Research Log Project Machine Learning

Diagnosing malignancy of breast masses using Machine Learning

Student: Vincent Talen

Student number: 389015

Class: BFV3

Study: Bioinformatics

Institute: Institute for Life Science & Technology

Teachers: Dave Langers (LADR) and Bart Barnard (BABA)

Date: 2022-10-30

Table of Contents

List of Figures

List of Tables

1 Preparing R environment

Set some options for the code chunks here so they don't have to be set for every chunk separately.
Also set the directory this markdown file knits and runs from to the project directory.

```
# Set code chunk options
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(cache = TRUE)
knitr::opts_chunk$set(warning = FALSE)
knitr::opts_chunk$set(fig.align = "center")
knitr::opts_chunk$set(fig.path = here::here("output/figures/"))
knitr::opts_chunk$set(cache.path = here::here("src/rmd/research_log_cache/"))

# Set directory of this Rmd file to project directory
knitr::opts_knit$set(root.dir = here::here())
```

For the data analysis and further processes multiple libraries are needed, they are loaded in here. A few other options/settings are configured and used scripts also loaded.

```
# Load required packages from vector using invisible and lapply
packages <- c("tidyverse", "pander", "data.table", "RWeka",
              "ggplot2", "ggpubr", "ggbiplot")
invisible(lapply(packages, library, character.only = TRUE))
remove(packages) # Drop variable since it will not be used again

# Source functions
source("src/scripts/split_violin_plot.R")
source("src/scripts/weka_analyser_custom.R")

# Disable printing 'table continues' lines between split sections
pander::panderOptions("table.continues", "")
# Change affix after table caption if it's a split table
pander::panderOptions("table.continues.affix", "(table continues below)")
```

2 Exploratory Data Analysis

2.1 About the chosen dataset

The dataset that is used is the *Wisconsin Breast Cancer (Diagnostic) Dataset*, which is publicly available from the UCI Machine Learning Repository. There are two published research articles, from the same team of researchers, where the dataset was first used, namely [1] and [2]. The samples for the data were collected from 569 patients at the University of Wisconsin Hospital with the goal of creating a machine learning model that was faster, improved the correctness and increased the objectivity of the diagnosis process of breast cancer.

Collection of the data

The data was gathered by first collecting the fine needle aspirates (FNA), which are expressed on a glass slide and stained. The images were generated by a color video camera mounted on top of a microscope, that projected the images with a 63x objective and 2.5x ocular into the camera. The images were then captured by a color frame grabber board as a 512x480, 8-bit-per-pixel Targa file.
The digitized image is then analyzed in the program Xcyt (custom made by Nick Street). First the user marks approximate initial boundaries of the nuclei and then the actual boundaries are further defined with an active contour model known as "Snake". In the end the snake reaches a point where it's curve accurately corresponds to the boundary of a cell nucleus. From the snake-generated cell nuclei boundaries 10 features are extracted, these are numerically modeled such that larger values will typically indicate a higher likelihood of malignancy.

The ten features that are extracted for each cell nucleus are the following:

1. Radius (mean of distances from center to points on the perimeter)
2. Texture (standard deviation of gray-scale values)
3. Perimeter (the total distance between all the points of the snake-generated boundary)
4. Area (the nuclear area is the sum of pixels on the interior, with half of the pixels of the perimeter)
5. Smoothness (local variation in radius lengths)
6. Compactness (perimeter^2 / area - 1.0)
7. Concavity (severity of concave portions of the contour)
8. Concave points (number of concave portions of the contour)
9. Symmetry (difference in length of perpendicular lines to the longest chord through the center, in both directions)
10. Fractal dimension (approximated using Mandelbrot's "coastline approximation" - 1)

For every image, three final values were computed for each feature and saved to the dataset, namely the mean, standard error and the extreme (largest) value.

Data structure and codebook

The dataset has 569 instances/rows with 32 columns, an ID column, a classification column with the diagnosis (benign or malignant) and 30 columns describing the nuclei boundaries (10x mean/extreme/se).
Because the dataset itself does not come with an annotated header with column names, a codebook has been manually made. This codebook has the abbreviated column name, the full column name, the data type and a description for each feature/column.

Below is an overview of the columns in the dataset, shown using the contents of the codebook after it has been loaded in:

```r
codebook <- readr::read_delim("data/raw/codebook.txt", delim = "|", show_col_types = FALSE)

# Pretty print the codebook (without descriptions) using pander
pander::pander(codebook[, 1:3], style = "rmarkdown",
               caption = "Overview of created codebook excluding descriptions")
```

Table 1: Overview of created codebook excluding descriptions

Column Name	Full Name	Type
id	ID	dbl
diagnosis	Diagnosis	fct
radius_mean	Mean Radius	dbl
texture_mean	Mean Texture	dbl
perimeter_mean	Mean Perimeter	dbl
area_mean	Mean Area	dbl
smoothness_mean	Mean Smoothness	dbl
compactness_mean	Mean Compactness	dbl
concavity_mean	Mean Concavity	dbl
concave_pts_mean	Mean Concave Points	dbl
symmetry_mean	Mean Symmetry	dbl
fractal_dim_mean	Mean Fractal Dimension	dbl
radius_se	Radius SE	dbl
texture_se	Texture SE	dbl
perimeter_se	Perimeter SE	dbl
area_se	Area SE	dbl
smoothness_se	Smoothness SE	dbl
compactness_se	Compactness SE	dbl
concavity_se	Concavity SE	dbl
concave_pts_se	Concave Points SE	dbl
symmetry_se	Symmetry SE	dbl
fractal_dim_se	Fractal Dimension SE	dbl
radius_worst	Worst Radius	dbl
texture_worst	Worst Texture	dbl
perimeter_worst	Worst Perimeter	dbl
area_worst	Worst Area	dbl
smoothness_worst	Worst Smoothness	dbl
compactness_worst	Worst Compactness	dbl
concavity_worst	Worst Concavity	dbl
concave_pts_worst	Worst Concave Points	dbl
symmetry_worst	Worst Symmetry	dbl
fractal_dim_worst	Worst Fractal Dimension	dbl

As can be seen, all the features are of the type double except the main diagnosis classification factor.

2.2 Loading in the dataset

The data will be loaded in with the `read_csv` function from the `readr` package, this function returns the data as a tibble data frame. This function allows a vector with column names to be given with an argument, the names from the column `Column Name` of the codebook will be used.

```
data <- readr::read_csv("data/raw/wdbc.data", col_names = codebook[[1]], show_col_types = FALSE)

# Print the amount of samples and columns
cat("Amount of samples:", dim(data)[1], "\tColumns in dataframe:", dim(data)[2], "\n")
```

```
## Amount of samples: 569   Columns in dataframe: 32
```

The amount of samples and columns are as expected, so in this aspect the dataset has been read correctly. However, what is more important is if the values are read correctly, since the values are that what is actually going to be used.

```
str(data)
```

```
## spec_tbl_df [569 x 32] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ id               : num [1:569] 842302 842517 84300903 84348301 84358402 ...
##  $ diagnosis        : chr [1:569] "M" "M" "M" "M" ...
##  $ radius_mean      : num [1:569] 18 20.6 19.7 11.4 20.3 ...
##  $ texture_mean     : num [1:569] 10.4 17.8 21.2 20.4 14.3 ...
##  $ perimeter_mean   : num [1:569] 122.8 132.9 130 77.6 135.1 ...
##  $ area_mean        : num [1:569] 1001 1326 1203 386 1297 ...
##  $ smoothness_mean  : num [1:569] 0.1184 0.0847 0.1096 0.1425 0.1003 ...
##  $ compactness_mean : num [1:569] 0.2776 0.0786 0.1599 0.2839 0.1328 ...
##  $ concavity_mean   : num [1:569] 0.3001 0.0869 0.1974 0.2414 0.198 ...
##  $ concave_pts_mean : num [1:569] 0.1471 0.0702 0.1279 0.1052 0.1043 ...
##  $ symmetry_mean    : num [1:569] 0.242 0.181 0.207 0.26 0.181 ...
##  $ fractal_dim_mean : num [1:569] 0.0787 0.0567 0.06 0.0974 0.0588 ...
##  $ radius_se        : num [1:569] 1.095 0.543 0.746 0.496 0.757 ...
##  $ texture_se       : num [1:569] 0.905 0.734 0.787 1.156 0.781 ...
##  $ perimeter_se     : num [1:569] 8.59 3.4 4.58 3.44 5.44 ...
##  $ area_se          : num [1:569] 153.4 74.1 94 27.2 94.4 ...
##  $ smoothness_se    : num [1:569] 0.0064 0.00522 0.00615 0.00911 0.01149 ...
##  $ compactness_se   : num [1:569] 0.049 0.0131 0.0401 0.0746 0.0246 ...
##  $ concavity_se     : num [1:569] 0.0537 0.0186 0.0383 0.0566 0.0569 ...
##  $ concave_pts_se   : num [1:569] 0.0159 0.0134 0.0206 0.0187 0.0188 ...
##  $ symmetry_se      : num [1:569] 0.03 0.0139 0.0225 0.0596 0.0176 ...
##  $ fractal_dim_se   : num [1:569] 0.00619 0.00353 0.00457 0.00921 0.00511 ...
##  $ radius_worst     : num [1:569] 25.4 25 23.6 14.9 22.5 ...
##  $ texture_worst    : num [1:569] 17.3 23.4 25.5 26.5 16.7 ...
##  $ perimeter_worst  : num [1:569] 184.6 158.8 152.5 98.9 152.2 ...
##  $ area_worst       : num [1:569] 2019 1956 1709 568 1575 ...
##  $ smoothness_worst : num [1:569] 0.162 0.124 0.144 0.21 0.137 ...
##  $ compactness_worst: num [1:569] 0.666 0.187 0.424 0.866 0.205 ...
##  $ concavity_worst  : num [1:569] 0.712 0.242 0.45 0.687 0.4 ...
##  $ concave_pts_worst: num [1:569] 0.265 0.186 0.243 0.258 0.163 ...
##  $ symmetry_worst   : num [1:569] 0.46 0.275 0.361 0.664 0.236 ...
##  $ fractal_dim_worst: num [1:569] 0.1189 0.089 0.0876 0.173 0.0768 ...
```

Luckily it looks like the values for every column have correctly been read, but there is one thing that could still be changed; the diagnosis column. It would be more helpful if diagnosis was a factor and not just a character column.

```
data$diagnosis <- factor(data$diagnosis, labels = c("Benign", "Malignant"))

ggplot(data, aes(x = diagnosis, fill = diagnosis)) + geom_bar() +
  geom_text(stat = "count", aes(label = after_stat(count)), nudge_y = -15) +
  scale_fill_hue(direction = 1, h.start = 180) +
  ggtitle("Barplot of diagnosis classification factor")
```

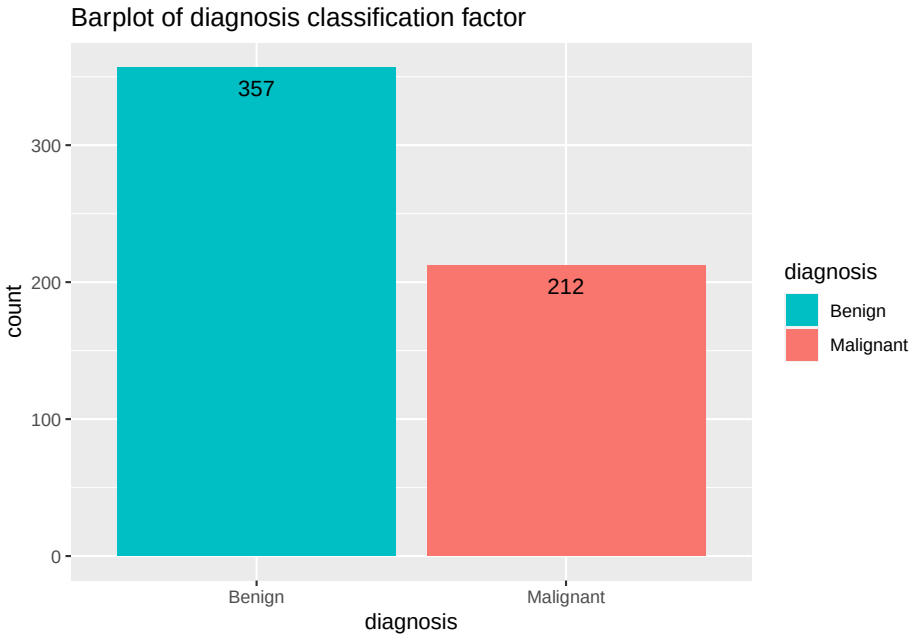

Figure 1: Barplot showing distribution of diagnosis classification labels

There are more benign cases than malignant, this means the dataset is not entirely balanced and could cause bias if not handled correctly.

The dataset has the `id` column, this column is not needed for the analysis that will be performed so it will therefore be dropped from the data frame. The `id` column could even cause small a hiccup when creating the machine learning model, since all the values are unique the model could use that as the only feature to predict the classification label. This would of course not work since unseen data will not have the same id's as the data used for training the machine learning algorithm.

```
data <- dplyr::select(data, -id)
cat( sprintf("ID column present?: %s", "id" %in% colnames(data)) )
```

```
## ID column present?: FALSE
```

2.3 Data inspection

It is important to have a good understanding of what the data is like, for example how the data is distributed and if there is data corruption. A few things that should be checked are if there are any outliers present, any data is skewed (an asymmetry in data distribution) or if there is any missing data.

```
cat("Missing values:", any(is.na(data)))
```

```
## Missing values: FALSE
```

Luckily there are no missing values in the dataset. But now it is good to get an idea of what the values of the columns look like, what ranges do their values fall in? This can be done with the function `summary()` for all columns at the same time, it will create a basic statistics overview about the columns.

Overall data summary

```
pander::pander(summary(data), caption = "Summary with basic statistics about the data colums")
```

Table 2: Summary with basic statistics about the data colums (table continues below)

diagnosis	radius_mean	texture_mean	perimeter_mean
Benign :357	Min. : 6.981	Min. : 9.71	Min. : 43.79
Malignant:212	1st Qu.:11.700	1st Qu.:16.17	1st Qu.: 75.17
NA	Median :13.370	Median :18.84	Median : 86.24
NA	Mean :14.127	Mean :19.29	Mean : 91.97
NA	3rd Qu.:15.780	3rd Qu.:21.80	3rd Qu.:104.10
NA	Max. :28.110	Max. :39.28	Max. :188.50

area_mean	smoothness_mean	compactness_mean	concavity_mean
Min. : 143.5	Min. :0.05263	Min. :0.01938	Min. :0.00000
1st Qu.: 420.3	1st Qu.:0.08637	1st Qu.:0.06492	1st Qu.:0.02956
Median : 551.1	Median :0.09587	Median :0.09263	Median :0.06154
Mean : 654.9	Mean :0.09636	Mean :0.10434	Mean :0.08880
3rd Qu.: 782.7	3rd Qu.:0.10530	3rd Qu.:0.13040	3rd Qu.:0.13070
Max. :2501.0	Max. :0.16340	Max. :0.34540	Max. :0.42680

concave_pts_mean	symmetry_mean	fractal_dim_mean	radius_se
Min. :0.00000	Min. :0.1060	Min. :0.04996	Min. :0.1115
1st Qu.:0.02031	1st Qu.:0.1619	1st Qu.:0.05770	1st Qu.:0.2324
Median :0.03350	Median :0.1792	Median :0.06154	Median :0.3242
Mean :0.04892	Mean :0.1812	Mean :0.06280	Mean :0.4052
3rd Qu.:0.07400	3rd Qu.:0.1957	3rd Qu.:0.06612	3rd Qu.:0.4789
Max. :0.20120	Max. :0.3040	Max. :0.09744	Max. :2.8730

texture_se	perimeter_se	area_se	smoothness_se
Min. :0.3602	Min. : 0.757	Min. : 6.802	Min. :0.001713
1st Qu.:0.8339	1st Qu.: 1.606	1st Qu.: 17.850	1st Qu.:0.005169
Median :1.1080	Median : 2.287	Median : 24.530	Median :0.006380
Mean :1.2169	Mean : 2.866	Mean : 40.337	Mean :0.007041
3rd Qu.:1.4740	3rd Qu.: 3.357	3rd Qu.: 45.190	3rd Qu.:0.008146
Max. :4.8850	Max. :21.980	Max. :542.200	Max. :0.031130

compactness_se	concavity_se	concave_pts_se	symmetry_se
Min. :0.002252	Min. :0.00000	Min. :0.000000	Min. :0.007882
1st Qu.:0.013080	1st Qu.:0.01509	1st Qu.:0.007638	1st Qu.:0.015160
Median :0.020450	Median :0.02589	Median :0.010930	Median :0.018730
Mean :0.025478	Mean :0.03189	Mean :0.011796	Mean :0.020542
3rd Qu.:0.032450	3rd Qu.:0.04205	3rd Qu.:0.014710	3rd Qu.:0.023480
Max. :0.135400	Max. :0.39600	Max. :0.052790	Max. :0.078950

fractal_dim_se	radius_worst	texture_worst	perimeter_worst
Min. :0.0008948	Min. : 7.93	Min. :12.02	Min. : 50.41
1st Qu.:0.0022480	1st Qu.:13.01	1st Qu.:21.08	1st Qu.: 84.11
Median :0.0031870	Median :14.97	Median :25.41	Median : 97.66
Mean :0.0037949	Mean :16.27	Mean :25.68	Mean :107.26
3rd Qu.:0.0045580	3rd Qu.:18.79	3rd Qu.:29.72	3rd Qu.:125.40
Max. :0.0298400	Max. :36.04	Max. :49.54	Max. :251.20

area_worst	smoothness_worst	compactness_worst	concavity_worst
Min. : 185.2	Min. :0.07117	Min. :0.02729	Min. :0.0000
1st Qu.: 515.3	1st Qu.:0.11660	1st Qu.:0.14720	1st Qu.:0.1145
Median : 686.5	Median :0.13130	Median :0.21190	Median :0.2267
Mean : 880.6	Mean :0.13237	Mean :0.25427	Mean :0.2722
3rd Qu.:1084.0	3rd Qu.:0.14600	3rd Qu.:0.33910	3rd Qu.:0.3829
Max. :4254.0	Max. :0.22260	Max. :1.05800	Max. :1.2520

concave_pts_worst	symmetry_worst	fractal_dim_worst
Min. :0.00000	Min. :0.1565	Min. :0.05504
1st Qu.:0.06493	1st Qu.:0.2504	1st Qu.:0.07146
Median :0.09993	Median :0.2822	Median :0.08004
Mean :0.11461	Mean :0.2901	Mean :0.08395
3rd Qu.:0.16140	3rd Qu.:0.3179	3rd Qu.:0.09208
Max. :0.29100	Max. :0.6638	Max. :0.20750

By glancing over the summary created above a few things can be noticed and questions can arise, for example the `area_mean`, `concave_pts_mean`, `radius_se`, `perimeter_se` and `area_worst` columns. These, among other columns, have a wide range of values with their minimum or maximum values far from the quantiles or median.
This could mean that there are outliers in the data, the questions that arise because of this is if these points are actually outliers and if they should be excluded from the data. It can however not be determined with just the summary above if these are actually outliers and if they need to be removed.

2.4 Univariate analysis

To check if points are outliers the data needs to be visualized, this can be done by creating a box plot and/or density plot for each of the columns. Creating them separately for all 30 feature columns would result in 60 plots, which is a lot and perhaps too many. To reduce the amount of total plots, the density plots will instead be visualized using violin plots with the box plots inside of them. Because printing all the data of a feature column in a single box or violin does not show what is desired, the plots will be split on the diagnosis classification factor.

There is not a function in `ggplot` to create these split violin plots, the source code file `src/scripts/split_violin_plot.R` contains a function that uses ggplot as a basis to create split violin plots. Then, using this function from the source file, another function is created that assembles the full plot including the box plot, plot title and themes. After this a list of column names can be given of which a plot will be generated for each and then the plots will be arranged with a common legend to the output file. So there is a clearer overview where the data can be compared the plots for the feature columns will be split on their specification, resulting in three arranged layouts with the mean-, standard error- and worst plots.

```
getColumnFullName <- function(col_name) {
  return(dplyr::filter(codebook, `Column Name` == col_name)$`Full Name`)
}

createFeaturePlot <- function(col_name, figure_tab) {
  plot <- ggplot(data, aes(x = "", y = !!sym(col_name), fill = diagnosis)) +
    geom_split_violin(alpha = 0.6, trim = FALSE) +
    geom_boxplot(width = 0.2, alpha = 0.6, fatten = NULL, show.legend = F) +
    stat_summary(fun.data = "mean_se", geom = "pointrange", show.legend = F,
                 position = position_dodge(0.2), size = 0.3) +
    scale_fill_brewer(palette = "Dark2", name = "Diagnosis:") +
    ggtitle( getColumnFullName(col_name) ) +
    theme_minimal() + labs(y = NULL, x = NULL, tag = figure_tab) +
    theme(plot.title = element_text(size = 9, face = "bold"))
  return(list(plot))
}

createPlotGrid <- function(extension) {
  desired_col_names <- colnames(data)[colnames(data) %like% extension]
  figure_tabs <- letters[1:length(desired_col_names)]
  # Create plots and put them in a list
  plot_list <- mapply(createFeaturePlot, desired_col_names, figure_tabs)
  # Print the plots in an arranged grid with the legend at the bottom
  ggpubr::ggarrange(plotlist = plot_list, ncol = 4, nrow = 3,
                    common.legend = TRUE, legend = "bottom")
}
```

With these functions the spits violin plots can easily be made as three plot grids with the `createPlotGrid()` function, split by the three nuclei features. Then the grids will be annotated with a plot title and then they are done.

```
# Create split violin plots for all mean feature columns
ggpubr::annotate_figure(
  createPlotGrid("_mean"),
  top = text_grob("Grid of violin plots with boxplots for each 'mean' feature column",
                  face = "bold", size = 14))
```

```
# Create split violin plots for all standard error feature columns
ggpubr::annotate_figure(
  createPlotGrid("_se"),
  top = text_grob("Grid of violin plots with boxplots for each 'standard error' feature column",
                  face = "bold", size = 14))
```

```
# Create split violin plots for all worst/extreme feature columns
ggpubr::annotate_figure(
  createPlotGrid("_worst"),
  top = text_grob("Grid of violin plots with boxplots for each 'worst' feature column",
                  face = "bold", size = 14))
```

Looking at the resulting plots it can easily be seen if there is a distinctive correlation between the classification factor and the values of feature columns. The couple of columns that stand out the most are the `Radius`, `Perimeter`, `Area`, `Concavity` and `Concave Points` feature columns, these all seem to have a clear distinction between the diagnosis classification. One thing that should be noted is that in most of the standard error columns a lot of outliers can be seen, for now nothing will be done with these but it is a good thing to know might problems arise later on.

Now just because it *looks* like there is a clear distinction it needs to be made sure that the difference between the class labels for the feature is actually significant. This can be tested with a 1-way ANOVA test and will be done for the radius_mean and texture_se to show the difference between significance and no significance.

```
pval_radius_mean <- summary(aov(radius_mean ~ diagnosis, data = data))[[1]][1,5]
pval_texture_se <-summary(aov(texture_se ~ diagnosis, data = data))[[1]][1,5]

cat("P-Value of radius_mean =", pval_radius_mean,
    "\nP-Value of texture_se  =", pval_texture_se)
```

```
## P-Value of radius_mean = 8.465941e-96
## P-Value of texture_se  = 0.843332
```

An alpha of 0.05 is used for the 1-way ANOVA test and when looking at the resulting p-values it is just as expected, the mean radius is significant and the texture standard error is not. So the chance of texture_se being used in an efficient machine learning model is very slim but radius_mean being used is very likely, unless there is a heavily correlated feature column with higher information gain.

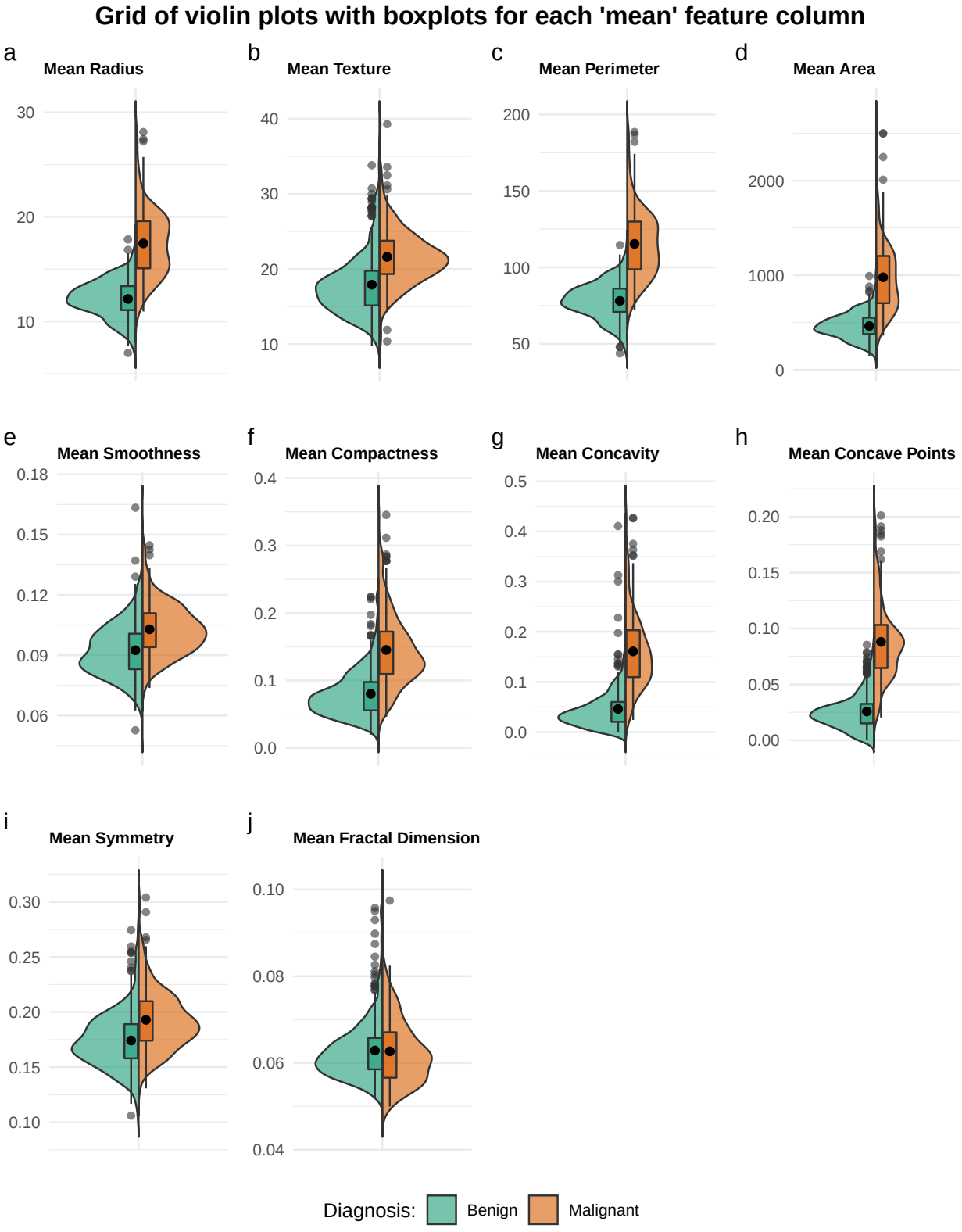

Figure 2: Split violin plots showing density joined with box plots showing quartiles and outliers, for all 'mean' feature columns

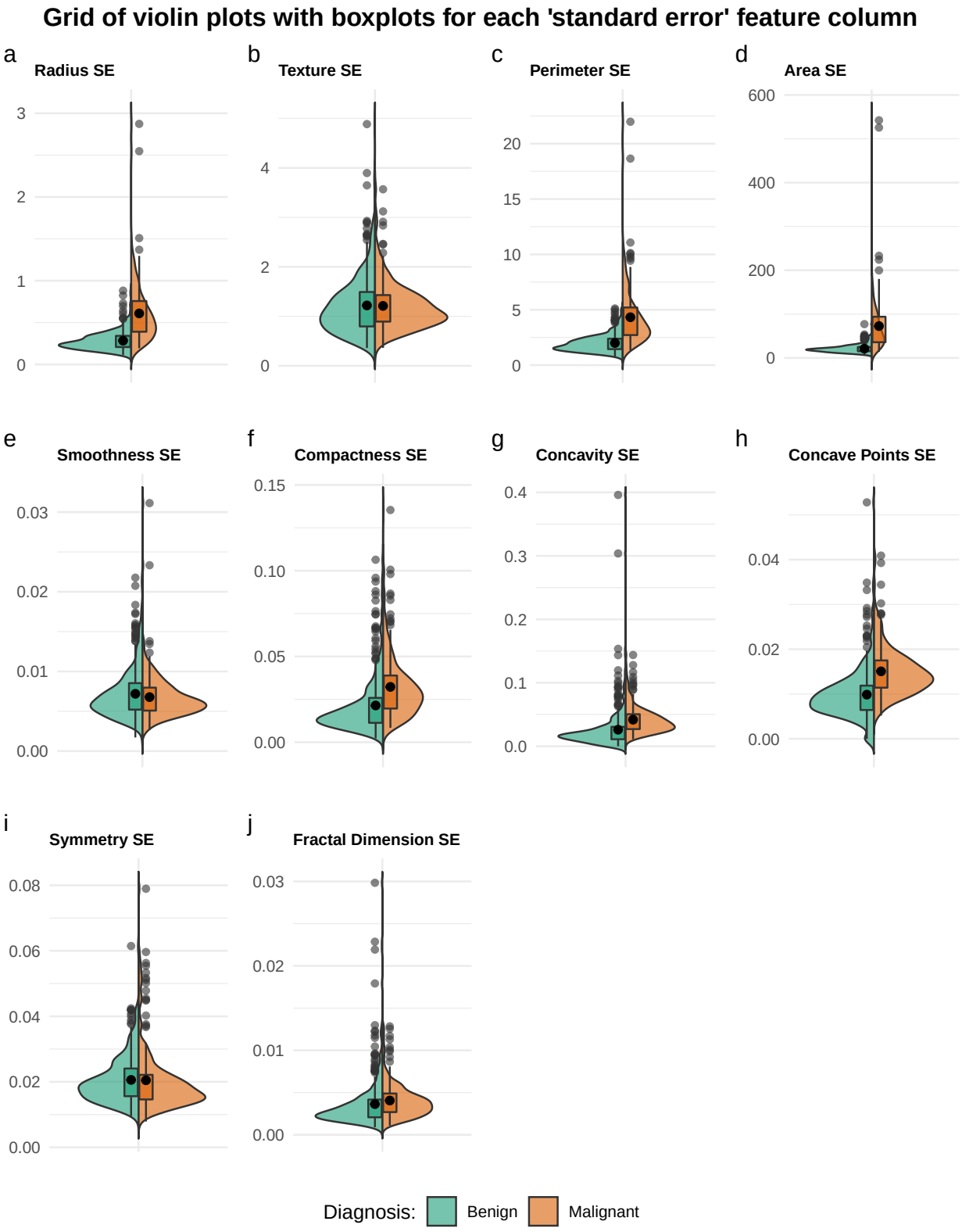

Figure 3: Split violin plots showing density joined with box plots showing quartiles and outliers, for all 'standard error' feature columns

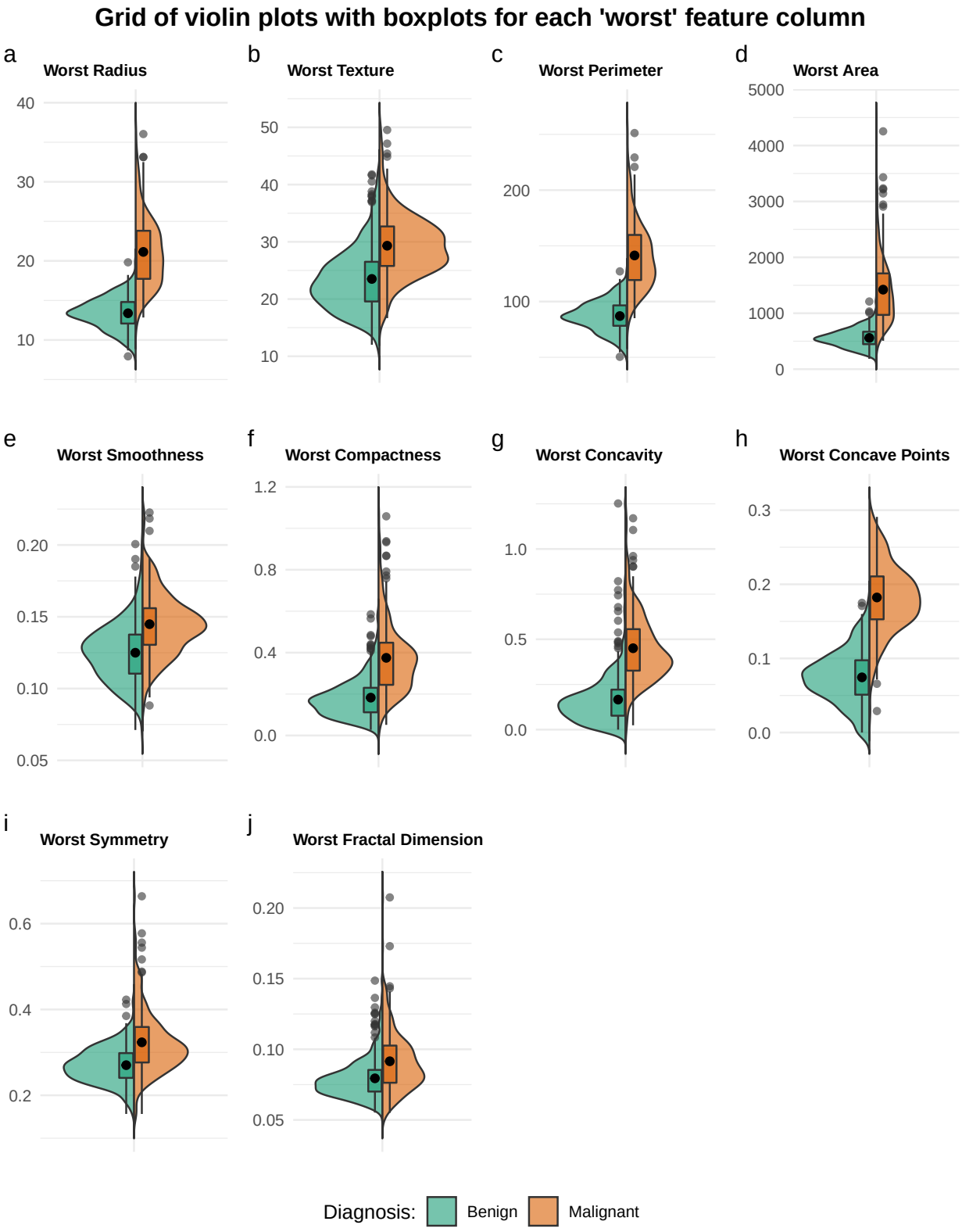

Figure 4: Split violin plots showing density joined with box plots showing quartiles and outliers, for all 'worst/extreme' feature columns

2.5 Multivariate analysis

A machine learning model should be kept as simple as possible whilst keeping the accuracy still high, this is so model does not become overfitted to the data used for training. Overfitting causes the model to be less accurate on unknown data because specific combinations of many features in the training data might not exist in the test data. So only the couple of features that correlate the highest with the classification factor are desired. But when two features correlate very highly to *each other* and if they are then both used for the model, then these features together does not improve the model any more than if only one of them was used. Because of this it is desired to identify correlations between features because one of the features from the pair should at least not be used for the model.

Heatmap of correlation matrix

To quickly identify the correlations between all features a correlation matrix is made where all possible pairs of features have their correlation calculated. Correlations can be visualized in a heatmap, where the strength of correlations is shown with a color gradient so they can easily be visually seen.

```
# Create correlation matrix with only numerical columns and insert feature name column
cor_mat <- tibble::as_tibble( stats::cor(select(data, -diagnosis)) ) %>%
  dplyr::mutate(col_names = all_of(colnames(.))) %>%
  dplyr::select(31, 1:30)
# Show first four columns of correlation matrix
pander::pander(head(cor_mat[1:5], n = 4), caption = "Head of wide correlation matrix")
```

Table 10: Head of wide correlation matrix

col_names	radius_mean	texture_mean	perimeter_mean	area_mean
radius_mean	1	0.3238	0.9979	0.9874
texture_mean	0.3238	1	0.3295	0.3211
perimeter_mean	0.9979	0.3295	1	0.9865
area_mean	0.9874	0.3211	0.9865	1

Before the correlation matrix can be used to create a heatmap it needs to be converted to long format, this is because of the way R and ggplot read and use data.

```
# Convert data to long format so a heatmap can be made from it
cor_mat_long <- tidyr::pivot_longer(data = cor_mat, cols = all_of(colnames(cor_mat[-1])),
                                    names_to = "variable", values_to = "pair_cor")
# Show what the data looks like now
pander::pander(head(cor_mat_long, n = 4), caption = "Head of long correlation matrix")
```

Table 11: Head of long correlation matrix

col_names	variable	pair_cor
radius_mean	radius_mean	1
radius_mean	texture_mean	0.3238
radius_mean	perimeter_mean	0.9979
radius_mean	area_mean	0.9874

In the matrix the features were not physically paired, now in the long format the feature pairs are saved as rows with their correlation score. This way the first two columns, each with the name of a feature of the pair, can be used as the axes of the heatmap plot.

```
ggplot(data = cor_mat_long, aes(x = col_names, y = variable, fill = pair_cor)) +
  geom_tile() + labs(x = NULL, y = NULL) +
  scale_fill_gradient(high = "purple", low = "white" ) +
  theme(axis.text.x = element_text(angle = 45, hjust=1)) +
  ggtitle("Heatmap of column correlations")
```

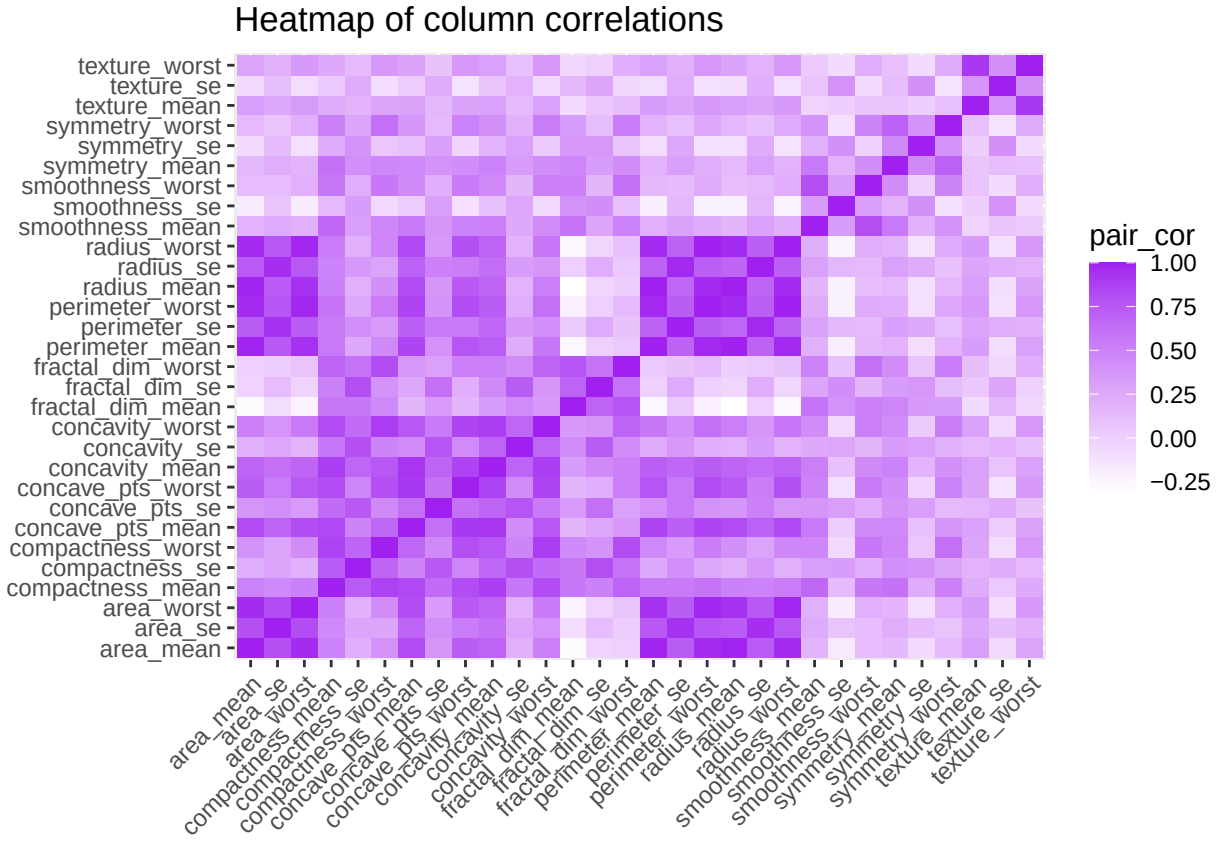

Figure 5: Heatmap visualizing pairwise correlation matrix of all feature colums

There are two groups of three nuclei features that highly correlate together on all three of their feature columns (mean, standard error and worst). The first group consists of the radius, perimeter and area nuclei features and the second group consists of the compactness, concavity and concave points features. For both groups only one of the mean, standard error and worst feature columns should be used, so for example only the area- and only the concavity feature columns.
Another thing that can immediately be noticed and should not be surprising is that the mean-, standard error- and worst feature columns of each nuclei boundary feature, i.e. of the area, also have a higher correlation to each other. So putting together that the area, radius and perimeter features all correlate to each other and themselves only one of the 9 feature columns will probably be used for the model, with the same being the case for the other group.

There are a few other feature columns that correlate highly together but since there are only 30 total the machine learning algorithms will decide the final ones to be used. So no feature columns will actually be

dropped from the data right now, but to show what a correlation between features looks like, two scatterplots will be made of radius_mean plotted against perimeter_mean and fractal_dim_mean.

```
createScatterPlot <- function(plot_title, feature_two) {
  ggplot(data, aes(x = radius_mean, y = !!sym(feature_two), color = diagnosis)) +
    labs(x = "Mean Radius", y = getColumnFullName(feature_two)) +
    geom_jitter(width=0.2, height=0.2, alpha=0.5, shape=16, size=0.8,
                mapping = aes(color = diagnosis)) +
    geom_smooth(formula = y ~ x, method = "loess") + ggtitle(plot_title) +
    scale_color_hue(direction = 1, h.start = 180)
}
scatter1 <- createScatterPlot("Plot of correlated features", "perimeter_mean")
scatter2 <- createScatterPlot("Plot of non-correlated features", "fractal_dim_mean")
my_grid <- ggpubr::ggarrange(scatter1, scatter2, common.legend = T, legend = "bottom")

ggpubr::annotate_figure(my_grid,
  top = text_grob("Scatterplots with trendlines showing the difference\nbetween correlated and non-corre
                  face = "bold", size = 14))
```

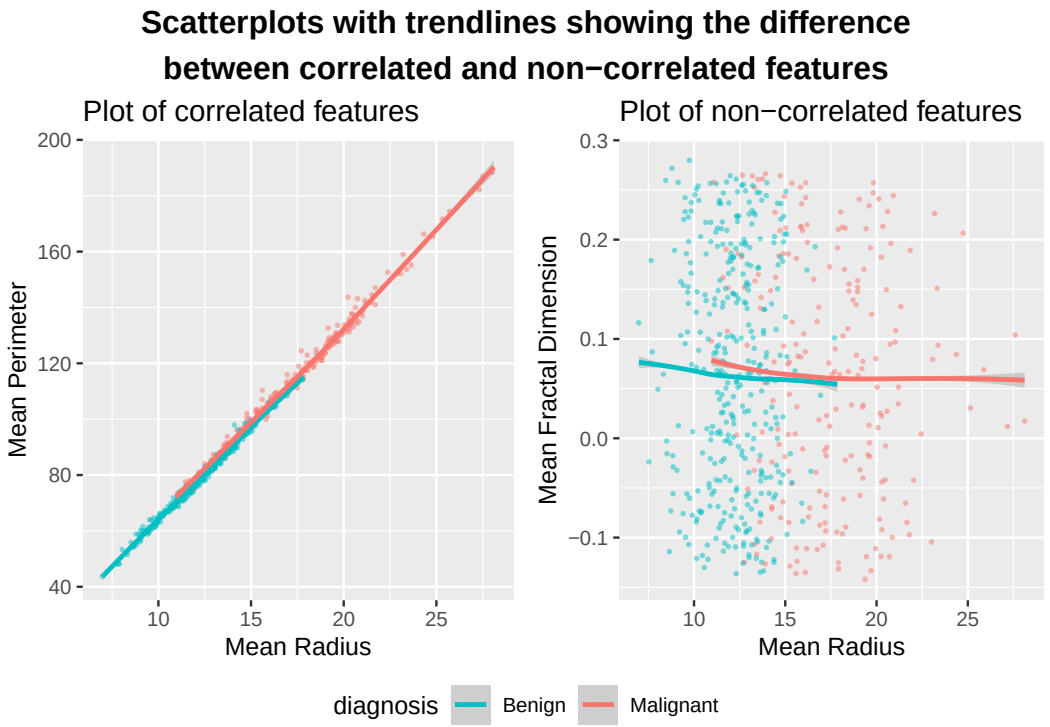

Figure 6: Scatterplots with trendlines of the Mean Radius and Mean Perimeter features and the Mean Radius and Mean Fractal Dimension features. Showing the difference in data point distribution between correlated and non-correlated features.

As can be seen in *Figure 6* , the points of Mean Radius and Mean Perimeter have a clear trend line drawn through them; when the radius increases, so does the perimeter by a certain amount. With one of them the other can be predicted, because of this they do not give a different insight or more information on how to classify an instance. In the second scatterplot of the Mean Radius and the Mean Fractal Dimension it is the opposite, there is no clear pattern showing that if the mean radius increases it is correlated to a linear increase in the mean fractal dimension.

Principal Component Analysis

It is helpful to see how and if the data clusters, by creating scatterplots to see the relation between the data points. This is however impossible to plot with the 30 feature columns, or rather dimensions, there are in the dataset. With only two features a two-dimensional plot could be made, but to create a 30-dimensional plot is not possible. It is possible to create lots of plots for all the feature pairs, but this is also not helpful and realistic.
One of the ways to be able to plot data on a 2D scale to see how the data is related is by creating a Principal Component Analysis (PCA) plot, where the dimensions are reduced by converting the correlations among all samples together into Principal Components. PCA starts by finding the best fitting line by maximizing the sum of squares from the projected points to the origin, which is called Principal Component 1 (PC1). After PC1 is found the next best fitting line that also goes through the origin but is perpendicular to PC1, this line is then subsequently called PC2. This continues until the PCs have been found for all features/dimensions, each going through the origin and being perpendicular to the previous principal components. Then, with all the PCs, the proportion of variance that each PC accounts for can be calculated.

```
# PCA of data except diagnosis column
pca_res <- stats::prcomp(data[-1], center = TRUE, scale = TRUE)

# Calculate variance explained from sdev
pc_names <- sprintf("PC%s", seq(1:length(pca_res$sdev)))
var_explained <- data.frame(pc = factor(pc_names, levels = pc_names),
                            var = pca_res$sdev^2 / sum(pca_res$sdev^2) )

# Create barplot of first 10 PCs with variance converted to %
ggplot(var_explained[1:10,], aes(x = pc, y = var, label = scales::percent(var))) +
  geom_bar(stat = "identity", fill = "darkturquoise") +
  labs(x = "", y = "Variance Explained",
       title = "Barplot showing the variances explained by first 10 PCs") +
  scale_y_continuous(labels = scales::percent) + geom_text(nudge_y = 0.015)
```

```
# Print total percentage covered by first two and first six PCs
cat(sprintf(" Variance explained by PC1 and PC2: %.1f%% \n", sum(var_explained$var[1:2])*100),
    sprintf( "Variance explained by PC1 through PC6: %.1f%%", sum(var_explained$var[1:6])*100))
```

```
##  Variance explained by PC1 and PC2: 63.2%
##  Variance explained by PC1 through PC6: 88.8%
```

A decent amount of variance is explained by just the first two principle components, with the first six explaining most variance. In *Figure 7* the variance explained by each of the first 10 PCs can be seen.

```
ggbiplot::ggbiplot(pca_res, obs.scale = 1, var.scale = 1,
                   groups = data$diagnosis, ellipse = FALSE, circle = TRUE) +
  scale_color_discrete(name = "Diagnosis") + ggtitle("Principal Component Analysis Plot")
```

In the PCA plot of *Figure 8* the first two PCs can be seen and show a good clustering by diagnosis of the data on the PC1 axis. It can also be seen that there are multiple features that have their directions overlapping, this is because of what already has been discussed, namely that there is a high correlation between those features.

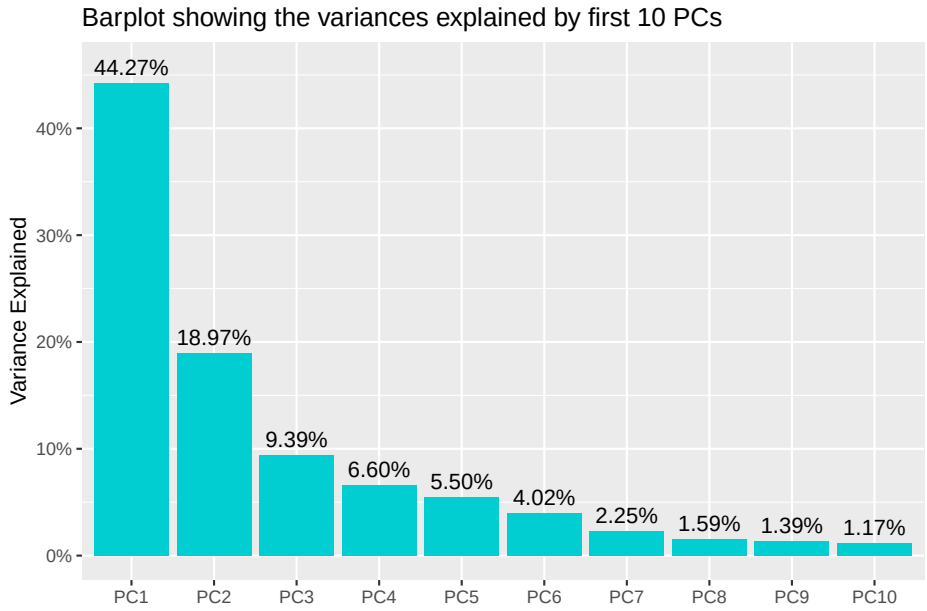

Figure 7: Barplot showing how much variance is explained by each of the first ten principle components of the PCA

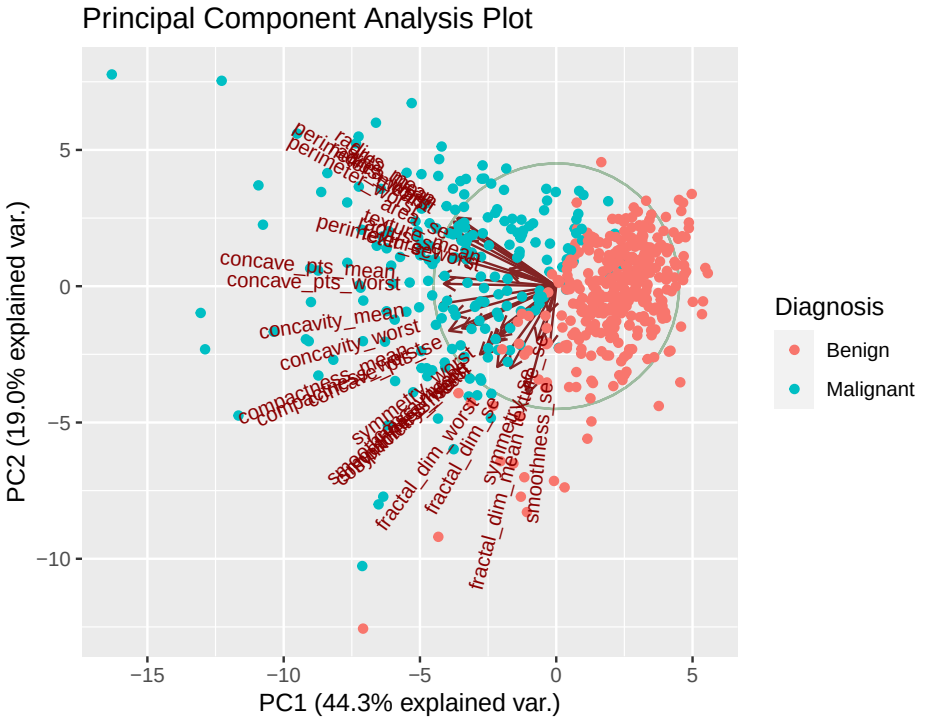

Figure 8: PCA plot of first two principle components, together accounting for 63.3 percent of variance

2.6 Creating .arff data file for machine learning experiments in Weka

The dataset does not need to be cleaned any further, there are no missing values and the id column has already been removed. There are a few clusters of columns but to manually decide which should be dropped and which used is not the best way. Since there are only 30 features they will all be passed to the machine learning algorithms and there the best features will be selected. The classification factor level order will be changed so confusion matrices (explained later) are more logical, the column will also be set as the last column, since the last column is the one Weka expects to be the classification column.

```
data$diagnosis <- factor(data$diagnosis, levels = c("Malignant", "Benign"))
RWeka::write.arff(select(data, 2:31, 1), file = "data/processed/data.arff")
```

3 Machine Learning with Weka

Now it is known what the data is like it is time to use machine learning algorithms to create a model that classifies instances as benign or malignant. This will all be done in the application Weka, which houses a multitude of machine learning algorithms with a plethora of built-in tools for standard machine learning tasks.

3.1 Relevant quality metrics

The default quality metric to measure the performance of an algorithm with is the accuracy, but accuracy is not always the most important or relevant for each project's use case. The accuracy could be less important than the speed of the model in which the classification is made. Or in some cases the data might not already be completely collected, then batch processing is not possible and stream processing should be used. With stream processing the data is continuously collected and processed fast, piece by piece, and is typically meant for when data is needed immediately.

In this case the data is already fully collected and new samples are processed manually by the acting physician, thus the model does not need to look at aspects like speed. Naturally the accuracy of resulting models is important, which describes the fraction of instances that are correctly classified of the total instances. Another potentially important thing for this project is which type of classification errors are more important. It might be preferable to earlier classify an instance as malignant over benign if not completely sure, because if a malignant breast mass gets classified as benign the patient might then make the decision to not get treatment even if they actually do need it. Almost all metrics that an algorithm can be scored by stem from the values recorded in a confusion matrix.

Confusion Matrix

A standard overview that is created and shown for models is the confusion matrix, in the confusion matrix it can be seen how instances are classified by the model versus what they actually are. For this dataset correctly classified malignant instances are true positive (TP), benign instances classified as malignant are false positive (FP), correctly classified benign instances are true negative (TN) and malignant instances classified as benign are false negative (FN). *Table 12* is an example of a confusion matrix returned by Weka.

a	b	<- classified as
TP	FP	a = Malignant
FN	TN	b = Benign

Table 12: Example of a confusion matrix. The columns are what the instances are classified as by the model and the rows show what their classification actually is.

Sensitivity and Specificity

Two important metrics for this project, and for machine learning algorithms in general, are sensitivity and specificity. These describe the probability of an instance being correctly classified as their actual classification, calculated using the values from the confusion matrix. The sensitivity is also known as the true positive rate (TPR) and is calculated as $\text{TPR} = \frac{\text{TP}}{\text{TP+FN}}$, describing the ability of the model to correctly predict positive instances as positive. Whereas the specificity is also known as the true negative rate (TNR) which describes the ability of the model to correctly predict negative instances as negative and is calculated as $\text{TNR} = \frac{\text{TN}}{\text{TN+FP}}$.

Precision

Because it is preferred for a model to get as few malignant instances classified as benign, meaning that there will be more false positive instances. Precision, or also known as the positive predictive values (PPV) metric, describes the ratio of correctly classified positive instances to the total amount of instances that were classified as positive. Since it is desired for there to be more false positives than false negatives this is not a main metric the algorithms will be scored by but it is still very relevant. Precision is calculated as $\text{PPV} = \frac{\text{TP}}{\text{TP+FP}}$.

ROC curve and Area under Curve

Receiver Operating Characteristics (ROC) was first developed by electrical engineers and radar engineers in World War II, but it is now also widely used for machine learning to measure the performance of a model with various probability threshold settings. It illustrates the diagnostic ability of a model as it's threshold varies, by plotting the true positive rate on the y axis against the false positive rate on the x axis. The ROC curve can be reduced and normalized to a single number, called the Area under Curve (AUC), which describes the probability of a classifier correctly classifying one random positive instance and one randomly selected negative instance.

The AUC of the ROC curve (AUC-ROC) can thus be used as a clear reference level for comparing algorithm performances. An AUC-ROC value of 1 would mean the model is the most optimal it could be and a value of 0 would mean the least optimal. A value of 0.5 would equate to just as good as a random guess, which is thus the worst case scenario. Even though a very low value close to 0 would not be optimal it does however classify instances consistently the wrong way around, thus if the classification is flipped, the instances are predicted correctly like if the AUC-ROC value would be close to 1. In general, algorithms with AUC-ROC values above 0.7 are considered accepted, above 0.8 are considered excellent and above 0.9 are even considered outstanding.

3.2 How the performances of machine learning algorithms and their settings will be tested

Manual exploration of algorithms and their settings can be done in the Weka Explorer but to compare a lot of different algorithms with different settings this becomes a tedious task. The Weka Experimenter is the solution to easily compare the different algorithms and their settings by enabling the user to select multiple datasets and also multiple algorithms that can all be run and tested with 10 fold cross-validation, 10 times each. An output file can be specified where each run and each cross validation gets all metrics and statistics saved to. The different algorithms' metrics can then be compared with a paired student's t-test to check for significant improvements or declines in performance.

Even though the Weka experimenter has the ability to do this, it is only able to do so per metric and the result will the need to be manually copied to this research log. For this reason the script `src/scripts/weka_analyser_custom.R` was created with the same functionality and with the exact same correction method for the t-test as found in the Weka Experimenter Analyse tab.

The resulting data from the Weka Experimenter can be loaded in using the `loadPerformanceData()` function and the comparison can then be made using `createTableWithSignificances()`, to print it in a nice format for the resulting pdf the `printNiceLatexTable()` function is used and `showAllKeys()` is called to show the used settings. Because this will be done for multiple runs this process is encapsulated in the function below for easier use.

```
printPerformanceFromFile <- function(file_path, caption) {
  # Load performance data from file
  performance_data <- loadPerformanceData(file_path)

  # Desired comparison fields/metrics
  comparison_fields <- c("Percent_correct", "True_positive_rate", "True_negative_rate",
                         "IR_precision", "Area_under_ROC")
  # Create table
  comparison_dt <- createTableWithSignificances(performance_df = performance_data,
                                                test_base_selection = 1,
                                                comparison_fields = comparison_fields)

  # Pretty print with nicer column names
  metric_names <- c("Accuracy", "Sensitivity", "Specificity",
                    "Precision", "Area under ROC")
  printNiceLatexTable(comparison_dt, metric_names, caption)

  # Show all algorithms with their settings
  showAllKeys(performance_data)
}
```

In this case the following five metrics are the most relevant and important and shall be shown and compared, namely the accuracy, sensitivity, specificity, precision and AUC-ROC. To provide some sort of a baseline performance the ZeroR and OneR algorithms are included, since they are very basic and do not use a lot of logic and computation, they can thus be used to benchmark other algorithms with to see if they are better than the baseline performance.

3.3 Some standard ML algorithms' default performance

The ZeroR and OneR algorithms are included to show a sort of baseline performance, these can be used to benchmark and compare other algorithms to. The other standard machine learning algorithms that will be used and tested are the following: Naïve Bayes, Simple Logistic, SVM (SMO), Nearest Neighbor (IBk), Decision Trees (J48) and Random Forest.

```
file_path <- "output/algorithm_performances/default.arff"
caption <- "Default performances of standard machine learning algorithms compared to ZeroR"
printPerformanceFromFile(file_path, caption)
```

Table 13: Default performances of standard machine learning algorithms compared to ZeroR

Algorithm	Accuracy	Sensitivity	Specificity	Precision	Area under ROC
(1) ZeroR	62.74	0	1	NA	0.5
(2) OneR	88.61 v	0.83 v	0.92 *	0.87	0.87 v
(3) NaiveBayes	93.31 v	0.89 v	0.96 *	0.93	0.98 v
(4) SimpleLogistic	97.40 v	0.95 v	0.99 *	0.98	1.00 v
(5) SMO	97.54 v	0.94 v	0.99	0.99	0.97 v
(6) IBk	95.64 v	0.94 v	0.97 *	0.94	0.95 v
(7) J48	93.29 v	0.91 v	0.95 *	0.91	0.93 v
(8) RandomForest	96.35 v	0.94 v	0.98 *	0.97	0.99 v

'*' = significantly worse; 'v' = significantly better

All present algorithms with their settings:
(1) ZeroR ' '
(2) OneR '*-B 6*'
(3) NaiveBayes ' '
(4) SimpleLogistic '*-I 0 -M 500 -H 50 -W 0.0*'
(5) SMO '*-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator "weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4"*'
(6) IBk '*-K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A "weka.core.EuclideanDistance -R first-last""*'
(7) J48 '*-C 0.25 -M 2*'
(8) RandomForest '*-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1*'

Based on the performances with default settings seen in *Table 13*, the SimpleLogistic and SMO algorithms look to be the best performing, with RandomForest as a close third. The algorithms could still be over-fitted and the J48 tree too big, so some settings will be changed to try to improve the resulting models. For now only OneR and J48 will be dropped and further tests will be performed on the remaining algorithms.

3.4 Testing algorithm performances using different settings with Weka experimenter

The effect of different algorithm settings with the goal of improving algorithm performance, on at least 2 machine learning algorithms. Also investigate the effect of Attribute Selection methods.
Applying appropriate statistical tests (with Weka Experimenter) taking into account the quality metrics specified earlier. Investigate some Meta learners (Stacking, Bagging, Boosting).

3.5 ROC and learning curve analysis

ROC curve visualization

Visualization of one or two final algorithms with optimal settings and explaining if the result is satisfying. Takes into account the quality metrics defined previously.

Learning curve

How much data is needed to get a reasonable performance estimate?

4 Java Wrapper for final learned model

Intens man.

References

[1] W.N. Street, W.H. Wolberg and O.L. Mangasarian. (1993), *Nuclear feature extraction for breast tumor diagnosis.*, 1993 International Symposium on Electronic Imaging: Science and Technology, volume 1905, pages 861-870, https://doi.org/10.1117/12.148698 (accessed Sep 16, 2022).

[2] O.L. Mangasarian, W.N. Street and W.H. Wolberg. (1995), *Breast cancer diagnosis and prognosis via linear programming*, Operations Research, volume 43, issue 4, pages 570-577, https://doi.org/10.1287/opre.43.4.570 (accessed Sep 17, 2022).