.06493   Median :  0.671   Median : 92.50   Median : -0.8000
##  Mean   : -0.33237   Mean   :  1.521   Mean   : 80.89   Mean   : -0.8427
##  3rd Qu.:  6.95053   3rd Qu.: 18.014   3rd Qu.:102.50   3rd Qu.: 19.6500
##  Max.   : 29.48414   Max.   : 49.889   Max.   :133.00   Max.   : 55.7000
##     velocity
##  Min.   :1409
##  1st Qu.:1523
##  Median :1586
##  Mean   :1594
##  3rd Qu.:1669
##  Max.   :1775
```

The data frame contains **323** observations and **5** columns. All entries are numeric. The last column is the velocity we want to predict. The first four variables (columns) are information used to express the relative positions of stars in a galaxy. More information on the dataset can be found here.

Split Data

Let us split the dataset into two parts: Training and testing. **Our target is using the training dataset to make predictions for each observation in the testing dataset.** Here, the output is `velocity` while the inputs are the other 4 columns. First, let us randomly select 200 observations from `galaxy` and store them in a new data frame called `train_data`.

```r
# This produce a random integer vector, containing random integers
# between sequence 1 to 323.
idx <- sample(1:323,200)
print(idx)
```

```
##   [1] 167 129 299 270 187 307  85 277 263  79 213  37 105 217 165 290  89 289
##  [19]  42 111  20  44  70 121  40 172  25 248 198  39 280 160  14 130  45  22
##  [37] 206 230 193 104 255 284 103  13 310 176 110  84  29 141 252 221 108  33
##  [55] 306 149 102 145 118 107  64 224 281  51 272 138 322  43 319  26 143 186
##  [73] 275 152 170  48 245 204 294  24 181 220 214 321 225  83  90 163 256   1
##  [91] 251  78 150 301  28 116 312  61 174 113 195  86  71 316  99 260 302  49
## [109] 311  60 184 216  50 135 304 303 300 250 218  53 144 100 308 296  65 279
## [127] 261 215 124  77  98  19  17 162  31 236 219 239 147  75  16 240   9 211
## [145]  92 122 287 258 292 323 309 140 126 291 228 183  15 201 199 267 191 127
## [163] 133  41 271 117  72  36 212 175 315 257 246 106  88 158 151 164 180  80
## [181]  30  93 166 157 171 232 298 169 202 262 207  73  27 154 273  57  38  62
## [199]   6   4
```

```r
# Your code here for creating train_data here:
train_data <- galaxy[idx, ]
```

Now, select the rest of the observations from `galaxy` and store them in a data frame `test_data`:

```
# Hint
a <- matrix(1:16, 4, 4)

# The following excludes the 1st and 3rd row from a matrix A:
b <- a[-c(1, 3), ]

# Your code for creating test_data here here:
test_data <- galaxy[-idx, ]
```

Now inspect your data frames from the environment pane. Do they have correct dimensions?

Standardize the data

Let us normalize our dataset **inputs** before making predictions.

Write a function which takes one vector `v` as input and returns a standardized vector `s`:

$$s = (v - \mu)/\sigma,$$

where μ is the mean of `v` and σ is the standard deviation of `v`.

```
standardize  <- function(v){
  # Write your code here:
  return ((v - mean(v))/sd(v))
}
```

Now test whether it works on a toy vector (say `1:100`):

```
# Write your code here.
# Hint if the standardization worked then the output vector should have mean
# zero and variance 1
s <- standardize(1:100)
mean(s)
```

```
## [1] 0
```

```
sd(s) # Works!
```

```
## [1] 1
```

Apply the `standardize` function to **the first 4 columns** of `test_data` (the 5th column is the output, we don't want to standardize it!). Assign the result back to `test_data`. Hint: your code should look like `test_data[, 1:4] <- apply(...)`.

```
# Write your code here:
test_data[, 1:4] <- apply(test_data[, 1:4], 2, standardize)
```

Check the mean and standard deviation of each columns of `test_data`. Have they been successfully standardized?

```
# Write your code here:
apply(test_data[ , 1:4], 2, mean)
```

```
##       east.west     north.south           angle radial.position
##    1.797836e-17   -6.854273e-18    1.537178e-16   -2.346813e-17
```

```
apply(test_data[ , 1:4], 2, sd)
```

```
##       east.west     north.south           angle radial.position 
##               1               1               1               1 
```

Similarly, apply the `standardize` function to **the first 4 columns** of `train_data`.

```
# Write your code here:
train_data[, 1:4] <- apply(train_data[,1:4], 2, standardize)
```

Check that it worked:

```
# Write your code here:
apply(train_data[ , 1:4], 2, mean)
```

```
##       east.west     north.south           angle radial.position 
##    3.289063e-17   -9.593021e-18   -1.770741e-16   -6.406551e-18 
```

```
apply(train_data[ , 1:4], 2, sd)
```

```
##       east.west     north.south           angle radial.position 
##               1               1               1               1 
```

Predict!

Let us first create a prediction function. It predicts the output (velocity) given a single input vector. We will use a nearest neighbour approach. Let x be the four-dimensional vector of inputs for a star. Compute the Euclidean distance between x and the input vectors of all stars in the training data. Find the nn (nn should be an integer between 1 and `nrow(train_data)`) stars with the smallest Euclidean distances from x (the neighbours). The predicted velocity should be the average of the neighbours' velocities. **NOTE**: x should not include the 5th column (the velocity!!).

```
predict_vel <- function(x, nn){
  # 1. find the 5 nearest neighbours of x in train_data
  # 2. average the velocities of the 5 nearest neighbours and
  # 3. use that as the predicted velocity
  # YOUR CODE HERE

  # 1) Slow version
  # dist <- c()
  # for(i in 1:nrow(train_data)){
  #   dist[i] <- sqrt(sum((x - train_data[i,-5])^2))
  # }
  # 2) Fast version
  dist <- sqrt(colSums( (t(train_data[ , -5]) - x)^2 ))

  nei <- order(dist)[1:nn]

  pred <- mean(train_data[nei,5])

  return(pred)
}
```

Apply this function to **all the observations** in `test_data`. Store the result to a new column `pred` in `test_data`. Use `nn = 5`.

```
# Hint: recall how to add a new column to a data frame
a <- data.frame(x = c(1,2,3))
# Here we are adding the new column "newcol" to "a"
```

```r
a$newcol <- c(4,5,6)

# Write your code here:
test_data$pred <- apply(test_data[, 1:4], 1, predict_vel, nn = 5)
```

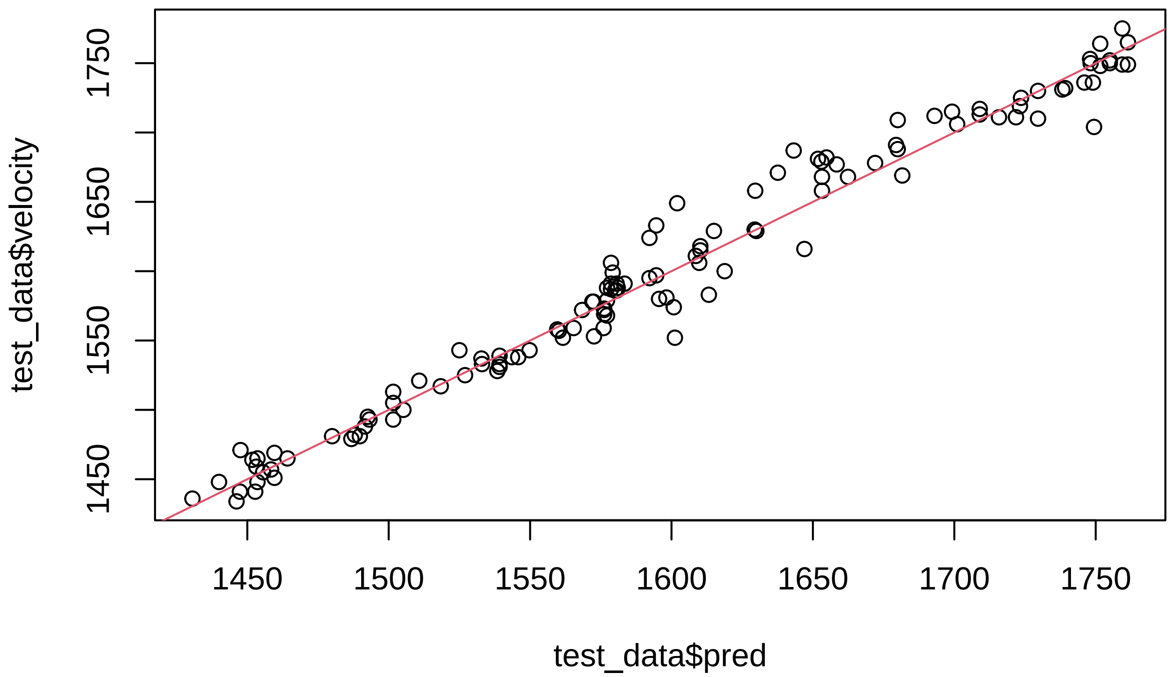
Compute the mean squared error of your predictions. That's a numerical quantification of how good your predictions are (the lower the better).

```r
# Write your code here
mse <- mean((test_data$pred - test_data$velocity)^2)
mse
```

```
## [1] 234.6963
```

Is it any good? Try to visually compare the observed and the predicted velocities:

```r
# Write your code here
plot(test_data$pred, test_data$velocity)
abline(0, 1, col = 2)
```




```r
# Looks quite good!
```

Now consider a naive prediction method: I take the average of the training output `train_data$velocity` , and use it as the prediction of the velocity of the testing dataset. What is the mean squared error of this naive approach?

```r
# Write your code here (one line)
mse_naive <- mean((mean(train_data$velocity) - test_data$velocity)^2)
mse_naive
```

```
## [1] 9092.621
```

How much better our prediction method is than the naive approach in terms of mean squared error?

```r
# Write your code here
mse / mse_naive
```

```
## [1] 0.02581173
```

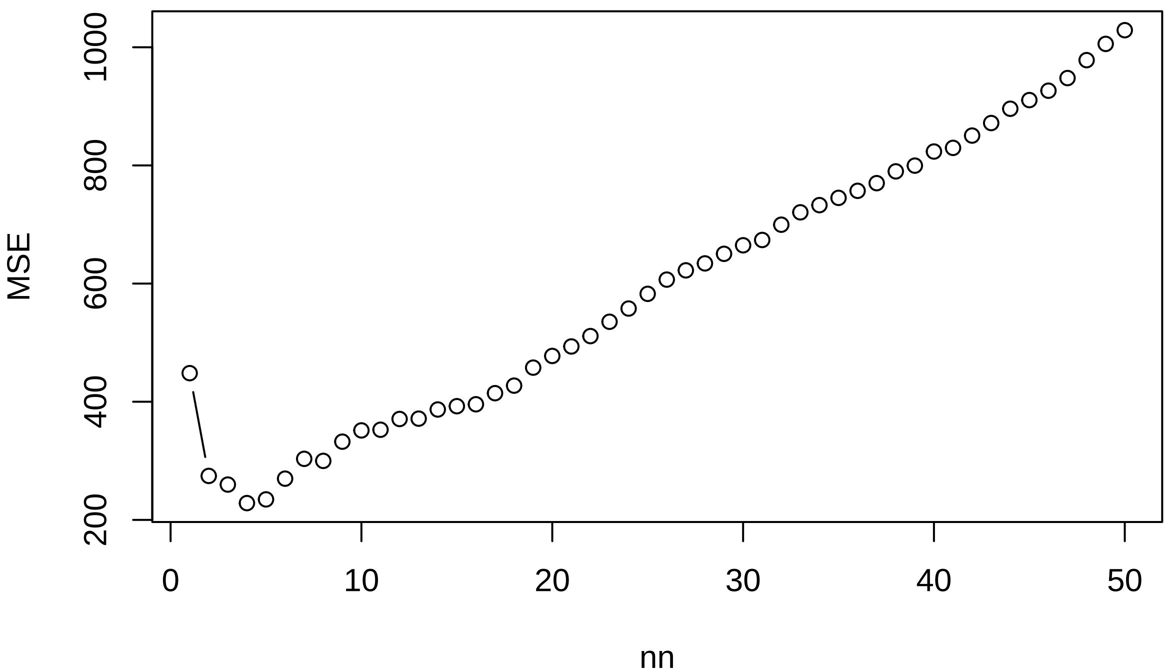
```r
# Much better! The MSE of our model is around 2.6% of the MSE of the naive model
```

Check how the performance of your model varies as you change the number of neighbours `nn`. NOTE: do not define a very fine grid of values for `nn` as the computation might be quite slow.

```r
# Write your code here
my_fun <- function(nn){
  cat(".")
  pred <- apply(test_data[, 1:4], 1, predict_vel, nn = nn)
  mse <-  mean((pred - test_data$velocity)^2)
}
nn_seq <- 1:50
mse_seq <- sapply(nn_seq, my_fun)
```

```
## ..................................................
```

```r
plot(nn_seq, mse_seq, type = 'b', xlab = "nn", ylab = "MSE")
```




```r
# nn = 5 was a pretty decent value!
```