MALProject

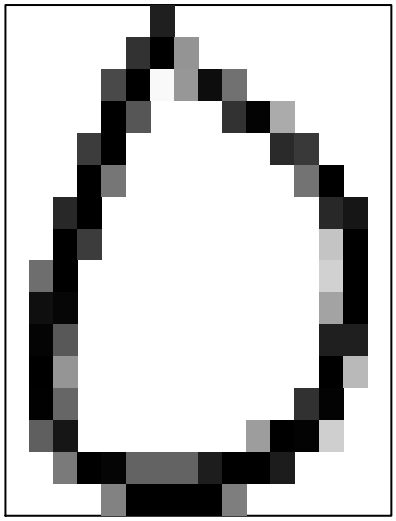
Frederick Deny - Paul Canat

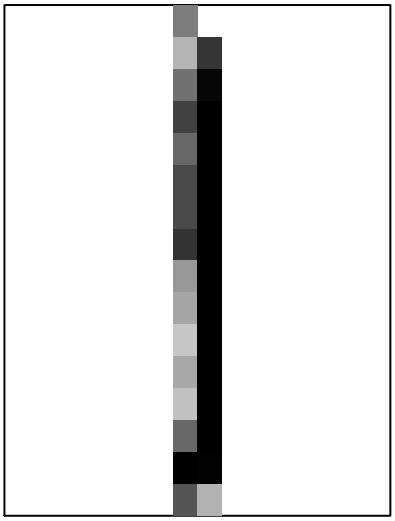
Chosen digits

We chosen the first two digits to recognise in our project. We felt that those two digits were relatively simple in concept: a circle and a straight line, but completely different from one another.

```
## [1] "digit  0  taken"
```

```
## [1] "digit  1  taken"
```





Random forest

With filtered training data

We first understood that we had to choose two digits to recognise so that the random forest algorithm could be computed faster than if he had to train the model for all digits. Consequently we filtered the training data to only keep the rows with “0” and “1” labels, as well as the testing data.

```
## -- Attaching packages ------------------------------------------------------------------- tidyverse
## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ---------------------------------------------------------------------- tidyverse_confl
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

We then applied the random forest classifier to build our model with our filtered data, and build a matrix showing our results:

```
##    evaluation_tr_filt_te_filt
## Yte   0   1
##   0 359   0
##   1   0 264
```

We see that in this case, our model estimated perfectly which images represented the digits.

NB: This isn't a confusion matrix: but shows that each zero digits recognised is indeed a zero and the same for the ones.

Without filtered training data

Seeing theses results, we understood we couldn't better theses results: the experiment had limited teachings, so we decided to keep the training data unfiltered, believing we missunderstood the instructions of the project.

```
##    evaluation_te_filt
## Yte   0   1   2
##   0 341  17   1
##   1   0 260   4

##      evaluation_0
## Yte_0   0   2
##     0 341  18
##     2   0 264

##      evaluation_1
## Yte_1   1   2
##     1 260   4
##     2  17 342
```

With the training data unfiltered we see that the classifier got some digits wrong, so we made another recap array with the value "2" representing a "neither 0 or 1 has been detected but another digit".

To allow the construction of the confusion matrixes, we made two different evaluation of the results, one for each of the two digits.

Those matrixes use the digit to guess as positive and "2" as negative,

Altogether we have satsifactory results, but we tried the same experiment but with twice as many trees.

```
##    evaluation_te_filt
## Yte   0   1   2
##   0 346  13   0
##   1   0 262   2
```

```
##      evaluation_0
## Yte_0   0   2
##     0 346  13
##     2   0 264
```

```
##      evaluation_1
## Yte_1   1   2
##     1 262   2
##     2  13 346
```

We see that by increasing the number of trees we enhance the scoring of the built model.

Unfiltered data

```
##    evaluation_te_filt
## Yte   0   1   2
##   0 346  13   0
##   1   0 262   2
##   2   0   0 198
##   3   0   0 166
##   4   0   0 200
##   5   0   0 160
##   6   0   0 170
##   7   0   0 147
##   8   0   0 166
##   9   0   0 177
```

```
##      evaluation_0
## Yte_0    0    2
##     0  346   13
##     2    0 1648
```

```
##      evaluation_1
## Yte_1    1    2
##     1  262    2
##     2   13 1730
```

It is important to test the built model with unfiltered test data to insure the model doesn't take another digit for a one or a zero.

Bagging

Unfiltered data

In retrospect, we found that using unfiltered data produced more information so decided to work with boosting with all our data.

```
##    evaluation_b
## Yte   0   2
##   0 359   0
##   1 264   0
##   2   0 198
##   3   0 166
##   4   0 200
##   5   0 160
##   6   0 170
##   7   0 147
##   8   0 166
##   9   0 177
```

```
##      evaluation_0
## Yte_0    0    2
##     0  359    0
##     2  264 1384
```

```
##      evaluation_1
## Yte_1    2
##     1  264
##     2 1743
```

We see here that bagging isn't powerfull in recognising ones, as he can't seem de detect as little as one of them, and moreover detects all of them as zeroes.

We did the test again while checking it recognises other digits, to see if it was a localised issue:

```
##    evaluation_b
## Yte   0   2   3   6   7   9
##   0 359   0   0   0   0   0
##   1 264   0   0   0   0   0
##   2   0 198   0   0   0   0
##   3   0   0 166   0   0   0
##   4   0   0 200   0   0   0
##   5   0   0   0 160   0   0
##   6   0   0   0 170   0   0
##   7   0   0   0   0 147   0
##   8   0   0   0   0   0 166
##   9   0   0   0   0   0 177
```

```
##      evaluation_0
## Yte_0    0    2
##     0  359    0
##     2  264 1384
```

```
##      evaluation_1
## Yte_1    2
##     1  264
##     2 1743
```

It seems that the issus is restricted to 1, 5 and 8, which bagging mistakes for 0, 6 and 9.

More bags

As bagging previously showed no prowess we tried again with more bootstrap replications, 100 whereas we precedently only did 25.

```
##    evaluation_b
## Yte   0   2   3   4   6   7   9
##   0 359   0   0   0   0   0   0
##   1 264   0   0   0   0   0   0
##   2   0 198   0   0   0   0   0
##   3   0   0 166   0   0   0   0
##   4   0   0   0 200   0   0   0
##   5   0   0   0   0 160   0   0
##   6   0   0   0   0 170   0   0
##   7   0   0   0   0   0 147   0
##   8   0   0   0   0   0   0 166
##   9   0   0   0   0   0   0 177
```

```
##      evaluation_0
## Yte_0    0    2
##     0  359    0
##     2  264 1384
```

```
##      evaluation_1
## Yte_1    2
##     1  264
##     2 1743
```

With more bootstrap replication the model does no better than before: it seems that bagging isn't efficient in this case.

Boosting

```
##                Observed Class
## Predicted Class   0   1
##               0 338   1
##               1   1 252
##               2   2   0
##               3   3   0
##               4   2   4
##               5   1   1
##               6   2   3
##               7   0   1
##               8   9   0
##               9   1   2
```

We see that bagging works in a satisfactory manner, but shows a lot of residual error.

We will try by making boosting use more iterations:

```
##                 Observed Class
## Predicted Class   0   1
##               0 337   1
##               1   1 252
##               2   3   0
##               4   2   4
##               5   2   1
##               6   2   1
##               7   0   2
##               8  11   3
##               9   1   0
```

Even with twice as many iterations, the model gives very similar results, showing that we are at the limit of this model to distinguish digits.

Conclusion

It seems that between the three classification methods that we tested during this study, the random forest is the most powerfull one: in less working time than boosting it was more precise, while bagging was unsurprisingly a weak predictor.