Eosinofiele Oesofagitis dataset EDA

Jim Jim Valkema

16/11/2020

Dataset

Nutritional intake was assessed in 40 Dutch adult EoE patients participating in the Supplemental Elemental Trial (SET) using 3-day food diaries. In this randomized controlled trial, diagnosed patients received either a four-food elimination diet alone (FFED) or FFED with addition of an amino acid-based formula (Neocate) for 6 weeks. Disease severity was assessed by peak eosinophil count/high power field (PEC) in esophageal biopsy specimens. Multiple linear regression analyses were performed to assess associations between the intake of nutrients and foods per 1000 kCal and PEC, both at baseline and after the 6 weeks diet, while controlling for baseline variables. TODO code book

Research question

Can the peak eosinophil baseline in individuals with eosinophilic oesophagitis be predicted with classical machine learning using the measured nutrient intake data?

Missing data

There are quite a bit of missing attributes and missing values in this data set. Luckily there are plenty of attributes that do have data from where the machine learning algorithm can make its prediction. Figure 1 shows that a lot of attributes miss more than 70% of their values which in combination with the few instances of this data set means that those attribute are not of value to a machine learning algorithm.

```r
library(ggpubr)
```

```
## Loading required package: ggplot2
```

```r
library(foreign)
library(janitor)
```

```
##
## Attaching package: 'janitor'
```

```
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

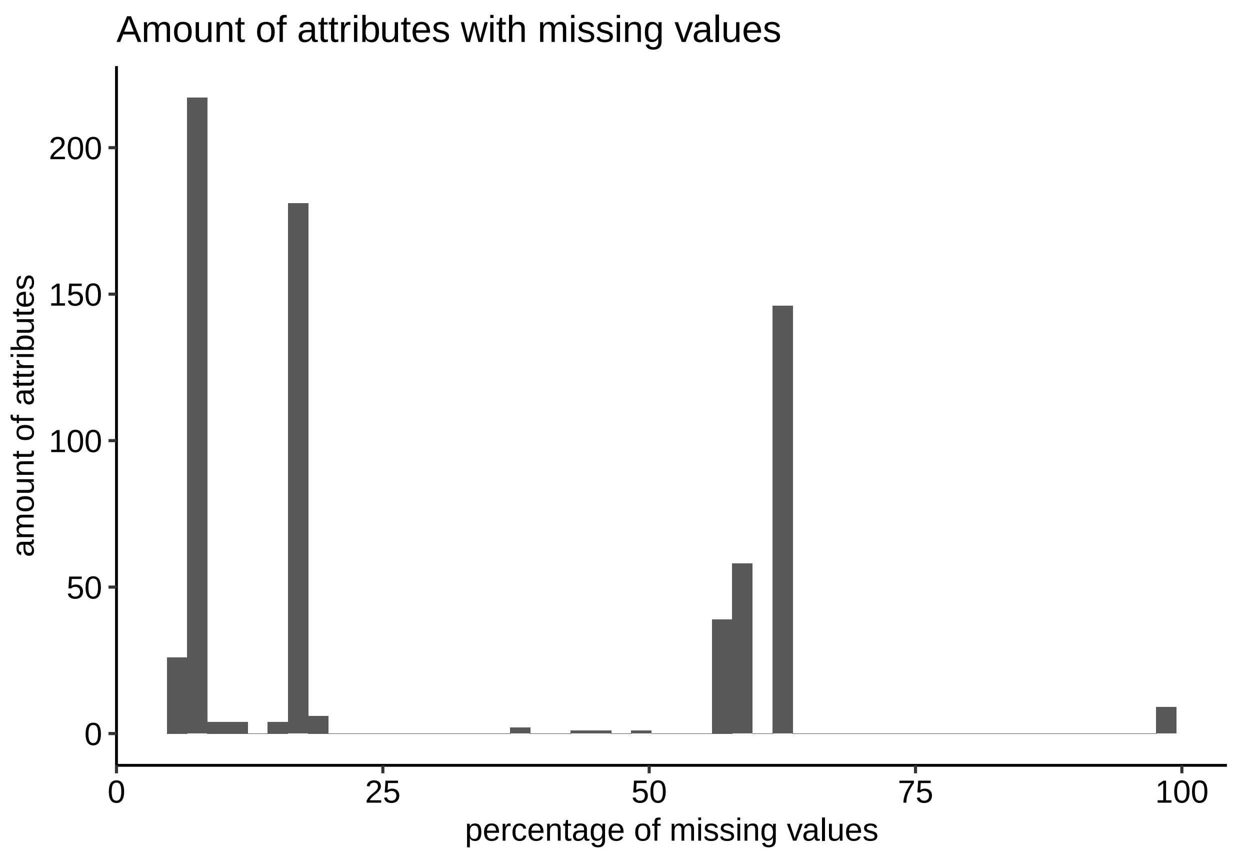
```r
library(memisc)
```

```
## Loading required package: lattice

## Loading required package: MASS

##
## Attaching package: 'memisc'

## The following object is masked from 'package:ggplot2':
##
##     syms

## The following objects are masked from 'package:stats':
##
##     contr.sum, contr.treatment, contrasts

## The following object is masked from 'package:base':
##
##     as.array
```

```r
library(ggplot2)

# http://www.milanor.net/blog/how-to-open-an-spss-file-into-r/
dataset_original = read.spss("../../data/Datafile compleet 20-03-2020 voor Hanze SET def.sav", to.data.
dataset_no_empty <- remove_empty(dataset_original, which = c("rows", "cols"), quiet = FALSE)
```

```
## Removing 37 empty rows of 79 rows total (46.8%).

## Removing 180 empty columns of 879 columns total (Removed: Date, Eoscutoff, Lenght, Weight, BMI, Age_
```

```r
# get amount of missing values
columns.NA <- as.data.frame(colSums(is.na(dataset_no_empty)))

# convert to percentage and plot TODO ggplot
get_percentage <- function(x) return((x/nrow(dataset_no_empty))*100)
columns.NA <- apply(columns.NA, 1, get_percentage)
names(columns.NA) <- c()

# plot
ggplot(data.frame(columns.NA), aes(x=columns.NA)) + geom_histogram(bins = 50) +
  ggtitle( "Amount of attributes with missing values") +
  xlab("percentage of missing values") +
  ylab("amount of attributes") +
  theme_pubr()
```



```
# remove all attributes with less than 40% na
dataset_clean <- dataset_no_empty[, which(colMeans(!is.na(dataset_no_empty)) > 0.4)]
```

Figure 1: Shows the amount of attributes that miss a certain percentage of values. With the percentage of missing values on the x axis and the amount of attributes on the y. This plot clearly show that a lot of attributes miss 60~70 percent of their values

```
print("columns removed because too many missing values")
```

```
## [1] "columns removed because too many missing values"
```

```
ncol(dataset_no_empty)- ncol(dataset_clean)
```

```
## [1] 155
```

```
print("columns removed in total")
```

```
## [1] "columns removed in total"
```

```
ncol(dataset_original)- ncol(dataset_clean)
```

```
## [1] 335
```

```
# TODO do in the markdown text
#colnames(dataset_original)
library(gridExtra)
library(grid)
```

distribution

Most attributes seem normally distributed however their range differs quite significantly so a normalization step might be required otherwise the range can bias the machine learning algorithm to specific attributes.

```
library(reshape2)
dataset_selected <- dataset_clean[c("PeakEosBaseline_Max","B.Fat.gr", "B.Sat.fat.gr", "B.PUFAS.gr","B.P
# folate is naturally occuring in food folate acid is added but they serve the same purpose
# PUFAS = polyunsaturated fatty acids

# still 2 empty rows left
dataset_selected <- remove_empty(dataset_selected, which = c("rows", "cols"), quiet = FALSE)
```

```
## Removing 2 empty rows of 42 rows total (4.76%).
```

```
## No empty columns to remove.
```

```
# row 21 has too many missing values in all B. columns
dataset_selected <- dataset_selected[-21,]

# prevent Inf values after logging
dataset_selected[1:12] <- dataset_selected[1:12]+0.000001

nrow(dataset_selected)
```

```
## [1] 39
```

```
logged.dataset <- cbind(log2(dataset_selected[1:12]),dataset_selected[13:17], by="pid")
melted.dataset <- melt(logged.dataset)
```

```
## Using by as id variables
```

```
nrow(logged.dataset)
```

```
## [1] 39
```

```
ggplot( melt(dataset_selected[c(1:12)]), aes(value)) +
  geom_density(adjust = 0.5) +
  facet_wrap(~variable) + ggtitle( "attributes density") +
  scale_x_continuous(limits = c(0, 300)) + theme_pubr()
```

```
## No id variables; using all as measure variables
```

```
## Warning: Removed 94 rows containing non-finite values (stat_density).
```

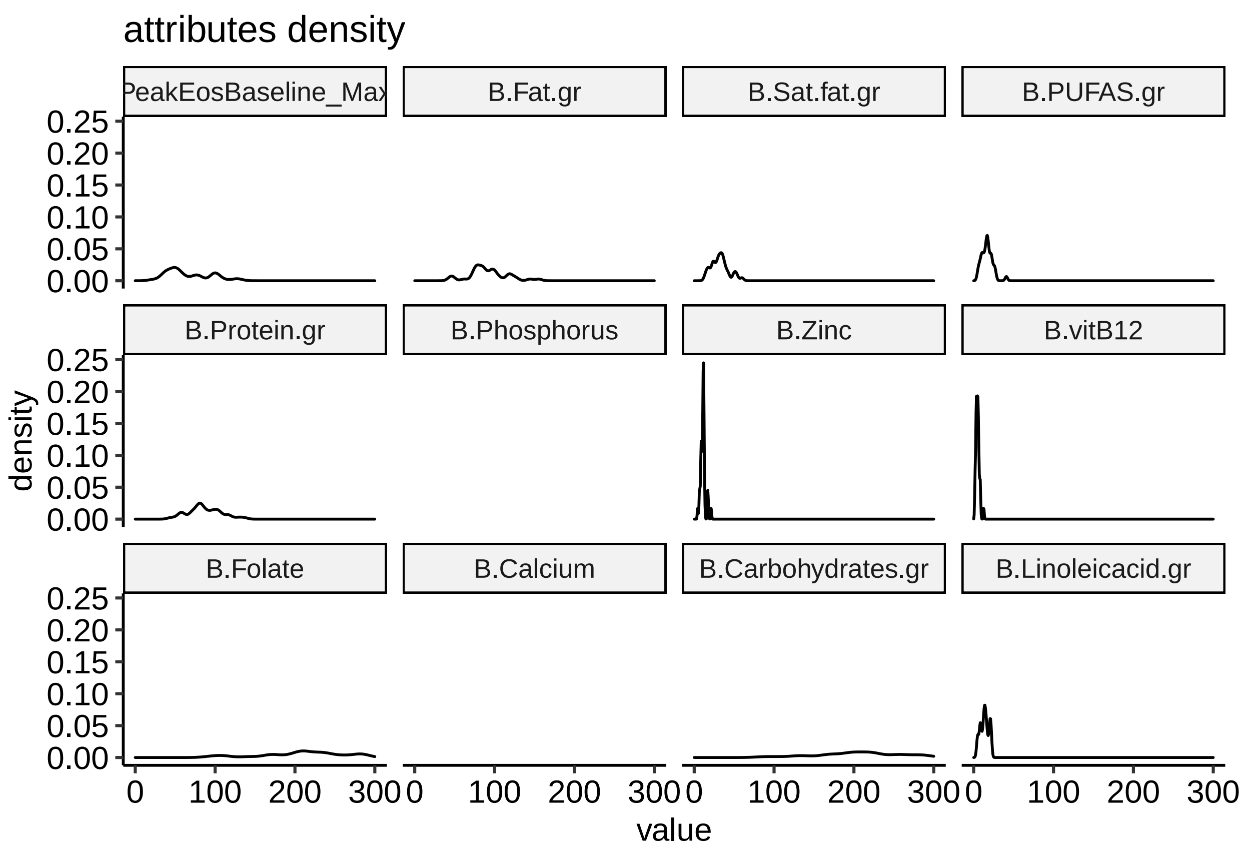


Figure 2: Shows the density of the values of a selection of attributes from the data set. With the density on the y axis and the values on the x axis. This plot clearly shows that the ranges differ dramatically and because and are skewed to the right which indicates that it could benefit from a log transformation.

```r
ggplot(melt(logged.dataset[c(1:12)]), aes(value)) +
  geom_density(adjust = 0.5) +
  facet_wrap(~variable) + ggtitle( "attributes density logged") +
  scale_x_continuous(limits = c(2, 12))  + theme_pubr()
```

```
## No id variables; using all as measure variables
```

```
## Warning: Removed 15 rows containing non-finite values (stat_density).
```

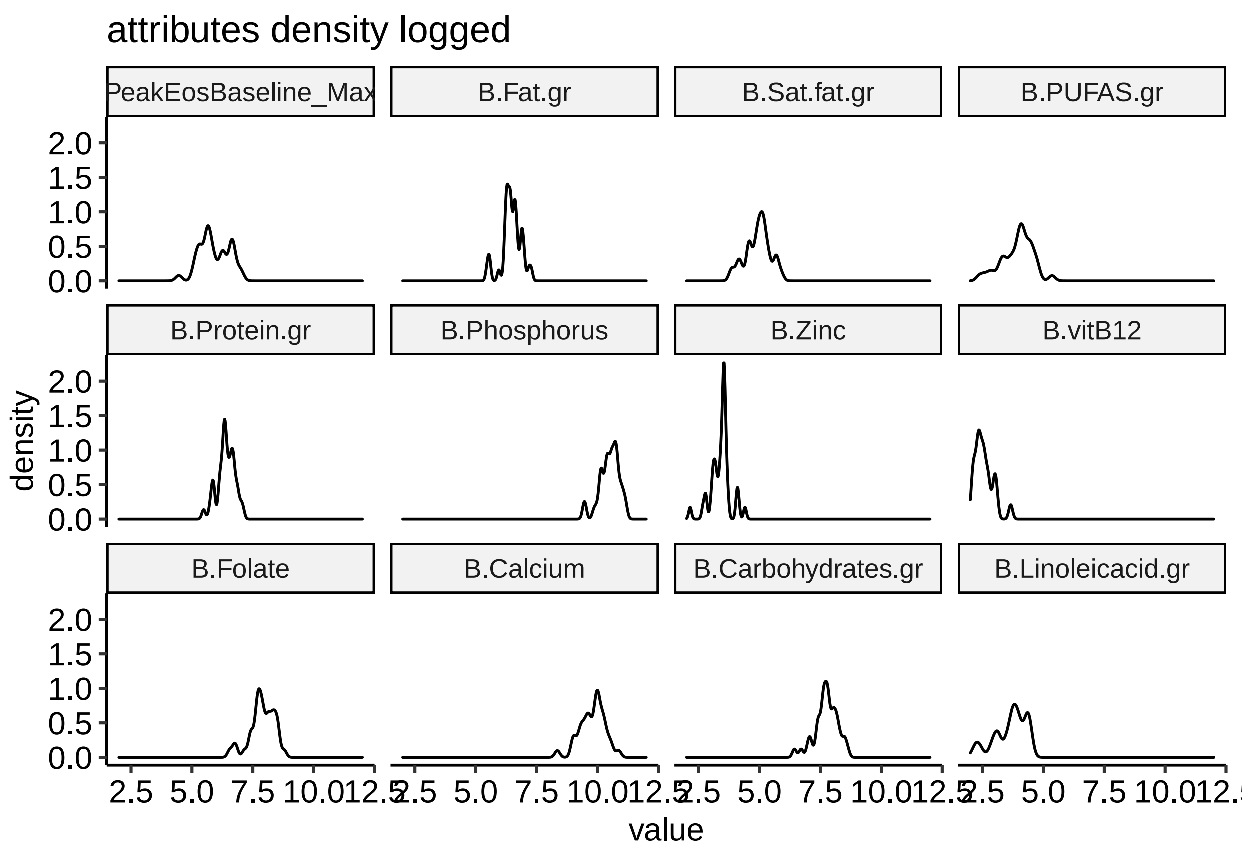


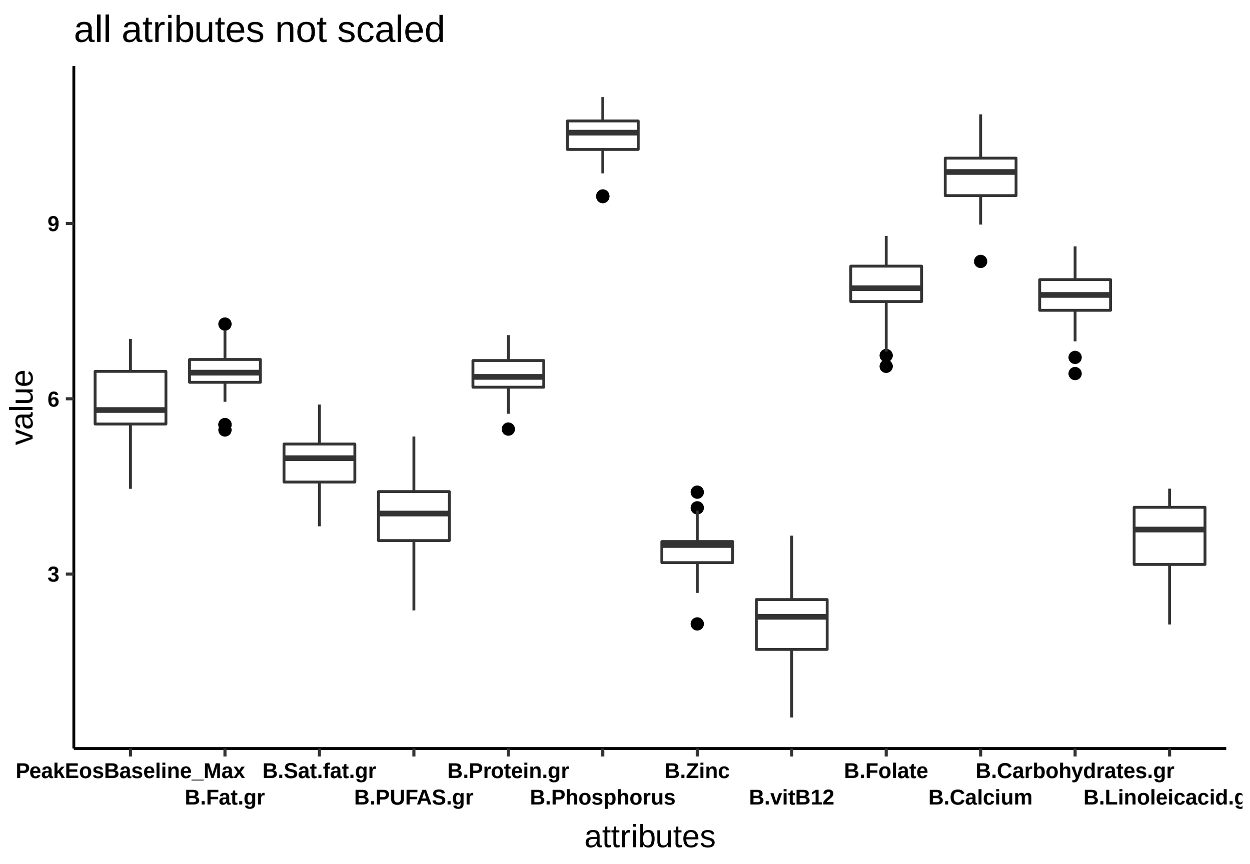
Figure 3: same as figure 2 but with then log two transformed. It shows the density of the values of a selection of attributes from the data set. With the density on the y axis and the values on the x axis.

Normalisation

The data varies a lot in its range per each attribute (figure 4) so normalisation is required. Scaling normalisation seems to be optimal since it is able to put the data in more reasonable ranges while still being able to accept new data even if it falls out of the range previously measured unlike min-max normalisation. It can also be more clearly observed that the data represent a more normal distribution from log transformation in figure 6. This is where the data is plotted in a density plot like figure 2. This is because the ranges are now more similar because of the scaling normalisation.

```
ggplot(data = melt(logged.dataset[1:12]), aes(x=variable,y=value)) + geom_boxplot(outlier.colour="black
  ggtitle("all atributes not scaled") +
  xlab("attributes") +
  scale_x_discrete(guide = guide_axis(n.dodge = 2)) +
    theme(axis.text = element_text(size = 8, face = "bold", color = "black"),
        axis.title.y = element_text(vjust = 1),
        axis.title.x = element_text(vjust = 0.005))
```

```
FALSE No id variables; using all as measure variables
```



```
#scale the data
scale_min_max <- function(x){
  x_min = min(x, na.rm = T)
  x_max = max(x, na.rm = T)
  return ((x - x_min) / (x_max - x_min))
}

min_max_norm <- cbind(apply(logged.dataset[1:12], MARGIN = 2, FUN = scale_min_max),logged.dataset[13:17
scaled_norm <- cbind(scale(logged.dataset[1:12]),logged.dataset[13:17])
scaled_norm_not_logged <- cbind(scale(dataset_selected[1:12]),dataset_selected[13:17])
```

Figure 4: Shows a selection of attributes and their distributions as a boxplot. The x axis shows the attributes and y axis its values those attributes range within. This plot shows that the range varies largely between attributes.

```
#ggplot(data = melt(min_max_norm[1:12]), aes(x=variable,y=value)) + geom_boxplot(outlier.colour="black"
ggplot(data = melt(scaled_norm[1:12]), aes(x=variable,y=value)) + geom_boxplot(outlier.colour="black", 
  xlab("attribute")+
  scale_x_discrete(guide = guide_axis(n.dodge = 2)) +
    theme(axis.text = element_text(size = 8, face = "bold", color = "black"),
        axis.title.y = element_text(vjust = 1),
        axis.title.x = element_text(vjust = 0.005))
```

```
## No id variables; using all as measure variables
```

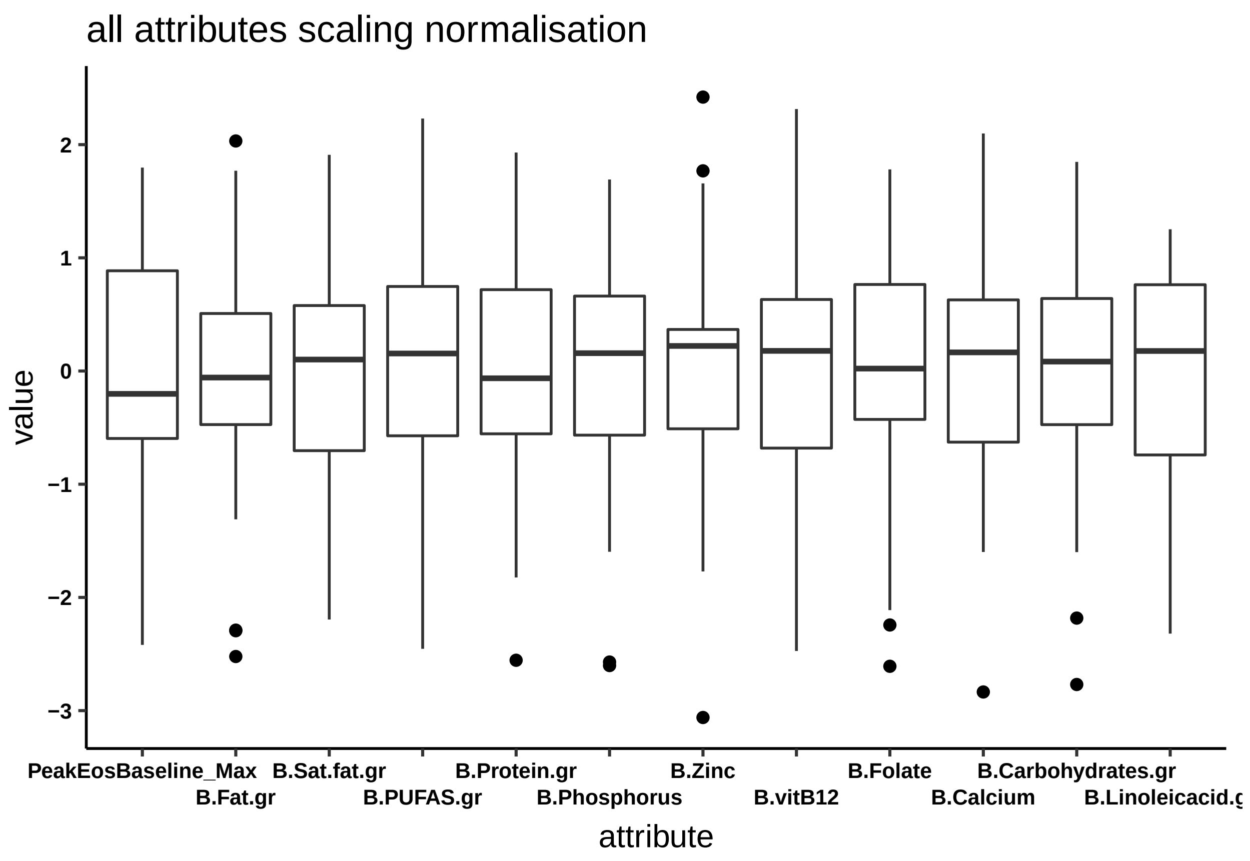


Figure 5: Shows a selection of attributes and their distributions after scaling normalisation as a boxplot. The x axis shows the attributes and y axis its values those attributes range within. This plot shows the distributions are far more comparable then without normalisation.

```r
# ggplot(melt(scaled_norm[c(1:12)]), aes(value)) +
#   geom_density(adjust = 0.5) +
#   facet_wrap(~variable) + ggtitle( "attributes density normalised and logged") + theme_pubr()
```

```r
ggplot(melt(scaled_norm_not_logged[c(1:12)]), aes(value)) +
  geom_density(adjust = 0.5) +
  facet_wrap(~variable) + ggtitle( "attributes density normalised not logged")  + theme_pubr()
```

```
## No id variables; using all as measure variables
```

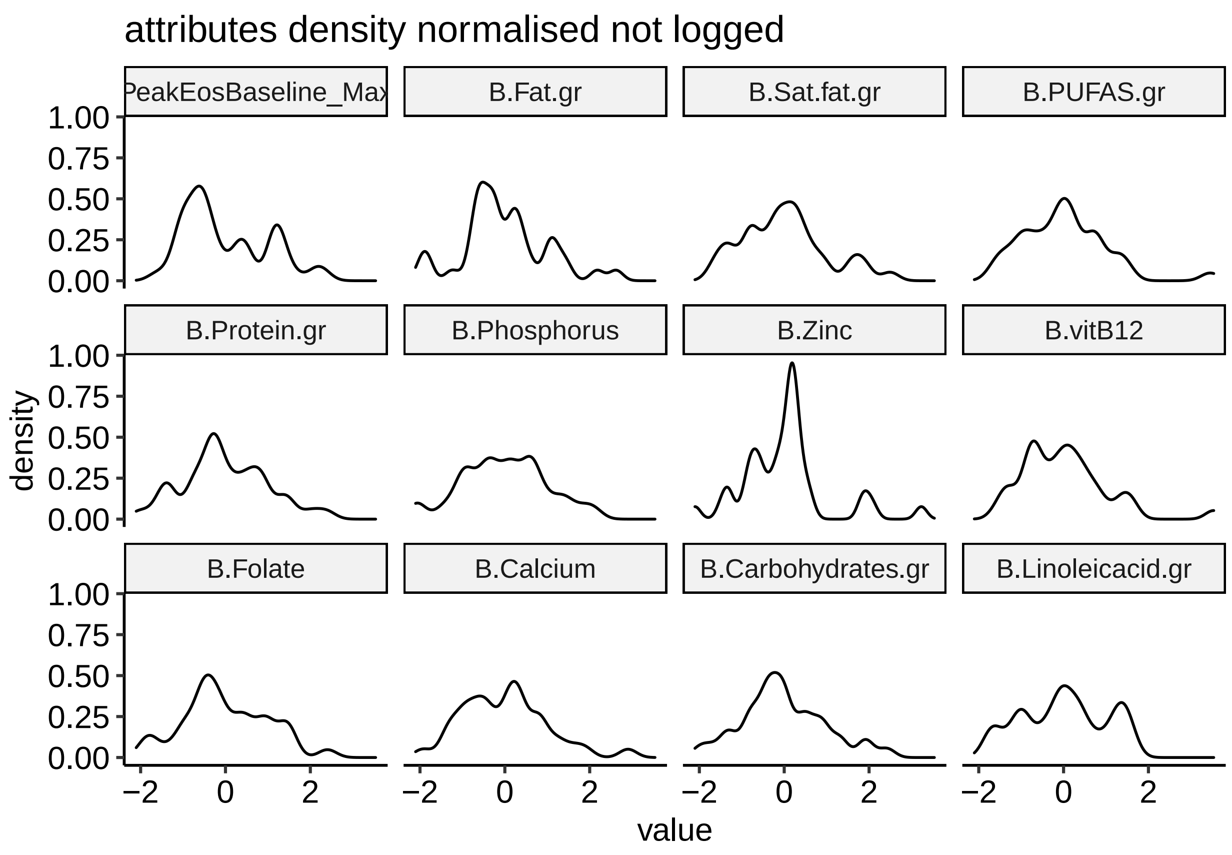
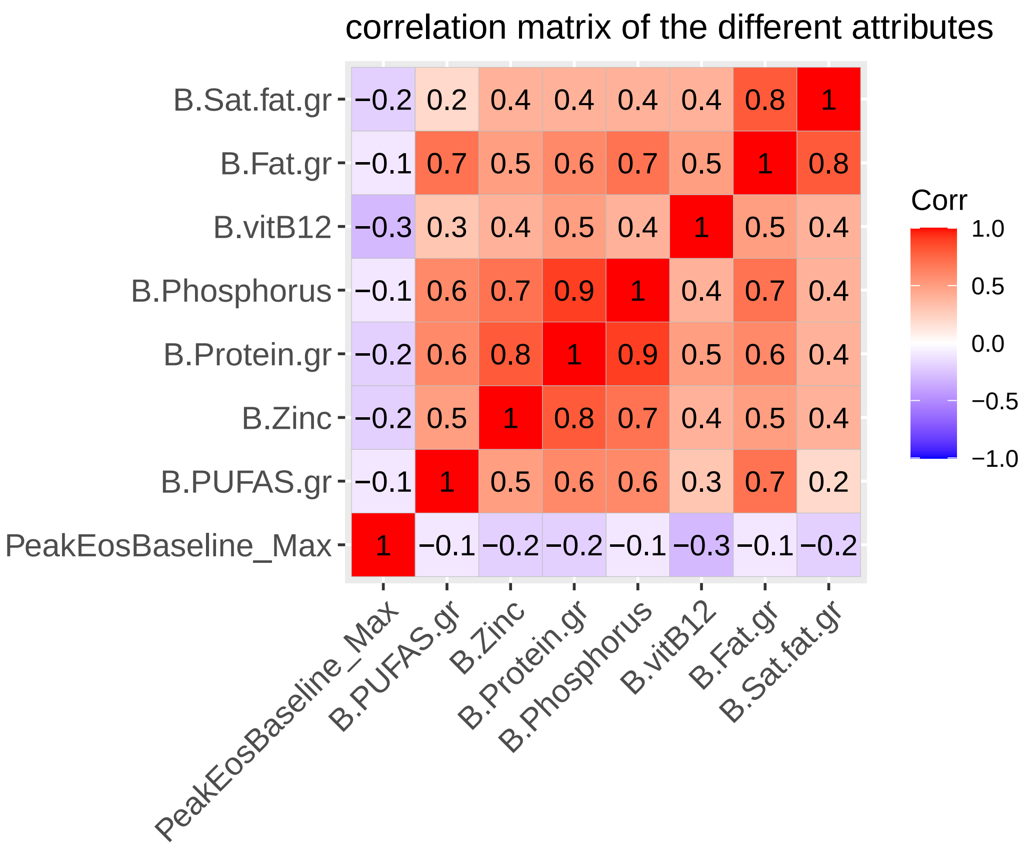


Figure 6: Shows the density of the values of a selection of attributes from the data set. With the density on the y axis and the values on the x axis. This plot shows that the data is now far more normal distributed after scaling and more comparable in its ranges then in figure 2

Correlation

It seems that the majority isn't highly correlated which might be a good indication that there is enough information between attributes for the algorithm to train on. Some attributes like saturated fat and fat in general (B.Sat.fat.gr and B.Fat.gr) are highly correlated. This means it is probably better to leave either one out.

```r
library("ggcorrplot")
# split the data in 2 because it wont fit in 1 plot
scaled_norm.1 <- scaled_norm[,1:8]

# add the first columns because that is PeakEos which is our attribute we want to predict
scaled_norm.2 <- scaled_norm[,c(1,9:17)]
cor.data.scaled.1 <- cor(scaled_norm.1)
cor.data.scaled.2 <- cor(scaled_norm.2)
ggcorrplot(cor.data.scaled.1,  ggtheme = ggplot2::theme_gray,hc.order = T,lab = T, digits=1) + ggtitle(
```



```
ggcorrplot(cor.data.scaled.2,  ggtheme = ggplot2::theme_gray,hc.order = T,lab = T, digits=1) + ggtitle(
```

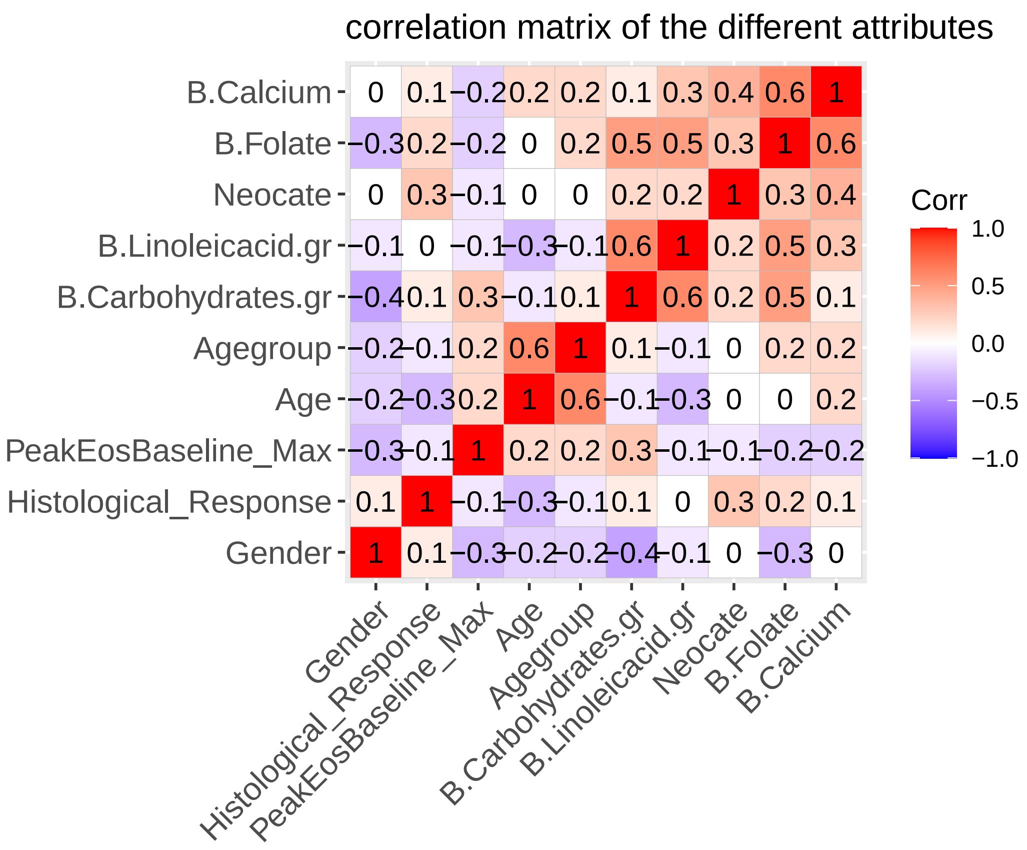


Figure 7 & 8: show the correlation between attributes. It seems that the majority isn't highly correlated which might be a good indication that there is enough information between attributes for the algorithm to train on.

splitting Peak eos baseline max into categories

The Peak eos baseline max needs to be split into categories because most classical machine learning algorithms can't use a numerical dependent class variable. The Peak eos baseline max is split into three categories: low, mid, high split at 0-49, 49-75, 75-200. This seems to create the most even distribution of instances while also being informative. It could be split into more categories but this would make it harder to create a machine learning model with high accuracy.

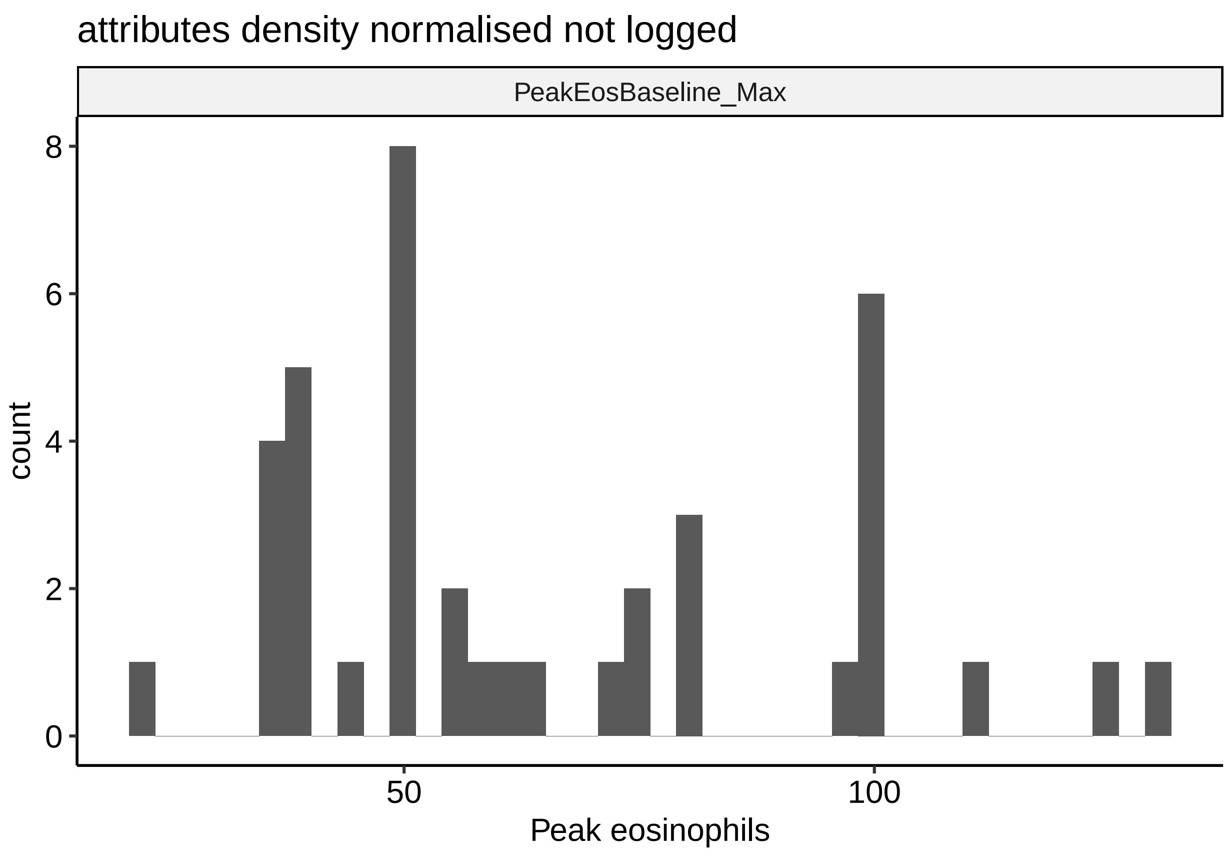
```
factor_columns <- c("Gender","Histological_Response","Neocate")
#scaled_norm[,factor_columns] <- as.factor(as.character(scaled_norm[,factor_columns]))

peak.eos.max <- remove_empty(dataset_clean[23], which = c("rows", "cols"), quiet = FALSE)
```

```
## Removing 2 empty rows of 42 rows total (4.76%).
```

```
## No empty columns to remove.
```

```
ggplot(melt(peak.eos.max), aes(value)) +
  geom_histogram(adjust = 0.5, bins = 40) +
  facet_wrap(~variable) + ggtitle( "attributes density normalised not logged") + theme_pubr() +
  xlab("Peak eosinophils")
```

```
## No id variables; using all as measure variables
## Warning: Ignoring unknown parameters: adjust
```



```
scaled_norm$PeakEosBaseline_Max <- cut(as.matrix(peak.eos.max), breaks = c(0,49,75,200), labels = c("lo
# TODO check if the gender is correct
scaled_norm$Gender <- cut(as.matrix(scaled_norm$Gender), breaks = c(-1,0,1), labels = c("male","female"
scaled_norm$Histological_Response <- cut(as.matrix(scaled_norm$Histological_Response), breaks = c(-1,0,
scaled_norm$Neocate <- cut(as.matrix(scaled_norm$Neocate), breaks = c(-1,0,1), labels = c("no response"
write.csv(scaled_norm,"../../data/selected_data_logged_scaled.csv", row.names = FALSE)

summary(scaled_norm$PeakEosBaseline_Max)
```

```
##  low  mid high
##   10   16   13
```

Figure 9: Shows the distribution of the peak max eosinophils with the peak eosinophils on the x axis and its frequency on the y axis. This plot shows that most values are approximately at 50 and 100 with the rest inbetween.

quality metrics

The most important metric to optimise for is the sensitivity a false positive case is less harmful then a false negative case. This is because a patient that is classified as sick but turns out to be healthy after further inspection, experiences far less harm then those who are classified as healthy but continue suffering from symptoms.

The algorithm should be able to run on a consumer desktop and doesn't need to be too fast when classifying. This is because it is likely that only one patient will be evaluated at the time so accuracy is far more important than speed.

model evaluation

Model evaluation is done in the weka experimenter with five data sets and seven algorithms (default setting from weka 3.8.4) and four variations of the stacking meta algorithm with different settings. The best performing algorithm is believed to be the meta stacking algorithm with the correlation-based feature selection dataset.

The meta stacking algorithm has the following settings:
Meta learner algorithm: Randomforest (default weka settings)
ensemble members (all default weka settings): J48, Random Forest, naive Bayes and logistic regression.

The other algorithms used are:
Naive Bayes
Logistic regression
K nearest neighbor
OneR
ZeroR
J48
Randomforest

The different data sets are created from the original dataset which is logged and normalised and then the different datasets are created by using different attribute selection methods.
These methods are: classifier attribute evaluation with j48. Done with the setting leave one out turned on and with it off.
And a dataset is created with correlation-based feature selection. The classifier attribute evaluation datasets also have different variations on when an attribute is no longer included based on rank.

```
library(DT)
library(rio)
#install_formats()

weka.results <- read.csv("../../data/results/results_attribute_selection_selected_data_logged_scaled.cs

confusion.df.all <- data.frame()

for (dataset.df in split.data.frame(weka.results,weka.results$Key_Dataset)) {
  for (algo.df in split.data.frame(dataset.df,dataset.df$Key_Scheme)) {
    for (options.df in split.data.frame(algo.df,algo.df$Key_Scheme_options)) {
      new.row <- data.frame(
            algo = options.df$Key_Scheme[1]
            ,key.dataset = options.df$Key_Dataset[1]
            ,acc = mean(options.df$Percent_correct)
            ,speed_test = mean(options.df$UserCPU_Time_testing)
            ,speed_train = mean(options.df$UserCPU_Time_training)
            ,TP = median(options.df$Num_true_positives)
            ,TN = median(options.df$Num_true_negatives)
            ,FP = median(options.df$Num_false_positives)
            ,FN = median(options.df$Num_false_negatives)
            ,key.scheme_options = options.df$Key_Scheme_options[1]
            )
```

```
          confusion.df.all <- rbind(confusion.df.all,new.row)
    }
  }
}

confusion.df.all
```

```
##                                  algo                 key.dataset      acc
## 1    weka.classifiers.bayes.NaiveBayes   selected_data_logged_scaled 14.83333
## 2            weka.classifiers.lazy.IBk   selected_data_logged_scaled 35.91667
## 3          weka.classifiers.rules.OneR   selected_data_logged_scaled 21.66667
## 4         weka.classifiers.rules.ZeroR   selected_data_logged_scaled 40.83333
## 5           weka.classifiers.trees.J48   selected_data_logged_scaled 31.58333
## 6  weka.classifiers.trees.RandomForest   selected_data_logged_scaled 25.00000
## 7    weka.classifiers.bayes.NaiveBayes selected_data_logged_scaled_1 29.75000
## 8            weka.classifiers.lazy.IBk selected_data_logged_scaled_1 34.25000
## 9          weka.classifiers.rules.OneR selected_data_logged_scaled_1 20.16667
## 10        weka.classifiers.rules.ZeroR selected_data_logged_scaled_1 40.83333
## 11          weka.classifiers.trees.J48 selected_data_logged_scaled_1 31.33333
## 12 weka.classifiers.trees.RandomForest selected_data_logged_scaled_1 26.00000
## 13   weka.classifiers.bayes.NaiveBayes selected_data_logged_scaled_2 30.25000
## 14           weka.classifiers.lazy.IBk selected_data_logged_scaled_2 35.58333
## 15         weka.classifiers.rules.OneR selected_data_logged_scaled_2 20.16667
## 16        weka.classifiers.rules.ZeroR selected_data_logged_scaled_2 40.83333
## 17          weka.classifiers.trees.J48 selected_data_logged_scaled_2 30.83333
## 18 weka.classifiers.trees.RandomForest selected_data_logged_scaled_2 23.66667
## 19   weka.classifiers.bayes.NaiveBayes selected_data_logged_scaled_3 31.91667
## 20           weka.classifiers.lazy.IBk selected_data_logged_scaled_3 37.83333
## 21         weka.classifiers.rules.OneR selected_data_logged_scaled_3 20.16667
## 22        weka.classifiers.rules.ZeroR selected_data_logged_scaled_3 40.83333
## 23          weka.classifiers.trees.J48 selected_data_logged_scaled_3 30.83333
## 24 weka.classifiers.trees.RandomForest selected_data_logged_scaled_3 23.58333
## 25   weka.classifiers.bayes.NaiveBayes selected_data_logged_scaled_4 31.83333
## 26           weka.classifiers.lazy.IBk selected_data_logged_scaled_4 25.50000
## 27         weka.classifiers.rules.OneR selected_data_logged_scaled_4 23.08333
## 28        weka.classifiers.rules.ZeroR selected_data_logged_scaled_4 40.83333
## 29          weka.classifiers.trees.J48 selected_data_logged_scaled_4 29.25000
## 30 weka.classifiers.trees.RandomForest selected_data_logged_scaled_4 24.16667
## 31   weka.classifiers.bayes.NaiveBayes selected_data_logged_scaled_5 36.33333
## 32           weka.classifiers.lazy.IBk selected_data_logged_scaled_5 27.91667
## 33         weka.classifiers.rules.OneR selected_data_logged_scaled_5 22.75000
## 34        weka.classifiers.rules.ZeroR selected_data_logged_scaled_5 40.83333
## 35          weka.classifiers.trees.J48 selected_data_logged_scaled_5 31.00000
## 36 weka.classifiers.trees.RandomForest selected_data_logged_scaled_5 26.41667
## 37   weka.classifiers.bayes.NaiveBayes selected_data_logged_scaled_6 37.91667
## 38           weka.classifiers.lazy.IBk selected_data_logged_scaled_6 28.83333
## 39         weka.classifiers.rules.OneR selected_data_logged_scaled_6 23.00000
## 40        weka.classifiers.rules.ZeroR selected_data_logged_scaled_6 40.83333
## 41          weka.classifiers.trees.J48 selected_data_logged_scaled_6 31.08333
## 42 weka.classifiers.trees.RandomForest selected_data_logged_scaled_6 26.16667
##    speed_test speed_train  TP TN FP FN
## 1       0e+00     0.00000 0.0  1  2  1
## 2       0e+00     0.00000 1.0  1  1  1
## 3       0e+00     0.00020 0.0  1  1  1
## 4       1e-04     0.00010 2.0  0  2  0
## 5       1e-04     0.00020 0.0  1  1  1
## 6       2e-04     0.00900 0.0  1  2  1
## 7       1e-04     0.00010 0.0  2  0  2
## 8       1e-04     0.00010 1.0  1  1  1
## 9       0e+00     0.00010 0.0  1  1  1
## 10      0e+00     0.00010 2.0  0  2  0
## 11      0e+00     0.00030 0.0  1  1  1
## 12      0e+00     0.00630 0.0  1  2  1
## 13      1e-04     0.00000 0.0  2  0  2
## 14      0e+00     0.00000 1.0  1  1  1
## 15      0e+00     0.00020 0.0  1  1  1
## 16      0e+00     0.00000 2.0  0  2  0
## 17      0e+00     0.00030 0.0  1  1  1
## 18      2e-04     0.00840 0.0  1  2  1
## 19      0e+00     0.00000 0.0  2  0  2
## 20      5e-05     0.00000 1.0  1  1  1
## 21      0e+00     0.00010 0.0  1  1  1
## 22      0e+00     0.00000 2.0  0  2  0
## 23      0e+00     0.00030 0.0  1  1  1
## 24      0e+00     0.00800 0.0  1  2  1
## 25      0e+00     0.00000 0.0  2  0  2
## 26      1e-04     0.00000 0.0  1  1  1
## 27      0e+00     0.00000 0.0  1  1  1
## 28      0e+00     0.00000 2.0  0  2  0
## 29      0e+00     0.00030 0.0  1  1  1
## 30      0e+00     0.00700 0.0  1  2  1
## 31      0e+00     0.00010 0.0  2  0  2
## 32      0e+00     0.00005 1.0  1  1  1
## 33      0e+00     0.00010 0.0  1  1  1
## 34      0e+00     0.00000 2.0  0  2  0
## 35      0e+00     0.00030 0.0  1  1  1
## 36      0e+00     0.00740 0.0  1  2  1
## 37      0e+00     0.00000 0.0  2  0  2
## 38      0e+00     0.00000 1.0  1  1  1
## 39      0e+00     0.00010 0.0  1  1  1
## 40      0e+00     0.00000 2.0  0  2  0
## 41      0e+00     0.00030 0.5  1  1  1
## 42      1e-04     0.00630 0.0  1  1  1
##
## 1
## 2  '-K 1 -W 0 -A \\weka.core.neighboursearch.LinearNNSearch -A \\\\\\weka.core.EuclideanDistance -R
## 3
## 4
## 5
## 6                                                                  '-P 100 -I 100 -num-slots 1 -K 0 -M
## 7
## 8  '-K 1 -W 0 -A \\weka.core.neighboursearch.LinearNNSearch -A \\\\\\weka.core.EuclideanDistance -R
## 9
## 10
## 11
## 12                                                                 '-P 100 -I 100 -num-slots 1 -K 0 -M
## 13
## 14 '-K 1 -W 0 -A \\weka.core.neighboursearch.LinearNNSearch -A \\\\\\weka.core.EuclideanDistance -R
## 15
## 16
## 17
## 18                                                              '-P 100 -I 100 -num-slots 1 -K 0 -M
## 19
## 20 '-K 1 -W 0 -A \\weka.core.neighboursearch.LinearNNSearch -A \\\\\\weka.core.EuclideanDistance -R
## 21
## 22
## 23
## 24                                                              '-P 100 -I 100 -num-slots 1 -K 0 -M
## 25
## 26 '-K 1 -W 0 -A \\weka.core.neighboursearch.LinearNNSearch -A \\\\\\weka.core.EuclideanDistance -R
## 27
## 28
## 29
## 30                                                              '-P 100 -I 100 -num-slots 1 -K 0 -M
## 31
## 32 '-K 1 -W 0 -A \\weka.core.neighboursearch.LinearNNSearch -A \\\\\\weka.core.EuclideanDistance -R
## 33
## 34
## 35
## 36                                                              '-P 100 -I 100 -num-slots 1 -K 0 -M
## 37
## 38 '-K 1 -W 0 -A \\weka.core.neighboursearch.LinearNNSearch -A \\\\\\weka.core.EuclideanDistance -R
## 39
## 40
## 41
## 42                                                              '-P 100 -I 100 -num-slots 1 -K 0 -M
```

```
# for (dataset.df in split.data.frame(confusion.df.all,confusion.df.all$key.dataset)) {
#   datatable(dataset.df)
# }
```

Figure 10: Shows the performance of all algorithms tested on all data sets. Row 51 shows the best performing model.

```
dataset_clean <- dataset_clean[-c(21,39,40), ]
dataset_clean$PeakEosBaseline_Max <- cut(as.matrix(peak.eos.max), breaks = c(0,49,75,200), labels = c("
#logged.dataset.full <- cbind(log2(dataset_clean[1:12]),dataset_clean[13:17], by="pid")
```