Tutorial 5

ST2137-2420

Material

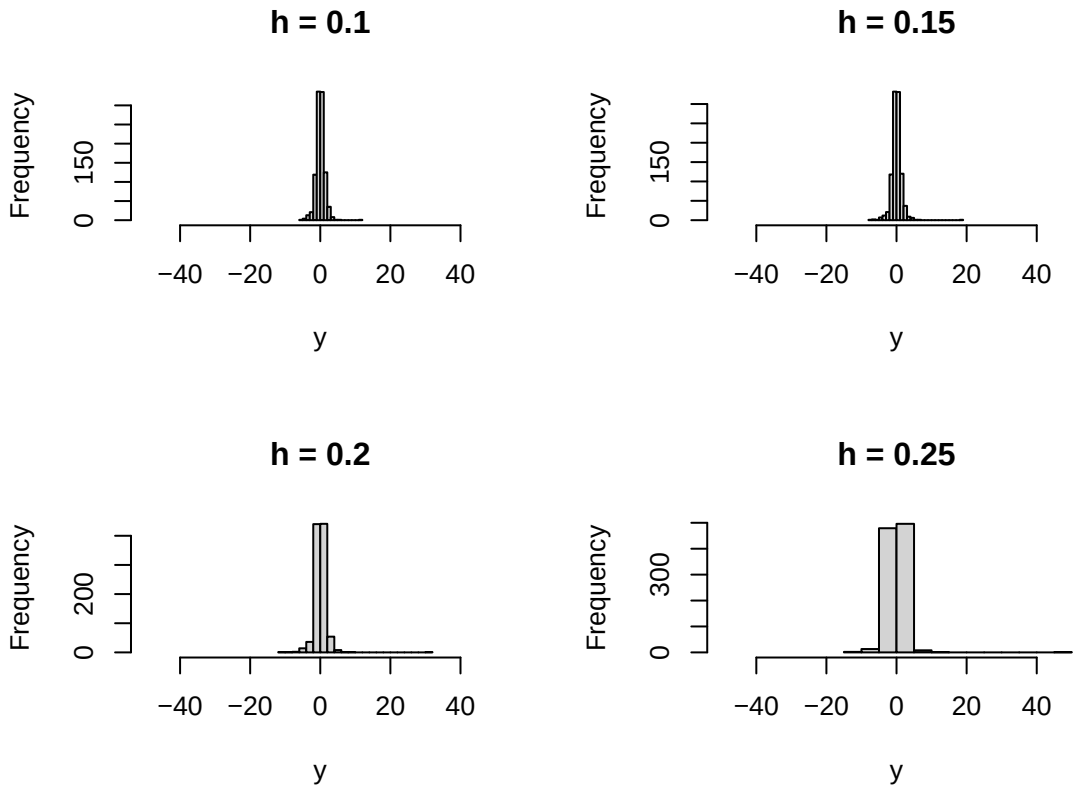
This tutorial covers the topics and concepts from chapter 5 of the course textbook: robust statistics. The first question is for you to grasp the value of robust statistics. It emphasises that we can use a computer to understand/test new methodologies out before diving deeper into them.

h-Distributions

Let us define a family of distributions that will enable us to study the value of robust statistics, in terms of dealing with distributions with fat tails. If $Z \sim N(0,1)$, then for a fixed h, define the random variable Y:

$$Y = Ze^{hZ^2/2}$$

Here are some sample histograms of observations for various values of h:



As we can see, the value of h can be used to "control" the amount of elongation in the tails, leading to very large observations.

1. Perform the following simulation experiment:

A. Repeat 50 times:
   1. Generate 30 observations from $h = 0.3$.
   2. Compute the sample mean and the trimmed mean.

B. Compute the average and s.d. of the 50 sample means and trimmed means.

What is your view/opinion of trimmed mean vs. mean in this case?

R code

```
set.seed(2138)
z <- matrix(rnorm(50*30), nrow=50)
h <- 0.3
generated_y <- z*exp(h*z*z/2)

x_bar <- apply(generated_y, 1, mean)
x_trim <- apply(generated_y, 1, mean, trim=0.1)

print(mean(x_bar))
```

```
[1] -0.0742567
```

```
print(sd(x_bar))
```

```
[1] 0.4348214
```

```
print(mean(x_trim))
```

```
[1] -0.0324937
```

```
print(sd(x_trim))
```

```
[1] 0.2869183
```

Python code

```
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

rng = np.random.default_rng(2138)      # to initialise the pseudo-random number generator
Z = rng.normal(0, 1, (50,30))          # generates 1000 N(0,1) random variables

h = 0.3
generated_y = Z*np.exp(h*(Z**2)/2)

y_bar = generated_y.mean(axis=1)
print(f"The (untrimmed) mean is {y_bar.mean():.3f} and the sd is {y_bar.std():.3f}.")
```

```
The (untrimmed) mean is -0.003 and the sd is 0.374.
```

```
y_trim  = np.apply_along_axis(stats.trim_mean, 1, generated_y, proportiontocut=0.1)
print(f"The trimmed mean is {y_trim.mean():.3f} and the sd is {y_trim.std():.3f}.")
```

```
The trimmed mean is -0.008 and the sd is 0.233.
```

Outlier Detection

The following three methods are sometimes used to detect outliers:

Rule based on means and variances:

The rule is to declare an outlier if

$$\frac{|X_i - \bar{X}|}{s} > K$$

Usually, K is taken to be 2.24. If X_i was truly Normal, 2.5% of observations would be classified as outliers, on average.

Rule based on IQR

The boxplot uses the following rule.

$$X_i < q_{0.25} - 1.5 \times IQR(X), \text{ or } X_i > q_{0.75} + 1.5 \times IQR(X)$$

where q_1 and q_3 are the sample quartiles.

Rule based on median and MAD(X):

X_i is declared an outlier if

$$\frac{|X_i - \text{median}(X)|}{MAD(X)/0.6745} > K$$

In this K is taken to be the square root of the 0.975 quantile of a χ^2 distribution. It is approximately 2.24 once again.

2. Use the above three methods to detect the outliers in the following dataset:

$$2, 2, 3, 3, 3, 4, 4, 4, 100000, 100000$$

R code

```
d1 <- c(2, 2, 3, 3, 3, 4, 4, 4, 100000, 100000)

# mtd 1
which(abs(d1 - mean(d1))/sd(d1) > 2.24)
```

```
integer(0)
```

```
# mtd 2
out <- boxplot(d1, plot=FALSE)
out$out
```

```
[1] 1e+05 1e+05
```

```
# mtd 3
which(0.6745 * abs(d1 - median(d1))/mad(d1) > 2.24)
```

```
[1]  9 10
```

Python code

```
import pandas as pd
import numpy as np
from scipy import stats

import matplotlib.pyplot as plt
```

```python
d1 = np.array([2, 2, 3, 3, 3, 4, 4, 4, 100000, 100000])

# mtd 1
np.where(np.absolute(d1 - d1.mean())/d1.std() > 2.24)
```

```
(array([], dtype=int64),)
```

```python
# mtd 2
iqr_x = stats.iqr(d1)
lower, upper = np.quantile(d1, [0.25, 0.75])

np.where((d1 > upper+1.5*iqr_x) | (d1 < lower  - 1.5*iqr_x) )
```

```
(array([8, 9]),)
```

```python
# mtd 3
np.where(.6745 * np.absolute(d1 - np.quantile(d1, 0.5))/
stats.median_abs_deviation(d1) > 2.24)
```

```
(array([8, 9]),)
```

The issue with the first method is that it uses non-robust estimates of location and scale. As a result, the outliers are masked, since they have influenced the estimates of the location and scale.

Student Performance Dataset

3. Use the Winsorized and trimmed means with $\gamma = 0.1$ to estimate the location parameter for `G3`, grouped by `Medu`. Compare these estimates of location to the usual sample mean. Why are the estimates different? Which would you use?

In this first section, we demonstrate how we can compute the robust statistics in a manual manner.

R code

```r
library(DescTools)
stud_perf <- read.csv('../data/student/student-mat.csv', sep=";")

t_mean <- aggregate(G3 ~ Medu, data = stud_perf, FUN = mean, trim=0.1)
w_mean <- aggregate(G3 ~ Medu, data = stud_perf, FUN = function(x) mean(Winsorize(x)))
r_mean <- aggregate(G3 ~ Medu, data = stud_perf, FUN = mean)

all_means <- cbind(r_mean, t_mean[,2], w_mean[,2])
colnames(all_means) <- c("Medu", "raw_mean", "trim_mean", "win_mean")
all_means
```

```
  Medu  raw_mean trim_mean  win_mean
1    0 13.000000 13.000000 13.200000
2    1  8.677966  8.938776  8.598305
3    2  9.728155 10.120482  9.611650
4    3 10.303030 10.629630 10.292929
5    4 11.763359 12.076190 11.935115
```

Python code

```python
# Python
stud_perf  = pd.read_csv("../data/student/student-mat.csv", delimiter=";")

all_means = []
for x,y in stud_perf.groupby('Medu'):
```

```
    tmp = y.G3.copy()
    r_mean = tmp.mean()
    t_mean = stats.trim_mean(tmp, 0.1)
    w_mean = stats.mstats.winsorize(tmp, limits=0.1).mean()
    all_means.append((x, r_mean, t_mean, w_mean))

pd.DataFrame.from_records(all_means, columns=['Medu', 'r_mean', 't_mean', 'w_mean'])
```

```
   Medu     r_mean     t_mean     w_mean
0     0  13.000000  13.000000  13.000000
1     1   8.677966   8.938776   8.525424
2     2   9.728155  10.120482   9.514563
3     3  10.303030  10.629630  10.606061
4     4  11.763359  12.076190  11.961832
```

R code

```
temp_fn <- function(x) {
  c(
    "raw"=mean(x),
    "trim"=mean(x, trim=0.1),
    "win"=mean(Winsorize(x))
    )
}
aggregate(G3 ~ Medu, data = stud_perf, FUN = temp_fn)
```

```
  Medu    G3.raw   G3.trim    G3.win
1    0 13.000000 13.000000 13.200000
2    1  8.677966  8.938776  8.598305
3    2  9.728155 10.120482  9.611650
4    3 10.303030 10.629630 10.292929
5    4 11.763359 12.076190 11.935115
```

Python code

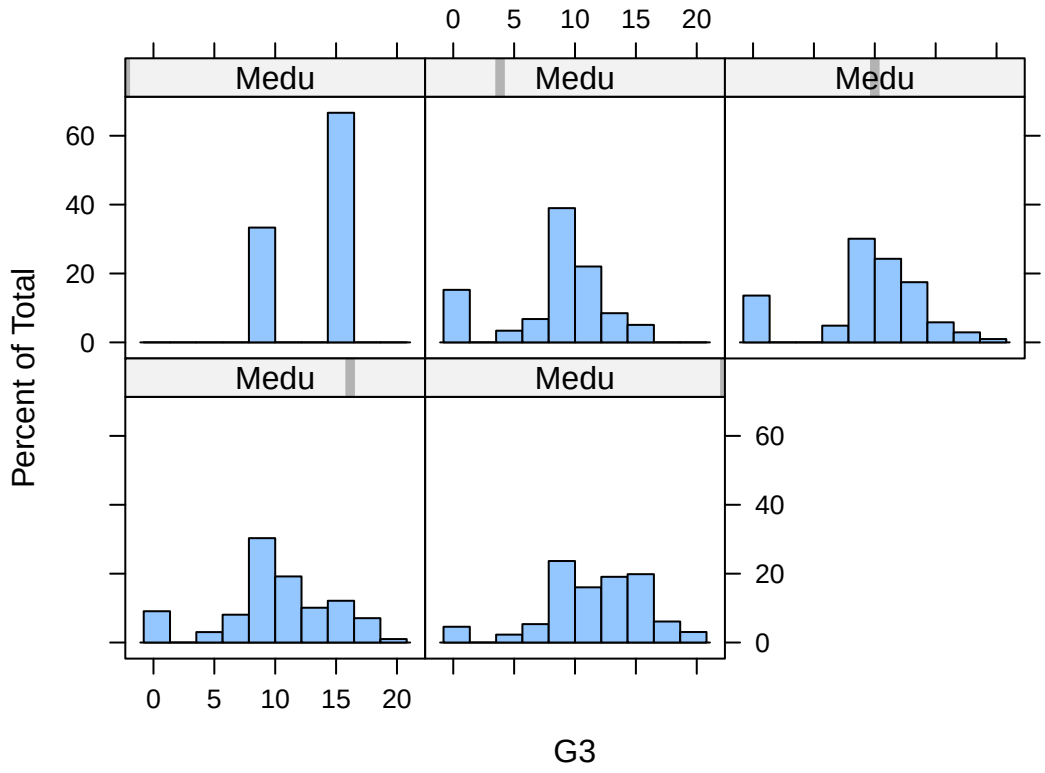
```
def compute_means(group):
    tmp = group.G3.copy()
    return pd.Series({
        'r_mean': tmp.mean(),
        't_mean': stats.trim_mean(tmp, 0.1),
        'w_mean': stats.mstats.winsorize(tmp, limits=0.1).mean()
    })

df_means = stud_perf.groupby('Medu')[['G3']].apply(compute_means)
df_means
```

```
          r_mean     t_mean     w_mean
Medu
0      13.000000  13.000000  13.000000
1       8.677966   8.938776   8.525424
2       9.728155  10.120482   9.514563
3      10.303030  10.629630  10.606061
4      11.763359  12.076190  11.961832
```

4. Use the three techniques in the previous sections to identify any potential outliers in `G3`, grouped by `Medu`.

Recall that there were several students with 0 scores for G3:

```
library(lattice)
histogram(~G3 | Medu, data=stud_perf, as.table = TRUE)
```



Once again, it is useful to write a function to compute the outliers according to each rule, and then use the apply function to extract them. We demonstrate in R..

```
return_outliers <- function(x) {
  # mtd 1
  id <- which(abs(x - mean(x))/sd(x) > 2.24)
  mtd_1 <- x[id]

  # mtd 2
  out <- boxplot(x, plot=FALSE)
  mtd_2 <- out$out

  # mtd 3
  id <- which(0.6745 * abs(x - median(x))/mad(x) > 2.24)
  mtd_3 <- x[id]

  list("method 1" = mtd_1, "method 2" = mtd_2, "method 3"= mtd_3)
}
```

Here, it is worthwhile to demonstrate yet another member of the `apply` family from R – `tapply`. This latter function works similar to `aggregate`, but works better when the output cannot be packed neatly into an array. Since the number of outliers varies by group and by method of outlier detection used, it is best to keep things as a list:

```
tapply(stud_perf$G3, stud_perf$Medu, return_outliers)
```

```
$`0`
$`0`$`method 1`
integer(0)

$`0`$`method 2`
numeric(0)

$`0`$`method 3`
[1] 9


$`1`
$`1`$`method 1`
integer(0)

$`1`$`method 2`
[1] 0 0 0 0 0 0 0 0 0

$`1`$`method 3`
[1] 0 0 0 0 0 0 0 0 0


$`2`
$`2`$`method 1`
integer(0)

$`2`$`method 2`
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0

$`2`$`method 3`
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0


$`3`
$`3`$`method 1`
integer(0)

$`3`$`method 2`
[1] 0 0 0 0 0 0 0 0 0

$`3`$`method 3`
integer(0)


$`4`
$`4`$`method 1`
[1] 0 0 0 0 0 0

$`4`$`method 2`
[1] 0 0 0 0 0 0

$`4`$`method 3`
integer(0)
```

The output indicates that for each group, when outliers are present, they correspond to the zero values. For an analyst working on this data, it would be prudent to study if the group of zero-valued observations should be studied as a separate sub-group, or if they should be included with the rest.