Machine Learning Research log

Can machine learning predict protein localization based on their aminoacid sequences

John Busker
352905
Bio-Informatics
Dave Langers, and Bart Barnard
28-10-2022

Table of Contents

R Environment

Multiple libraries have been uses for data analysis and further processes. The libraries used are:

```r
library(tidyr)
library(kableExtra)
library(ggplot2)
library(gridExtra)
library(cowplot)
library(ggpubr)
library(pander)
library(ggbiplot)
library(reshape2)
library(ggfortify)
library(ggalt)
library(ggridges)
library(cluster)
```

EDA

About The Data

The data has been retrieved from uci. The dataset consists of 1484 yeast sequences from SWISS-PROT using the annotations from YPD. In this section, we are going to talk about the results we found from the dataset. Eight features were used in classification: the presence or absence of an HDEL pattern as a signal for retention in the endoplasmic reticulum lumen (erl); The results of discriminant analysis on the amino acid content of vacuolar and extracellular proteins (vac); the result of discriminant analysis on the amino acid composition of the 20-residue N-terminal region of mitochondrial and non-mitochondrial proteins (mit); the presence or absence of nuclear localization consensus patterns combined with a term reflecting the frequency of basic residues (nuc); and some combination of the presence of a short sequence motif and the result of discriminant analysis of the amino acid composition of the protein sequence (pox).

Attributes information:

- **Sequence Name**: Accession number for the SWISS-PROT database.
- **mcg**: McGeoch's method for signal sequence recognition.
- **gvh**: von Heijne's method for signal sequence recognition.
- **alm**: Score of the ALOM membrane spanning region prediction program.
- **mit**: Score of discriminant analysis of the amino acid content of the N-terminal region (20 residues long) of mitochondrial and non-mitochondrial proteins.
- **erl**: Presence of "HDEL" substring (thought to act as a signal for retention in the endoplasmic reticulum lumen). Binary attribute.
- **pox**: Peroxisomal targeting signal in the C-terminus.
- **vac**: Score of discriminant analysis of the amino acid content of vacuolar and extracellular proteins.
- **nuc**: Score of discriminant analysis of nuclear localization signals of nuclear and non-nuclear proteins.
- **Class Distribution**: The class is the localization site. Consisting of (abbreviation (full name) the amount):

CYT	(cytosolic or cytoskeletal)	463
NUC	(nuclear)	429
MIT	(mitochondrial)	244
ME3	(membrane protein, no N-terminal signal)	163
ME2	(membrane protein, uncleaved signal)	51
ME1	(membrane protein, cleaved signal)	44
EXC	(extracellular)	37
VAC	(vacuolar)	30
POX	(peroxisomal)	20
ERL	(endoplasmic reticulum lumen)	5

Yeast proteins were classified into ten classes: **cytoplasmic:** cytoskeletal (CYT); nuc]ear (NUC); vacuolar (VAC); mitochondrial (MIT); isomal (POX); **extracellular:** including those localized against the cell wall (EXC); proteins localized to the lumen of the endoplasmic reticulum (ERL); membrane proteins with a cleaved signal (ME1); membrane proteins with an uncleared signal (ME2); and membrane proteins with no

N-terminal sign (ME3), where ME1, ME2,and ME3 proteins may be localized to the plasma membrane, the endoplasmic reticulum membrane, or the membrane of a golgi body.

Codebook

Since there is not a codebook. A own codebook has been created

```
# All the attributes name
attr_names <- c("seq.name", "mcg", "gvh", "alm", "mit", "erl",
                "pox", "vac","nuc", "loc.site" )

data_types <- c("str", "float", "float", "float", "float", "float",
                "bool", "float", "float", "factor")

# The description of the labels
data_labels <- c("Accession number for the SWISS-PROT database",
                 "McGeoch's method for signal sequence recognition",
                 "von Heijne's method for signal sequence recognition",
                 "Score of the ALOM membrane spanning region prediction program",
                 "Score of discriminant analysis of the amino
                 acid content of the N-terminal region",
                 "Presence of 'HDEL' substring",
                 "Peroxisomal targeting signal in the C-terminus",
                 "Score of discriminant analysis of the amino acid content
                 of vacuolar and extracellular proteins",
                 "Score of discriminant analysis of nuclear
                 localization signals of nuclear and non-nuclear proteins",
                 "The class is the localization site")

codebook <- data.frame(Name=attr_names,
                       Fullname=data_labels,
                       Datatypes=data_types)

pander(codebook)
```

Name	Fullname	Datatypes
seq.name	Accession number for the SWISS-PROT database	str
mcg	McGeoch’s method for signal sequence recognition	float
gvh	von Heijne’s method for signal sequence recognition	float
alm	Score of the ALOM membrane spanning region prediction program	float
mit	Score of discriminant analysis of the amino acid content of the N-terminal region	float
erl	Presence of ‘HDEL’ substring	float
pox	Peroxisomal targeting signal in the C-terminus	bool
vac	Score of discriminant analysis of the amino acid content of vacuolar and extracellular proteins	float
nuc	Score of discriminant analysis of nuclear localization signals of nuclear and non-nuclear proteins	float

Name	Fullname	Datatypes
loc.site	The class is the localization site	factor

Here is the codebook. With the attributes abbreviation, explanation and data types. As you can see there are many float datatypes.

Loading The Data

Now we need to load in the data. And change the column names to the attributes abbreviation, since these are non-existent. Let's give all the columns a name. And let's take a quick look at the data.

```r
# Read the file in ass a tibble
data <- as_tibble(read.table("Resources/yeast.data", sep = ""))
colnames(data) <- attr_names
head(data)
```

```
# A tibble: 6 x 10
  seq.name     mcg   gvh   alm   mit   erl   pox   vac   nuc loc.site
  <chr>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
1 ADT1_YEAST  0.58  0.61  0.47  0.13   0.5   0    0.48  0.22 MIT
2 ADT2_YEAST  0.43  0.67  0.48  0.27   0.5   0    0.53  0.22 MIT
3 ADT3_YEAST  0.64  0.62  0.49  0.15   0.5   0    0.53  0.22 MIT
4 AAR2_YEAST  0.58  0.44  0.57  0.13   0.5   0    0.54  0.22 NUC
5 AATM_YEAST  0.42  0.44  0.48  0.54   0.5   0    0.48  0.22 MIT
6 AATC_YEAST  0.51  0.4   0.56  0.17   0.5   0.5  0.49  0.22 CYT
```

The yeast data set contains scores per cellular localization sites. The last column is the sequence's localization site. There are ten different possibilities for this.

Cleaning The Data

We can drop the first columns since it is not necessary. Since the sequence names contribute nothing to create a prediction model.

```r
# Drop the first column
data <- data[, -1]
head(data)
```

```
# A tibble: 6 x 9
    mcg   gvh   alm   mit   erl   pox   vac   nuc loc.site
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
1  0.58  0.61  0.47  0.13   0.5   0    0.48  0.22 MIT
2  0.43  0.67  0.48  0.27   0.5   0    0.53  0.22 MIT
3  0.64  0.62  0.49  0.15   0.5   0    0.53  0.22 MIT
4  0.58  0.44  0.57  0.13   0.5   0    0.54  0.22 NUC
5  0.42  0.44  0.48  0.54   0.5   0    0.48  0.22 MIT
6  0.51  0.4   0.56  0.17   0.5   0.5  0.49  0.22 CYT
```

The first column has been succesfully dropped from the data.

Are there any missing values? Let's take a quick look.

```r
# Count all the NA values
sum(is.na(data))
```

```
[1] 0
```

There are not any missing values. So nothing needs to be changed for this.

Now we need to take a clearer look at the data set using `str()`.

```r
str(data)
```

```
tibble [1,484 x 9] (S3: tbl_df/tbl/data.frame)
 $ mcg     : num [1:1484] 0.58 0.43 0.64 0.58 0.42 0.51 0.5 0.48 0.55 0.4 ...
 $ gvh     : num [1:1484] 0.61 0.67 0.62 0.44 0.44 0.4 0.54 0.45 0.5 0.39 ...
 $ alm     : num [1:1484] 0.47 0.48 0.49 0.57 0.48 0.56 0.48 0.59 0.66 0.6 ...
 $ mit     : num [1:1484] 0.13 0.27 0.15 0.13 0.54 0.17 0.65 0.2 0.36 0.15 ...
 $ erl     : num [1:1484] 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
 $ pox     : num [1:1484] 0 0 0 0 0 0.5 0 0 0 0 ...
 $ vac     : num [1:1484] 0.48 0.53 0.53 0.54 0.48 0.49 0.53 0.58 0.49 0.58 ...
 $ nuc     : num [1:1484] 0.22 0.22 0.22 0.22 0.22 0.22 0.22 0.34 0.22 0.3 ...
 $ loc.site: chr [1:1484] "MIT" "MIT" "MIT" "NUC" ...
```

As you can see, the last column, loc.site, consists of characters, but this one needs to be converted to factors for clarity and complexity.

```r
# Transform the last column to a factor
data$loc.site <- as.factor(data$loc.site)
str(data$loc.site)
```

```
 Factor w/ 10 levels "CYT","ERL","EXC",..: 7 7 7 8 7 1 7 8 7 1 ...
```

As you can see the data type of the column loc.site has been succesfully changed to factors.

There are alot of rows. Let's visualize the amount of each classification.

```r
# A grid to fill in 28x53
df <- expand.grid(y = 1:28, x = 1:53)
# Sort the table
categ_table <- sort(table(data$loc.site), decreasing = T)
df$category <- factor(rep(names(categ_table), categ_table))

ggplot(df, aes(x = x, y = y, fill = category)) +
        geom_tile(color = "black", size = 0.5) +
        scale_x_continuous(expand = c(0, 0)) +
        scale_y_continuous(expand = c(0, 0), trans = 'reverse') +
        scale_fill_brewer(palette = "Set3") +
        labs(title="Waffle Chart", subtitle="'Class' of localization") +
        xlab(NULL) + ylab(NULL)
```

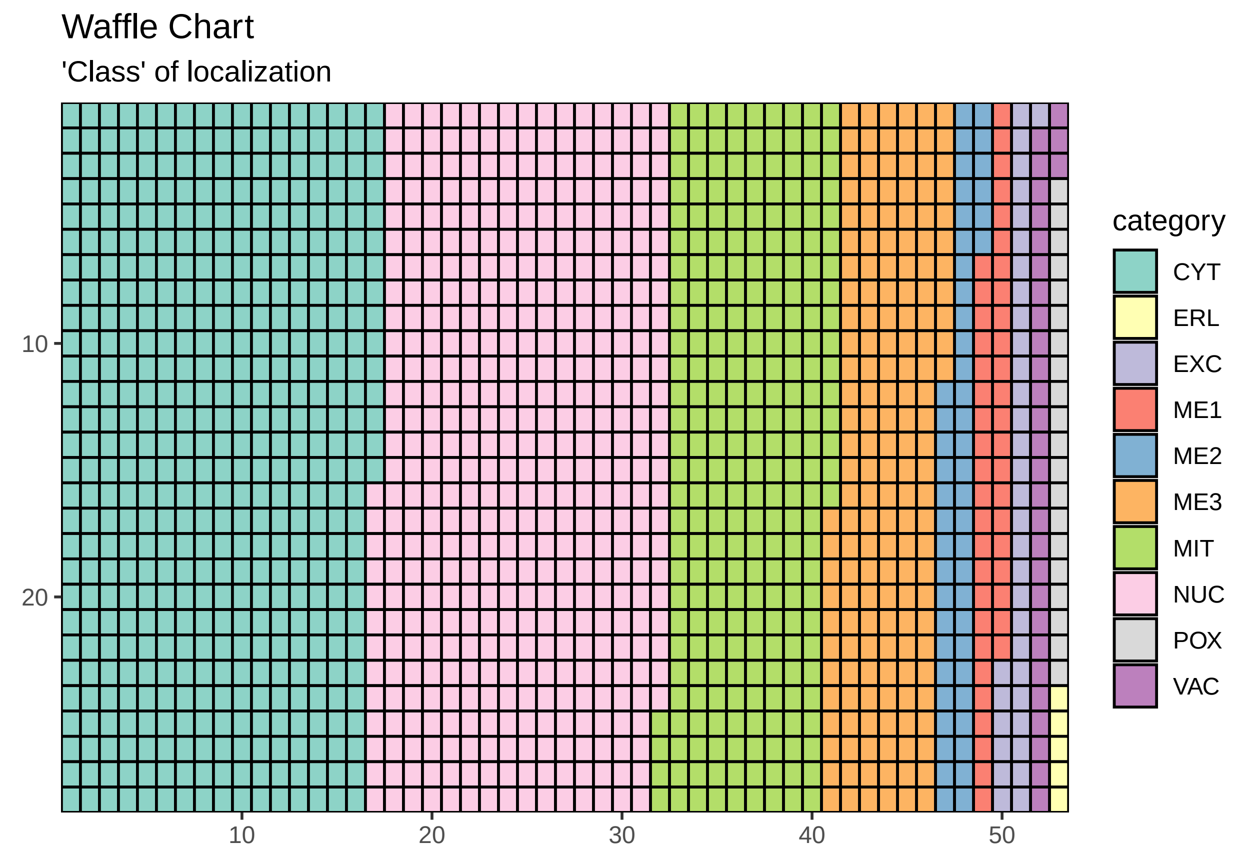


Figure 1: A waffle chart of the categorical composition

There are alot of CYT and NUC localization. This probably means that CYT has a greater variation in localisation than ERL.

Data Transformation

The ERL variable also should be changed. As you can see, it is now a num data type but I needs to be a bool/binary datatype. Let's take a closer look at th ERL column.

```r
summary(data$erl)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.5000  0.5000  0.5000  0.5047  0.5000  1.0000 
```

This looks weird because the value should be 0 or 1.

There are alot of 0.5 values. But should not exist in this column. The variable need to be a bool, or in other words it must be a 0 or a 1. So all the 0.5 values needs to be changed to a 0.

```r
table(data[, "erl"])
```

```

 0.5    1 
1470   14 
```

Before the data transformation there are 1470 counts of the value 0.5 and 14 of 1.

```r
# Change every 0.5 value to a 0
data$erl[data$erl == 0.5] <- 0
table(data[,"erl"])
```

```

   0    1 
1470   14 
```

The data has been succesfully transformed. Now the datatype has to be changed to a bool.

```r
# Change the datatype to a logical
data$erl <- as.logical(data$erl)
str(data$erl)
```

```
 logi [1:1484] FALSE FALSE FALSE FALSE FALSE FALSE ...
```

Now every column has the right datatype.

Univariate Analysis

Let's take a quick look at the data using `summary()`. We can drop column 5 and 9 for this. Column 5, ERL, is a logical datatype and column 9, loc.site, is a string datatype.

```r
summary(data[, -c(5,9)])
```

```
      mcg              gvh              alm             mit        
 Min.   :0.1100   Min.   :0.1300   Min.   :0.21   Min.   :0.0000  
 1st Qu.:0.4100   1st Qu.:0.4200   1st Qu.:0.46   1st Qu.:0.1700  
 Median :0.4900   Median :0.4900   Median :0.51   Median :0.2200  
 Mean   :0.5001   Mean   :0.4999   Mean   :0.50   Mean   :0.2612  
 3rd Qu.:0.5800   3rd Qu.:0.5700   3rd Qu.:0.55   3rd Qu.:0.3200  
 Max.   :1.0000   Max.   :1.0000   Max.   :1.00   Max.   :1.0000  
      pox              vac              nuc        
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
 1st Qu.:0.0000   1st Qu.:0.4800   1st Qu.:0.2200  
 Median :0.0000   Median :0.5100   Median :0.2200
 Mean   :0.0075   Mean   :0.4999   Mean   :0.2762
 3rd Qu.:0.0000   3rd Qu.:0.5300   3rd Qu.:0.3000
 Max.   :0.8300   Max.   :0.7300   Max.   :1.0000
```

As you can see all the datapoints are between 0 and 1. Se the data already has been transformed with a min-max normalization.

Let's visualise this with ggplot. Using jitterpoints and a violing plot.

```
p1 <- ggplot(data, mapping = aes(x = "", y = mcg)) + geom_violin(alpha=0.2) +
  geom_jitter(width = 0.2, alpha = 0.25, height = 0, color = "red") + xlab(NULL)
p2 <- ggplot(data, mapping = aes(x = "", y = gvh)) + geom_violin(alpha=0.2) +
  geom_jitter(width = 0.2, alpha = 0.25, height = 0, color = "blue") + xlab(NULL)
p3 <- ggplot(data, mapping = aes(x = "", y = alm)) + geom_violin(alpha=0.2) +
  geom_jitter(width = 0.2,alpha = 0.25, height = 0, color = "purple") + xlab(NULL)
p4 <- ggplot(data, mapping = aes(x = "", y = mit)) + geom_violin(alpha=0.2) +
  geom_jitter(width = 0.2, alpha = 0.25, height = 0, color = "brown") + xlab(NULL)
p5 <- ggplot(data, mapping = aes(x = "", y = pox)) + geom_violin(alpha=0.2) +
  geom_jitter(width = 0.2,alpha = 0.25, height = 0, color = "orange") + xlab(NULL)
p6 <- ggplot(data, mapping = aes(x = "", y = vac)) + geom_violin(alpha=0.2) +
  geom_jitter(width = 0.2, alpha = 0.25, height = 0, color = "green") + xlab(NULL)
p7 <- ggplot(data, mapping = aes(x = "", y = nuc)) + geom_violin(alpha=0.2) +
  geom_jitter(width = 0.2,alpha = 0.25, height = 0, color = "orange") + xlab(NULL)

plot <- ggarrange(p1, p2, p3, p4, p5, p6, p7, nrow = 4, ncol = 2)
annotate_figure(plot, top = text_grob("Boxplots", face = "bold", size = 14))
```

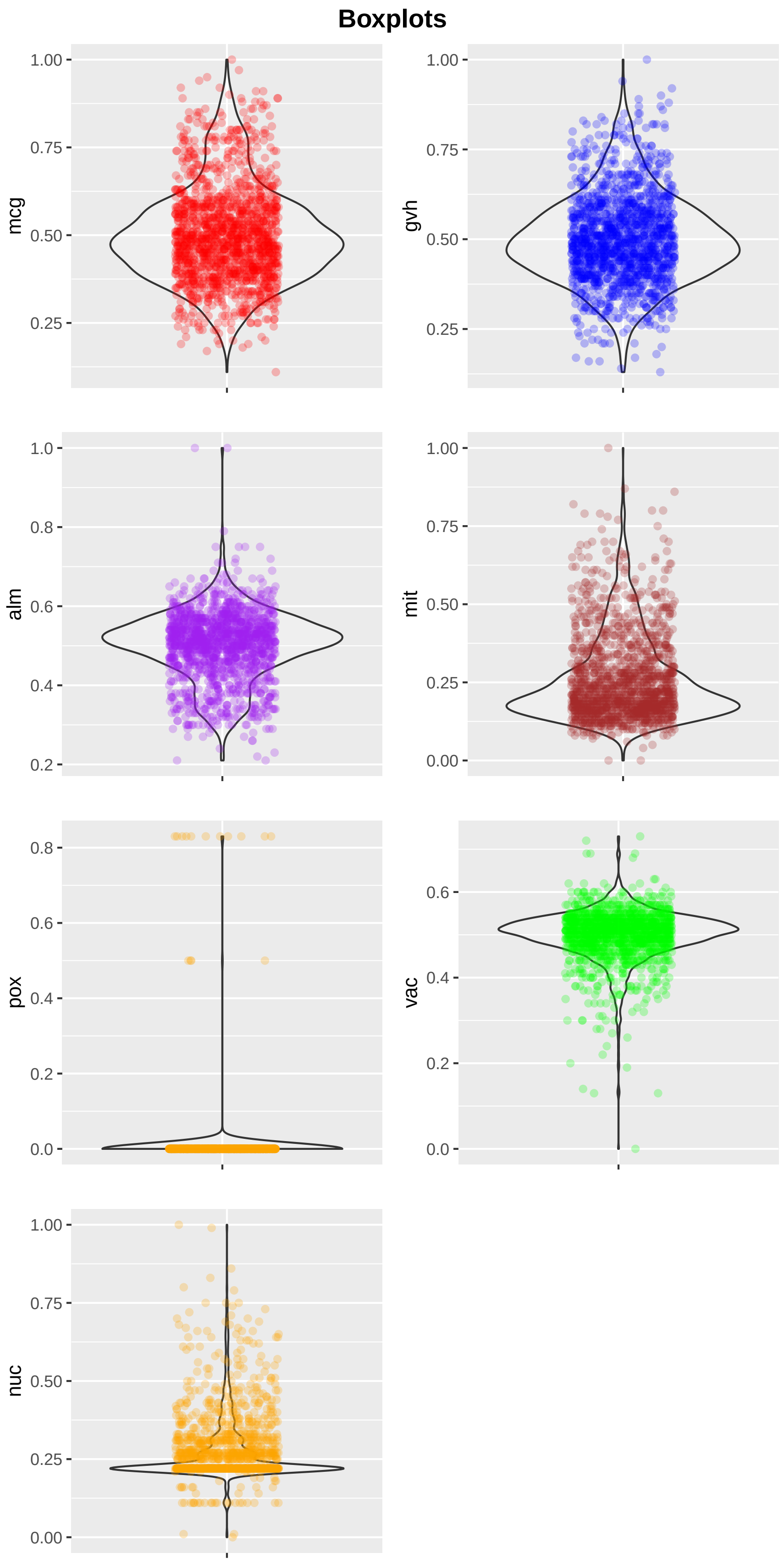


Figure 2: Boxplot comparing basic statistic for all columns

Mcg and gvh are normally distributed. Alm and mit are skewed but nothing crazy to worry about. Vac and nuc are really skewed however, not yet bad enough to worry about it. Pox is looks weird let's take a detailed look at pox.

```r
table(data[, "pox"])
```

```

   0  0.5 0.83 
1469    4   11 
```

As you can see almost all the numbers are 0. This does not mean that we just discord this column. Maybe it is very significant if the number is not 0.

```r
# Subset the data where pox is larger than 0
subset(data, pox > 0, select=loc.site)
```

```
# A tibble: 15 x 1
   loc.site
   <fct>   
 1 CYT     
 2 MIT     
 3 POX     
 4 POX     
 5 POX     
 6 POX     
 7 POX     
 8 POX     
 9 POX     
10 MIT     
11 POX     
12 MIT     
13 POX     
14 POX     
15 POX     
```

If the number is non-zero then the probability is very high that the protein is localised in peroxisomal region.

Bivariate Analysis

With a heatmap, you can easily see where there are correlations between variables. First let's create a correlation matrix

```r
# Create a cor matrix
cor_matrix <- cor(data[, -c(5,9)])
cor_matrix <- as_tibble(cor_matrix)
# Add a column with the varibale names
cor_matrix <- cor_matrix %>% mutate(varnames = all_of(attr_names)[-c(1,6,10)])
kbl(cor_matrix, booktabs = T, align = "c") %>%
  kable_styling(full_width = T)
```

mcg	gvh	alm	mit	pox	vac	nuc	varnames
1.0000000	0.5816314	-0.1639513	0.1581755	0.0055970	0.0750427	-0.1245404	mcg
0.5816314	1.0000000	-0.2718000	0.1403139	0.0003918	0.0887594	-0.1029840	gvh
-0.1639513	-0.2718000	1.0000000	0.0596683	0.0093779	-0.1858054	-0.0220428	alm
0.1581755	0.1403139	0.0596683	1.0000000	-0.0090398	-0.1035914	-0.0547965	mit
0.0055970	0.0003918	0.0093779	-0.0090398	1.0000000	0.0208997	-0.0356586	pox
0.0750427	0.0887594	-0.1858054	-0.1035914	0.0208997	1.0000000	0.0896904	vac
-0.1245404	-0.1029840	-0.0220428	-0.0547965	-0.0356586	0.0896904	1.0000000	nuc

The calculated correlation matrix. This needs to be tranformed to a long matrix.

```
# Create a long matrix
cols <- all_of(attr_names)[-c(1,6,10)]
cor_matrix_long <- pivot_longer(data = cor_matrix,
                                cols = cols,
                                names_to = "variable", values_to = "cor")
pander(head(cor_matrix_long))
```

varnames	variable	cor
mcg	mcg	1
mcg	gvh	0.5816
mcg	alm	-0.164
mcg	mit	0.1582
mcg	pox	0.005597
mcg	vac	0.07504

The long calculated correlation matrix. Let's visualise this using a heatmap.

```
ggplot(data = cor_matrix_long, aes(x=varnames, y=variable, fill=cor)) +
    geom_tile() +
    labs(x=NULL, y=NULL, title="Heatmap Correlation") +
    scale_fill_gradient(high = "purple", low = "white" )
```

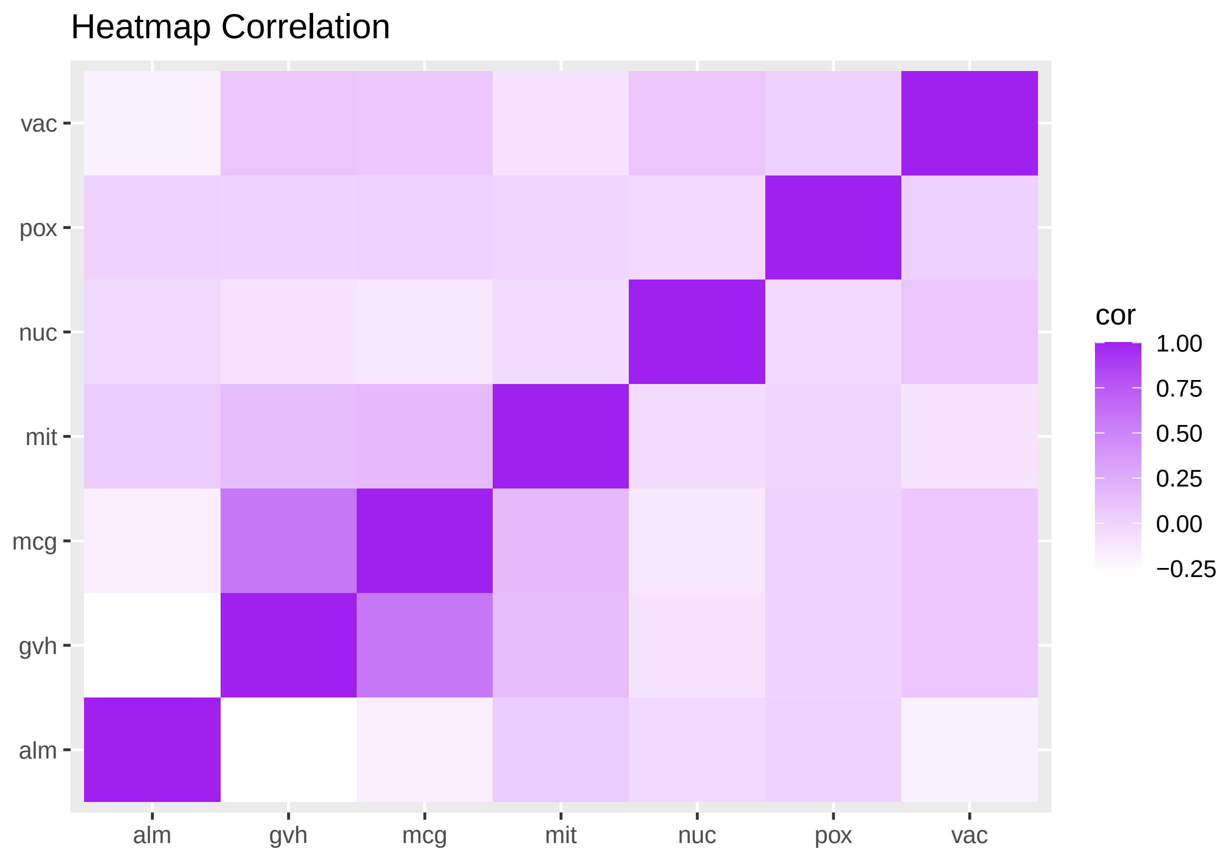


Figure 3: A heatmap pairwise correlation of selected numeric variables

As seen from the heatmap there is some correlation between mcg and gvh. Let's visualise this.

```
ggplot(data, aes(x=mcg, y=gvh, color=loc.site)) +
        labs(x="MCG", y="GVH") +
    geom_jitter(mapping = aes(color=loc.site),
                na.rm=T, width=0.2, height=0.2,
                alpha=0.5, shape=16, size=0.8) +
    ylim(0,1) + labs(title="Scatterplot and trendline") +
    geom_smooth(formula = y ~ x, method = "loess")
```

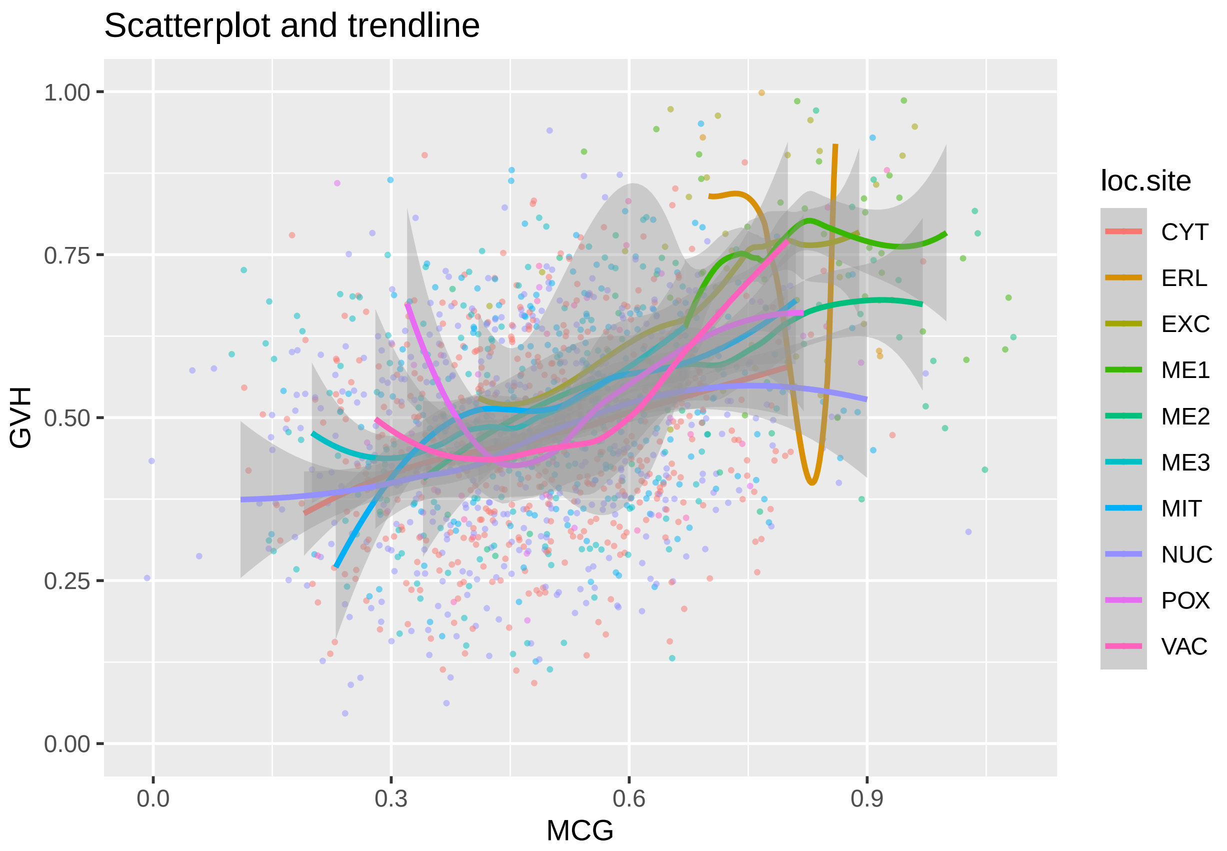


Figure 4: Scatterplot with trendline with the dependent variables

Every variable goes with a slow upward trend except erl this one has a weird curve.

Class Labels

Now we need to look at how the data correlates with the classes. The heigth of the peak doesn't matter much, shifted peaks do.

```
p1 <- ggplot(data, aes(x=mcg)) + geom_density(aes(color=loc.site))
p2 <- ggplot(data, aes(x=gvh)) + geom_density(aes(color=loc.site))
p3 <- ggplot(data, aes(x=alm)) + geom_density(aes(color=loc.site))
p4 <- ggplot(data, aes(x=mit)) + geom_density(aes(color=loc.site))
p5 <- ggplot(data, aes(x=pox)) + geom_density(aes(color=loc.site))
p6 <- ggplot(data, aes(x=vac)) + geom_density(aes(color=loc.site))
p7 <- ggplot(data, aes(x=nuc)) + geom_density(aes(color=loc.site))
plot <- ggarrange(p1, p2, p3, p4, p5, p6, p7, ncol=4, nrow=2,
         common.legend=TRUE, legend="right")
annotate_figure(plot, top = text_grob("Density plots", face = "bold", size = 14))
```

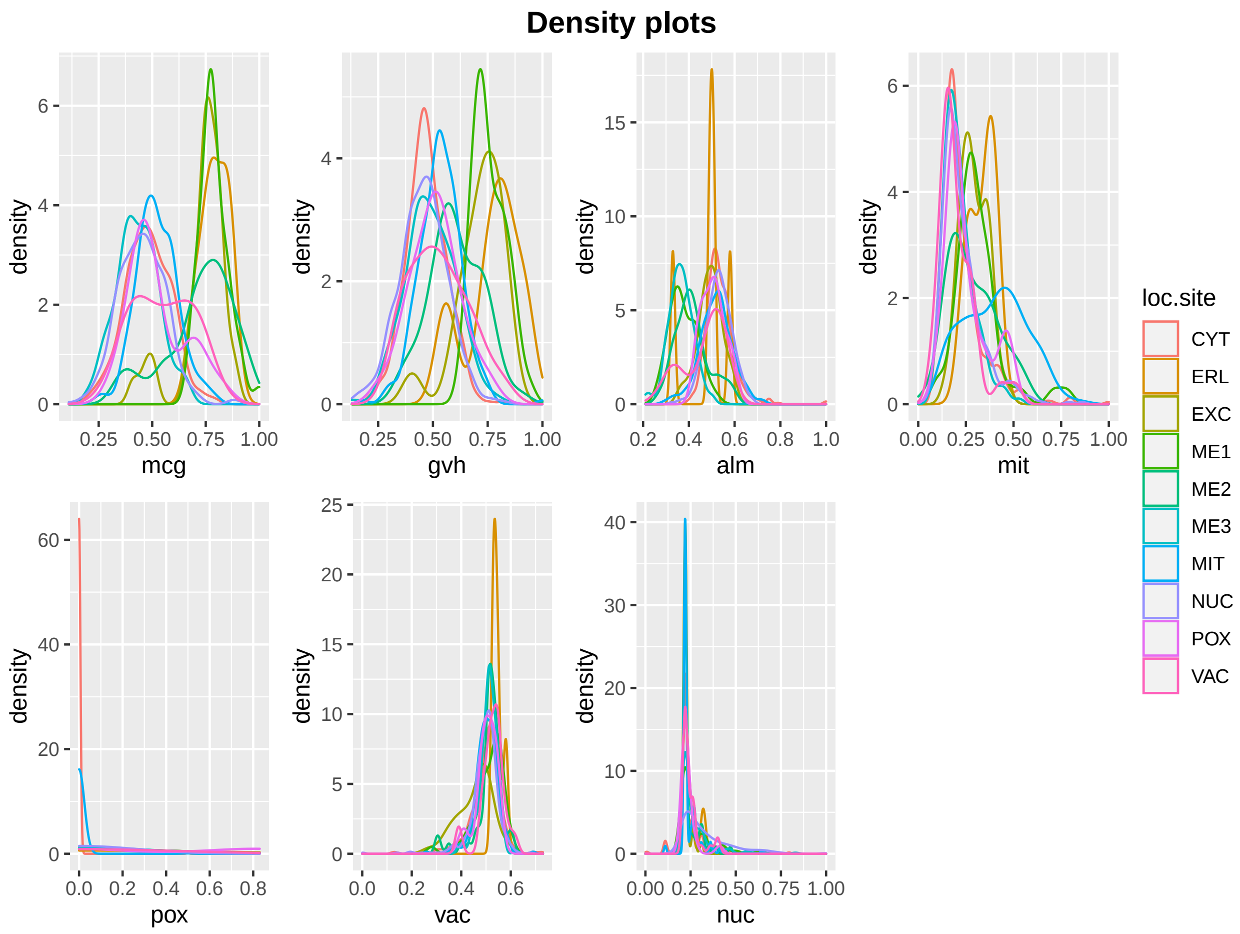


Figure 5: Density plots shows class distinction

At pox, vac and nuc, you don't see shifted peaks. At mcg and gvh you really see shifted peaks this shows a distribution of the different classes. There are small shifted peaks at alm en mit, this shows that there is difference but not so much. Let's look at mcg and gvh using ridge lines plot.

```
p1 <- ggplot(data, aes(x=mcg, y=loc.site, fill = loc.site)) +
  geom_density_ridges2(rel_min_height = 0.005) + theme_minimal() +
  coord_cartesian(clip = "off")
p2 <- ggplot(data, aes(x=gvh, y=loc.site, fill = loc.site)) +
  geom_density_ridges2(rel_min_height = 0.005) + theme_minimal() +
  coord_cartesian(clip = "off")
plot <- ggarrange(p1, p2, common.legend = TRUE)
annotate_figure(plot, top = text_grob("Ridge line plot", face = "bold", size = 14))
```

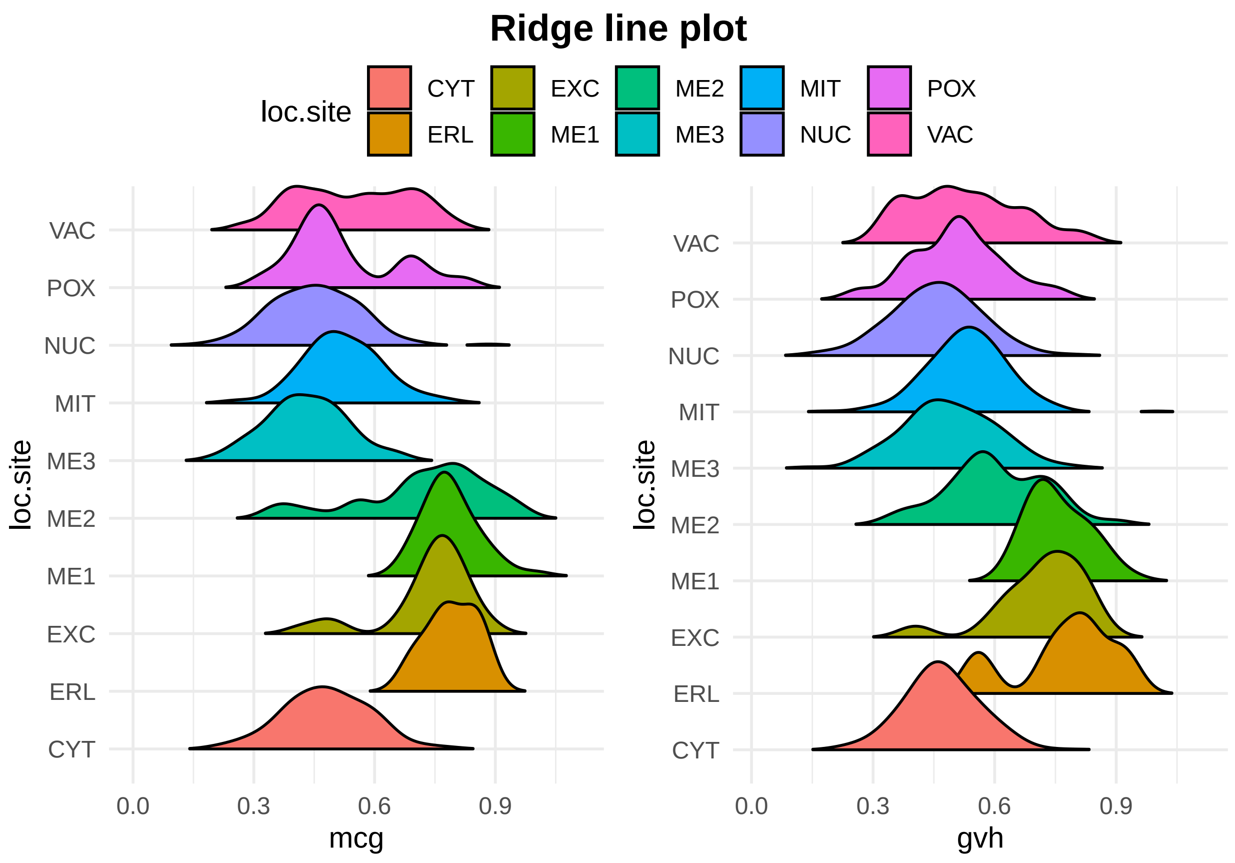


Figure 6: Ridge line plot between mcg and gvh

Now you can really see that the peaks are shifted.

We need to test the data to see if there is significant difference between it. Using a 1-way ANOVA test.

```
# Perform a one way anova test
res.aov <- summary(aov(mcg ~ loc.site, data = data))
res.aov[[1]]$`Pr(>F)`[1]
```

```
[1] 1.203676e-146
```

The P-value is < 0.05. So there is a significant difference.

Multivariate analysis

One way to see if groups are clustered is to MDS plot the groups and calculated the distance matrix.

```
# Create a matrix
matrix <- with(data, rbind(mcg, gvh, alm, mit, pox, vac, nuc))
(distmat <- dist(matrix))
```

```
          mcg       gvh       alm       mit       pox       vac
gvh  4.623700
alm  6.699455  6.524822
mit 11.476977 11.321051 11.025910
pox 19.910020 19.776585 19.479594 11.495734
vac  5.580672  5.083611  4.342165 10.946100 19.312255
nuc 11.161711 10.858508 10.144550  6.884693 11.545921  9.715282
```

Here is the distance matrix.

```
autoplot(cmdscale(distmat, eig = TRUE), shape = FALSE, label = TRUE, label.size = 4)
```

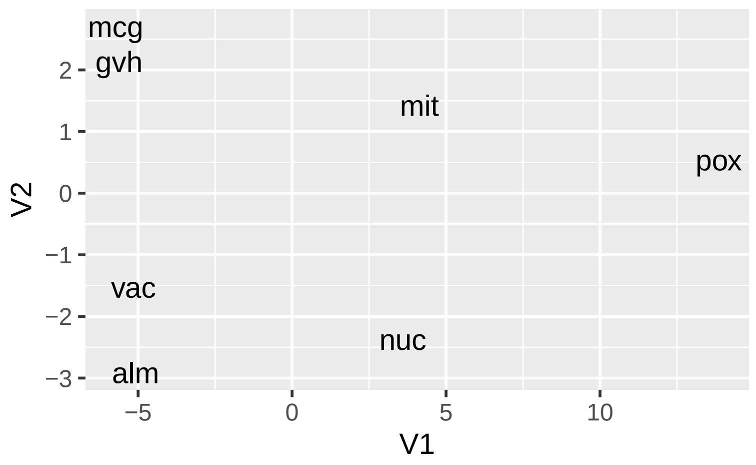


Figure 7: Classical (Metric) Multidimensional Scaling

As seen from above mcg and gvh are the clustered group.

A other plot to show is using PCA.

```r
df <- data[-c(5,9)]

pca_res <- prcomp(df, scale. = TRUE)

autoplot(pca_res, data = data, colour = 'loc.site', loadings = TRUE, loadings.label = TRUE)
```

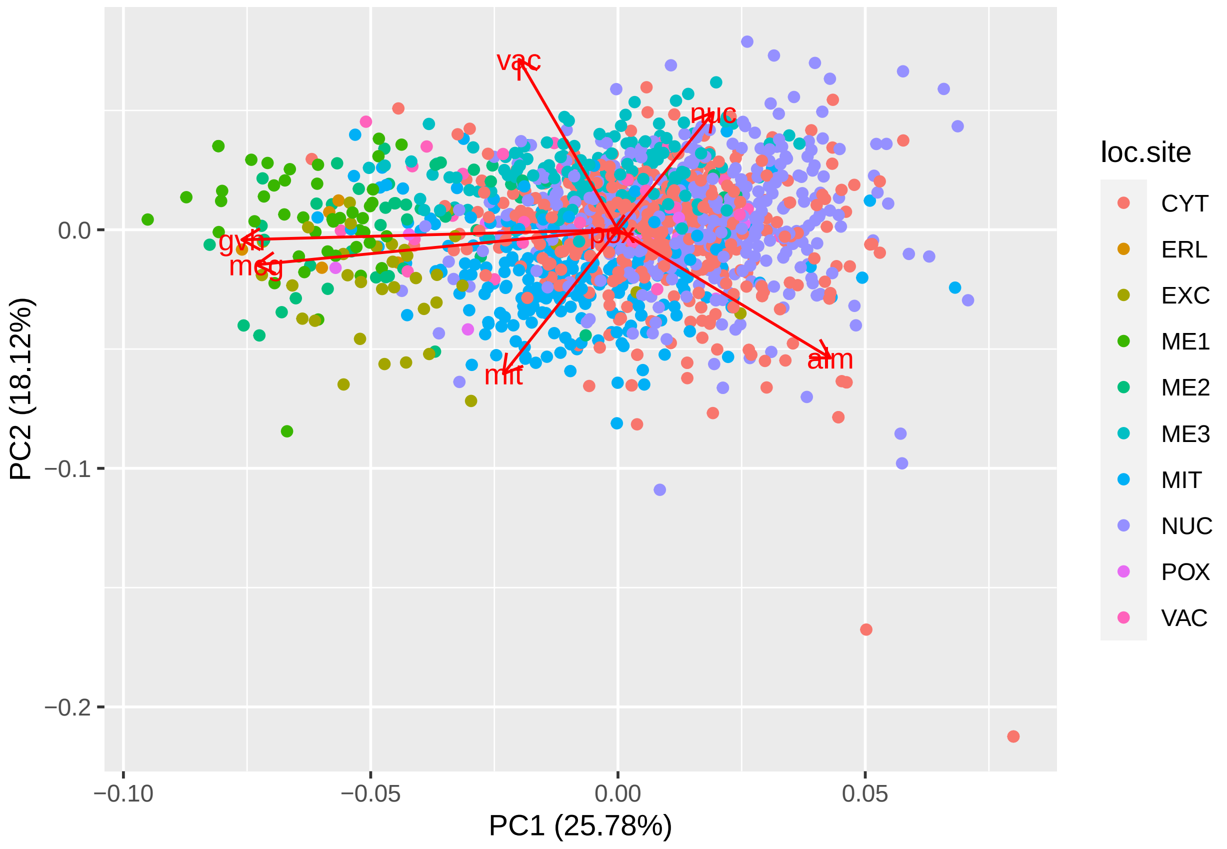


Figure 8: PCA plot showing the groups

As earlier shown mcg and gvh are grouped together.

MIL Algorithms

First we need to investigate which standard ML algorithms is the best to use for this dataset using Weka.

Weka accepts .arff files. You can convert a .csv to a .arff file in Weka. So we need to export the clean dataset to a .csv file.

```
write.table(data,"data.csv", quote = FALSE, sep = ",", row.names=FALSE)
```

Open the .csv in Weka under tools -> ArffViewer. And saved the file as .arff. The first 15 lines of the .arff file needs to look like this.

```
cat(readLines("Resources/data.arff", n = 15), sep = "\n")
```

```
@relation data

@attribute mcg numeric
@attribute gvh numeric
@attribute alm numeric
@attribute mit numeric
@attribute erl {FALSE,TRUE}
@attribute pox numeric
@attribute vac numeric
@attribute nuc numeric
@attribute loc.site {MIT,NUC,CYT,ME1,EXC,ME2,ME3,VAC,POX,ERL}

@data
0.58,0.61,0.47,0.13,FALSE,0,0.48,0.22,MIT
0.43,0.67,0.48,0.27,FALSE,0,0.53,0.22,MIT
```

Criteria For MIL Algorithms

An accurate algorithm is needed that accurately predicts the dataset. It needs a high percentage of correct classifications on the dataset. An algorithm that is fast. What if there is a dataset of 1 million observations. And it should also not contain too many false negatives and false positives.

Research has been done on which algorithm is the best. From every representatives of all classifier categories. And ZeroR and OneR for a baseline performance. Eg: Decision Trees (C4.5, J48, RandomForest), Nearest Neighbor (IBk), SVM (SMO), NaiveBayes (Naïve Bayes) and Linear Logistic (SimpleLogistic). And carried out classifications 10-fold cross validation.

The experimenter can save the results in a csv file.

```
weka.result <- read.table("Resources/Result.csv", header = TRUE, sep = ",")
# Remove weka.classifiers.* from classifier name
for (i in 1:length(weka.result$Key_Scheme)) {
  name <- weka.result$Key_Scheme[i]
  tweaked.name <- strsplit(name, split = ".", fixed = TRUE)[[1]][4]
  weka.result$Key_Scheme[i] <- tweaked.name
}
```

Now the results of the csv can been plotted with ease of use. Like the ROC and precision.

```
# Create a dataframe of the mean of the ROC of every classifier
res <- aggregate(Area_under_ROC ~ Key_Scheme, data = weka.result, mean)

ggplot(data=res, aes(x=Key_Scheme, y=Area_under_ROC, fill=Key_Scheme)) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1)) +
  labs(x="Classifier", y="Area under curve (ROC)",
       title="Area under the curve for different classifiers") +
  geom_hline(aes(yintercept=min(Area_under_ROC)), linetype = 'dashed') +
  geom_hline(aes(yintercept=max(Area_under_ROC)), linetype = 'dashed') +
  geom_text(aes(label=round(Area_under_ROC, digits=2)), vjust=1.5, size=3.5)
```

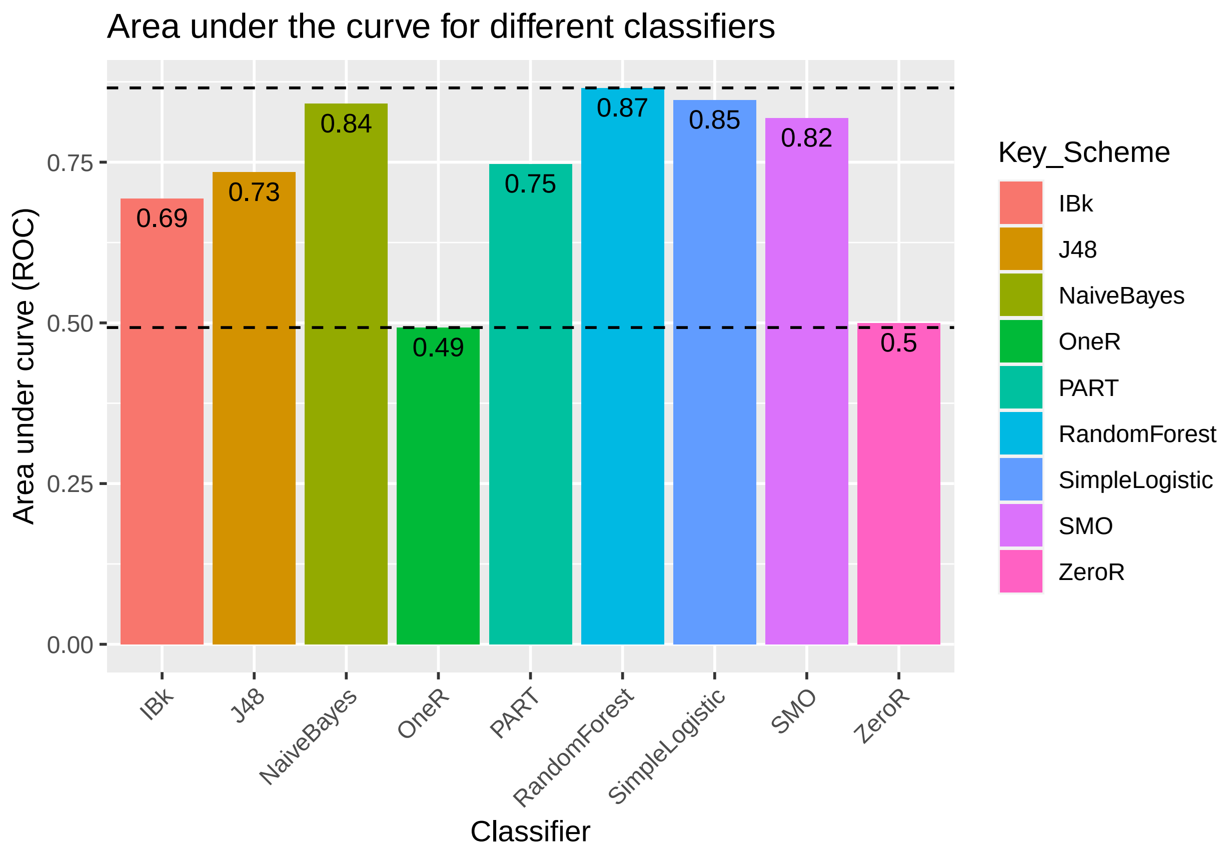


Figure 9: Area under the curve for different classifiers

RandomForest has the highest area under the curve, while OneR has the lowest with 0.49 and ZeroR with a close second last with 0.5.

```r
res <- aggregate(IR_precision ~ Key_Scheme, data = weka.result, mean)

ggplot(data=res, aes(x=Key_Scheme, y=IR_precision, fill=Key_Scheme)) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1)) +
  labs(x="Classifier", y="Precision",
       title="Average precision for different classifiers") +
  geom_hline(aes(yintercept=min(IR_precision)), linetype = 'dashed') +
  geom_hline(aes(yintercept=max(IR_precision)), linetype = 'dashed') +
  geom_text(aes(label=round(IR_precision, digits=2)), vjust=1.5, size=3.5)
```

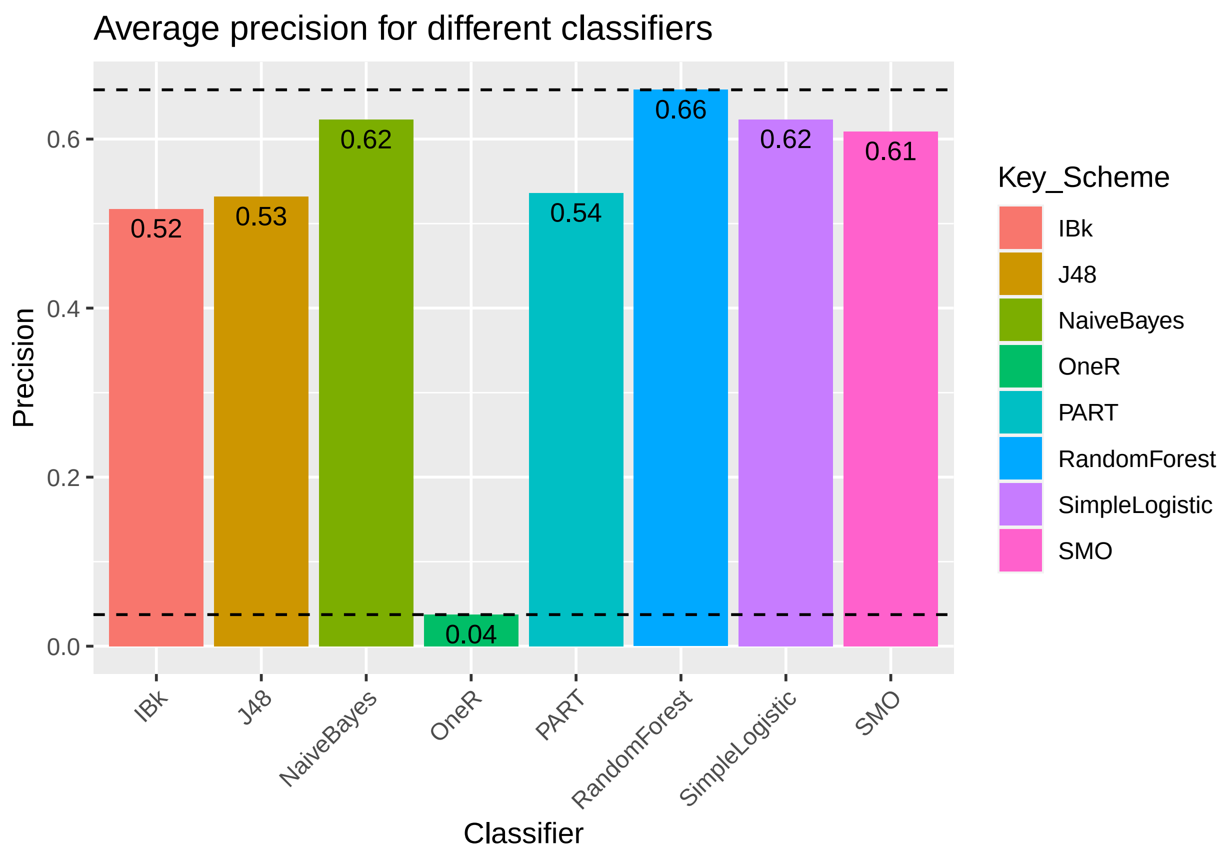
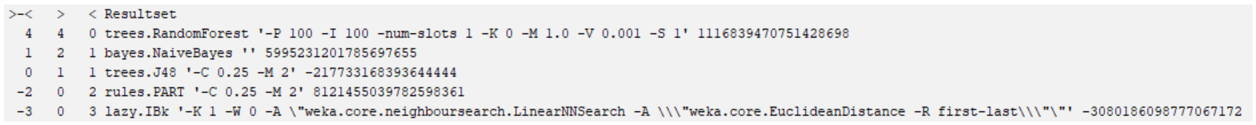


Figure 10: Average precisionfor different classifiers

RandomForest also has the highest average precision with 0.66. And OneR has the lowest with 0.04. ZeroR is not in this plots because it only consist of NaN.

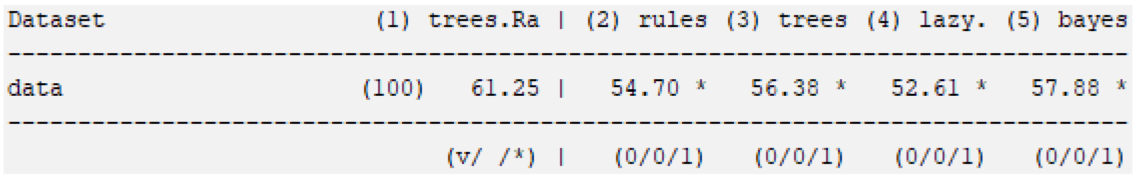
Several algorithms cannot make a prediction for the classification VAC (0.000) and for some they get an error (?). 1 algorithm IBk has a very low prediction for VAC. All the algorithm above in the figure except for ZeroR, OneR, SMO and SimpleLogistic were used in the Weka Experimenter to compare.



```
>-<   >   < Resultset
  4   4   0 trees.RandomForest '-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1' 1116839470751428698
  1   2   1 bayes.NaiveBayes '' 5995231201785697655
  0   1   1 trees.J48 '-C 0.25 -M 2' -217733168393644444
 -2   0   2 rules.PART '-C 0.25 -M 2' 8121455039782598361
 -3   0   3 lazy.IBk '-K 1 -W 0 -A \"weka.core.neighboursearch.LinearNNSearch -A \\\"weka.core.EuclideanDistance -R first-last\\\"\"' -3080186098777067172
```

Figure 11: Ranking of chosen algorithms

This shows the ranking of the algorithms by the number of times a given algorithm beat the other algorithms. And RandomForest beat all 4 algorithms.



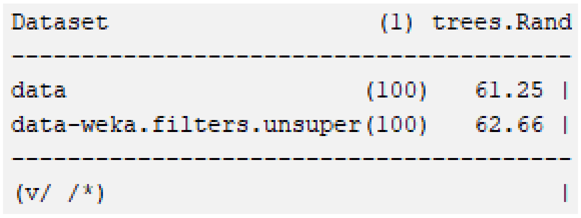
```
Dataset                    (1) trees.Ra | (2) rules (3) trees (4) lazy. (5) bayes
------------------------------------------------------------------------------------
data                      (100)   61.25 |   54.70 *   56.38 *   52.61 *   57.88 *
------------------------------------------------------------------------------------
                               (v/ /*) |   (0/0/1)   (0/0/1)   (0/0/1)   (0/0/1)
```

Figure 12: Ranking of chosen algorithms by SD

RandomForest has a ‘v’ next to their result. This means that the difference in the accuracy for these algorithms compared to all the other algorithms is significant.

After testing multiple algorithms in Weka, the algorithm RandomForest was chosen. It has the highest average precision of 0.611. And this was one of the few algorithms where the precision for VAC was not a ‘?’.

But for the ratings, no algorithm could make a prediction except 1 for VAC, however, it was also very low. So we looked at whether there is siginificant difference if we remove this classifier.



```
Dataset                    (1) trees.Rand
------------------------------------------
data                      (100)   61.25 |
data-weka.filters.unsuper(100)   62.66 |
------------------------------------------
(v/ /*)                                 |
```

Figure 13: Removed class difference

And by the looks of it, there is no significant difference. So VAC is not removed from the dataset.

RandomForest has seven attributes. Below is a table with each attribute and its meaning.

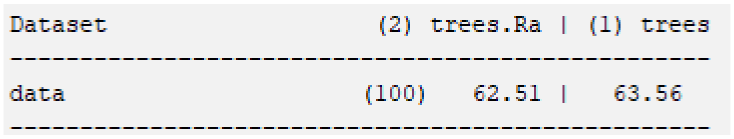
Attribute	Description
P	Size of each bag, as a percentage of the training set size.
I	Number of iterations.
num-slots	Number of execution slots.
K	Number of attributes to randomly investigate.
M	Number of attributes to randomly investigate.
V	Set minimum numeric class variance proportion.
S	Seed for random number generator.

Weka will find the optimal parameters for a classifier. With CVParameterSelection in Weka were the best parameters found.

Attribute	Old.Values	New.Values
P	100	50
I	100	1000
num-slots	1	0
K	0	1
M	1	1

Attribute	Old.Values	New.Values
V	0.001	0.001
S	1	1

This table shows the parameters with their default settings and their new. P, I, num-slots and K were different than the default settings. The build will take more time because it does 10 times more I(terations).



```
Dataset                    (2) trees.Ra | (1) trees
---------------------------------------------------
data                      (100)   62.51 |   63.56
---------------------------------------------------
```

Figure 14: Iterations ranked

Number 1 in the figure above is with 1000 I(terations) and number 2 is 100 I(terations). And there is not a significant difference. So for performance, the value for I will be 100.

Weka Meta learners were used to get a better idea of an optimal classifier.

Classifier	TP Rate	FP Rate	Precision	ROC Area	
RandomForest.Standard	0.611	0.132	0.603	0.844	Weighted.Avg
RandomForest.tweaked	0.62	0.132	0.610	0.846	Weighted.Avg
Boosting	0.586	0.142	0.586	0.8	Weighted.Avg
Stacking	0.544	0.157	0.538	0.758	Weighted.Avg
Bagging	0.623	0.13	?	0.853	Weighted.Avg
Vote	0.625	0.128	?	0.851	Weighted.Avg

As it can be seen in this table, RandomForest with tweaked parameters is the best. This one has no '?' at VAC and precision.