

# breastcancer-project

*Dibyajyoti Sarkar*

*6/8/2019*

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

## Overview

An exploratory data analysis is carried out in order to develop a machine learning algorithm that could predict whether a breast cancer cell is benign or malignant until a final model. Results will be explained. Finally, the report will end with some concluding remarks.

## Introduction

Breast cancer refers to a pathology in which a tumor develops in the breast tissue. Breast cancer is amongst the most common type of cancer in both sexes since 1975 and causes 411,000 annual deaths worldwide. Mammography is the most common mass screening tool for an early detection of breast cancers because of its sensitivity in recognising breast masses. After detection of suspicious breast masses, a biopsy test procedure would be carried out, such as Fine Needle Aspirates (FNA), that is the method this report focus on. This method has been showed to be safe, cost-effective, accurate and fast. A small drop of viscous fluid is aspirated from the breast masses to be analysed under the microscope.

This project will make a performance comparison between different machine learning algorithms in order to assess the correctness in classifying data with respect to efficiency and effectiveness of each algorithm in terms of accuracy, precision, sensitivity and specificity, in order to find the best diagnosis. The utilization of data science and machine learning approaches in medical fields proves to be prolific as such approaches may be considered of great assistance in the decision making process of medical practitioners.

## Aim of the project

The objective of this report is to train machine learning models to predict whether a breast cancer cell is Benign or Malignant. Data will be transformed and its dimension reduced to reveal patterns in the dataset and create a more robust analysis. The optimal model will be selected following the resulting accuracy, sensitivity, and f1 score, amongst other factors. The metrics will be defined later in the process. Machine learning method can be used to extract the features of cancer cell nuclei image and classify them. It would be helpful to determine whether a given sample appears to be Benign ("B") or Malignant ("M").

The machine learning models in this report will try to create a classifier that provides a high accuracy level combined with a low rate of false-negatives (high sensitivity).

## Dataset

The present report covers the Breast Cancer Wisconsin (Diagnostic) DataSet (<https://www.kaggle.com/uciml/breast-cancer-wisconsin-data/version/2>) created by Dr. William H. Wolberg, physician at the University Of Wisconsin Hospital at Madison, Wisconsin, USA. The data used for this project was collected in

1993 by the University of Wisconsin and it is composed by the biopsy result of 569 patients in Wisconsin Hospital.

- [Wisconsin Breast Cancer Diagnostic Dataset] <https://www.kaggle.com/uciml/breast-cancer-wisconsin-data/version/2>

The .csv format file containing the data is loaded from my personal github account.

The dataset's features describe characteristics of the cell nuclei on the image. The features information are specified below:

- Attribute Information:

1. ID number
2. Diagnosis (M = malignant, B = benign)

- Ten features were computed for each cell nucleus:

1. radius: mean of distances from center to points on the perimeter
2. texture: standard deviation of grey-scale values
3. perimeter
4. area: Number of pixels inside contour +  $\frac{1}{2}$  for pixels on perimeter
5. smoothness: local variation in radius lengths), , t
6. compactness:  $\text{perimeter}^2 / \text{area} - 1.0$  ; this dimensionless number is at a minimum with a circular disk and increases with the irregularity of the boundary, but this measure also increases for elongated cell nuclei, which is not indicative of malignancy
7. concavity: severity of concave portions of the contour
8. concave points: number of concave portions of the contour
9. symmetry
10. fractal dimension: "coastline approximation" - 1; a higher value corresponds a less regular contour and thus to a higher probability of malignancy

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 variables. From this diagnosis, 357 of the cases were classified as benign tumors and 212 were considered malignant tumors. All cancers and some of the benign masses were histologically confirmed

The column 33 is invalid.

```
data$diagnosis <- as.factor(data$diagnosis)
# the 33 column is invalid
data[,33] <- NULL
```

## Methods and Analysis

### Data Analysis

By observing our dataset, it was found that it contains 569 observations with 32 variables.

```
str(data)
```

```
## 'data.frame': 569 obs. of 32 variables:
## $ id : int 842302 842517 84300903 84348301 84358402 843786 844359 84458202 844...
## $ diagnosis : Factor w/ 2 levels "B","M": 2 2 2 2 2 2 2 2 2 ...
## $ radius_mean : num 18 20.6 19.7 11.4 20.3 ...
## $ texture_mean : num 10.4 17.8 21.2 20.4 14.3 ...
## $ perimeter_mean : num 122.8 132.9 130 77.6 135.1 ...
## $ area_mean : num 1001 1326 1203 386 1297 ...
## $ smoothness_mean : num 0.1184 0.0847 0.1096 0.1425 0.1003 ...
## $ compactness_mean : num 0.2776 0.0786 0.1599 0.2839 0.1328 ...
## $ concavity_mean : num 0.3001 0.0869 0.1974 0.2414 0.198 ...
## $ concave.points_mean : num 0.1471 0.0702 0.1279 0.1052 0.1043 ...
## $ symmetry_mean : num 0.242 0.181 0.207 0.26 0.181 ...
## $ fractal_dimension_mean : num 0.0787 0.0567 0.06 0.0974 0.0588 ...
## $ radius_se : num 1.095 0.543 0.746 0.496 0.757 ...
## $ texture_se : num 0.905 0.734 0.787 1.156 0.781 ...
## $ perimeter_se : num 8.59 3.4 4.58 3.44 5.44 ...
## $ area_se : num 153.4 74.1 94 27.2 94.4 ...
## $ smoothness_se : num 0.0064 0.00522 0.00615 0.00911 0.01149 ...
## $ compactness_se : num 0.049 0.0131 0.0401 0.0746 0.0246 ...
## $ concavity_se : num 0.0537 0.0186 0.0383 0.0566 0.0569 ...
## $ concave.points_se : num 0.0159 0.0134 0.0206 0.0187 0.0188 ...
## $ symmetry_se : num 0.03 0.0139 0.0225 0.0596 0.0176 ...
## $ fractal_dimension_se : num 0.00619 0.00353 0.00457 0.00921 0.00511 ...
## $ radius_worst : num 25.4 25 23.6 14.9 22.5 ...
## $ texture_worst : num 17.3 23.4 25.5 26.5 16.7 ...
## $ perimeter_worst : num 184.6 158.8 152.5 98.9 152.2 ...
## $ area_worst : num 2019 1956 1709 568 1575 ...
## $ smoothness_worst : num 0.162 0.124 0.144 0.21 0.137 ...
## $ compactness_worst : num 0.666 0.187 0.424 0.866 0.205 ...
## $ concavity_worst : num 0.712 0.242 0.45 0.687 0.4 ...
## $ concave.points_worst : num 0.265 0.186 0.243 0.258 0.163 ...
## $ symmetry_worst : num 0.46 0.275 0.361 0.664 0.236 ...
## $ fractal_dimension_worst: num 0.1189 0.089 0.0876 0.173 0.0768 ...
```

```
head(data)
```

```
##      id diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1 842302      M      17.99      10.38      122.80      1001.0
## 2 842517      M      20.57      17.77      132.90      1326.0
## 3 84300903     M      19.69      21.25      130.00      1203.0
## 4 84348301     M      11.42      20.38       77.58       386.1
## 5 84358402     M      20.29      14.34      135.10      1297.0
## 6 843786      M      12.45      15.70       82.57       477.1
## smoothness_mean compactness_mean concavity_mean concave.points_mean
## 1      0.11840      0.27760      0.3001      0.14710
## 2      0.08474      0.07864      0.0869      0.07017
## 3      0.10960      0.15990      0.1974      0.12790
## 4      0.14250      0.28390      0.2414      0.10520
## 5      0.10030      0.13280      0.1980      0.10430
## 6      0.12780      0.17000      0.1578      0.08089
## symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## 1      0.2419      0.07871      1.0950      0.9053      8.589
## 2      0.1812      0.05667      0.5435      0.7339      3.398
## 3      0.2069      0.05999      0.7456      0.7869      4.585
```

```
## 4      0.2597      0.09744      0.4956      1.1560      3.445
## 5      0.1809      0.05883      0.7572      0.7813      5.438
## 6      0.2087      0.07613      0.3345      0.8902      2.217
##      area_se smoothness_se compactness_se concavity_se concave.points_se
## 1 153.40      0.006399      0.04904      0.05373      0.01587
## 2  74.08      0.005225      0.01308      0.01860      0.01340
## 3  94.03      0.006150      0.04006      0.03832      0.02058
## 4  27.23      0.009110      0.07458      0.05661      0.01867
## 5  94.44      0.011490      0.02461      0.05688      0.01885
## 6  27.19      0.007510      0.03345      0.03672      0.01137
##      symmetry_se fractal_dimension_se radius_worst texture_worst
## 1   0.03003      0.006193      25.38      17.33
## 2   0.01389      0.003532      24.99      23.41
## 3   0.02250      0.004571      23.57      25.53
## 4   0.05963      0.009208      14.91      26.50
## 5   0.01756      0.005115      22.54      16.67
## 6   0.02165      0.005082      15.47      23.75
##      perimeter_worst area_worst smoothness_worst compactness_worst
## 1      184.60      2019.0      0.1622      0.6656
## 2      158.80      1956.0      0.1238      0.1866
## 3      152.50      1709.0      0.1444      0.4245
## 4       98.87      567.7      0.2098      0.8663
## 5      152.20      1575.0      0.1374      0.2050
## 6      103.40      741.6      0.1791      0.5249
##      concavity_worst concave.points_worst symmetry_worst
## 1       0.7119      0.2654      0.4601
## 2       0.2416      0.1860      0.2750
## 3       0.4504      0.2430      0.3613
## 4       0.6869      0.2575      0.6638
## 5       0.4000      0.1625      0.2364
## 6       0.5355      0.1741      0.3985
##      fractal_dimension_worst
## 1      0.11890
## 2      0.08902
## 3      0.08758
## 4      0.17300
## 5      0.07678
## 6      0.12440
```

```
summary(data)
```

```
##      id      diagnosis radius_mean texture_mean
## Min.   :    8670 B:357      Min.   : 6.981      Min.   : 9.71
## 1st Qu.:  869218 M:212      1st Qu.:11.700      1st Qu.:16.17
## Median :  906024      Median :13.370      Median :18.84
## Mean   : 30371831      Mean   :14.127      Mean   :19.29
## 3rd Qu.:  8813129      3rd Qu.:15.780      3rd Qu.:21.80
## Max.   :911320502      Max.   :28.110      Max.   :39.28
## perimeter_mean area_mean smoothness_mean compactness_mean
## Min.   : 43.79      Min.   : 143.5      Min.   :0.05263      Min.   :0.01938
## 1st Qu.: 75.17      1st Qu.: 420.3      1st Qu.:0.08637      1st Qu.:0.06492
## Median : 86.24      Median : 551.1      Median :0.09587      Median :0.09263
## Mean   : 91.97      Mean   : 654.9      Mean   :0.09636      Mean   :0.10434
## 3rd Qu.:104.10      3rd Qu.: 782.7      3rd Qu.:0.10530      3rd Qu.:0.13040
```

```

## Max. :188.50 Max. :2501.0 Max. :0.16340 Max. :0.34540
## concavity_mean concave.points_mean symmetry_mean
## Min. :0.00000 Min. :0.00000 Min. :0.1060
## 1st Qu.:0.02956 1st Qu.:0.02031 1st Qu.:0.1619
## Median :0.06154 Median :0.03350 Median :0.1792
## Mean :0.08880 Mean :0.04892 Mean :0.1812
## 3rd Qu.:0.13070 3rd Qu.:0.07400 3rd Qu.:0.1957
## Max. :0.42680 Max. :0.20120 Max. :0.3040
## fractal_dimension_mean radius_se texture_se perimeter_se
## Min. :0.04996 Min. :0.1115 Min. :0.3602 Min. : 0.757
## 1st Qu.:0.05770 1st Qu.:0.2324 1st Qu.:0.8339 1st Qu.: 1.606
## Median :0.06154 Median :0.3242 Median :1.1080 Median : 2.287
## Mean :0.06280 Mean :0.4052 Mean :1.2169 Mean : 2.866
## 3rd Qu.:0.06612 3rd Qu.:0.4789 3rd Qu.:1.4740 3rd Qu.: 3.357
## Max. :0.09744 Max. :2.8730 Max. :4.8850 Max. :21.980
## area_se smoothness_se compactness_se concavity_se
## Min. : 6.802 Min. :0.001713 Min. :0.002252 Min. :0.00000
## 1st Qu.: 17.850 1st Qu.:0.005169 1st Qu.:0.013080 1st Qu.:0.01509
## Median : 24.530 Median :0.006380 Median :0.020450 Median :0.02589
## Mean : 40.337 Mean :0.007041 Mean :0.025478 Mean :0.03189
## 3rd Qu.: 45.190 3rd Qu.:0.008146 3rd Qu.:0.032450 3rd Qu.:0.04205
## Max. :542.200 Max. :0.031130 Max. :0.135400 Max. :0.39600
## concave.points_se symmetry_se fractal_dimension_se
## Min. :0.000000 Min. :0.007882 Min. :0.0008948
## 1st Qu.:0.007638 1st Qu.:0.015160 1st Qu.:0.0022480
## Median :0.010930 Median :0.018730 Median :0.0031870
## Mean :0.011796 Mean :0.020542 Mean :0.0037949
## 3rd Qu.:0.014710 3rd Qu.:0.023480 3rd Qu.:0.0045580
## Max. :0.052790 Max. :0.078950 Max. :0.0298400
## radius_worst texture_worst perimeter_worst area_worst
## Min. : 7.93 Min. :12.02 Min. : 50.41 Min. : 185.2
## 1st Qu.:13.01 1st Qu.:21.08 1st Qu.: 84.11 1st Qu.: 515.3
## Median :14.97 Median :25.41 Median : 97.66 Median : 686.5
## Mean :16.27 Mean :25.68 Mean :107.26 Mean : 880.6
## 3rd Qu.:18.79 3rd Qu.:29.72 3rd Qu.:125.40 3rd Qu.:1084.0
## Max. :36.04 Max. :49.54 Max. :251.20 Max. :4254.0
## smoothness_worst compactness_worst concavity_worst concave.points_worst
## Min. :0.07117 Min. :0.02729 Min. :0.0000 Min. :0.00000
## 1st Qu.:0.11660 1st Qu.:0.14720 1st Qu.:0.1145 1st Qu.:0.06493
## Median :0.13130 Median :0.21190 Median :0.2267 Median :0.09993
## Mean :0.13237 Mean :0.25427 Mean :0.2722 Mean :0.11461
## 3rd Qu.:0.14600 3rd Qu.:0.33910 3rd Qu.:0.3829 3rd Qu.:0.16140
## Max. :0.22260 Max. :1.05800 Max. :1.2520 Max. :0.29100
## symmetry_worst fractal_dimension_worst
## Min. :0.1565 Min. :0.05504
## 1st Qu.:0.2504 1st Qu.:0.07146
## Median :0.2822 Median :0.08004
## Mean :0.2901 Mean :0.08395
## 3rd Qu.:0.3179 3rd Qu.:0.09208
## Max. :0.6638 Max. :0.20750

```

Check if the dataset has any missing value:

```
## $id
```

```

## [1] 0
##
## $diagnosis
## [1] 0
##
## $radius_mean
## [1] 0
##
## $texture_mean
## [1] 0
##
## $perimeter_mean
## [1] 0
##
## $area_mean
## [1] 0
##
## $smoothness_mean
## [1] 0
##
## $compactness_mean
## [1] 0
##
## $concavity_mean
## [1] 0
##
## $concave.points_mean
## [1] 0
##
## $symmetry_mean
## [1] 0
##
## $fractal_dimension_mean
## [1] 0
##
## $radius_se
## [1] 0
##
## $texture_se
## [1] 0
##
## $perimeter_se
## [1] 0
##
## $area_se
## [1] 0
##
## $smoothness_se
## [1] 0
##
## $compactness_se
## [1] 0
##
## $concavity_se

```

```
## [1] 0
##
## $concave.points_se
## [1] 0
##
## $symmetry_se
## [1] 0
##
## $fractal_dimension_se
## [1] 0
##
## $radius_worst
## [1] 0
##
## $texture_worst
## [1] 0
##
## $perimeter_worst
## [1] 0
##
## $area_worst
## [1] 0
##
## $smoothness_worst
## [1] 0
##
## $compactness_worst
## [1] 0
##
## $concavity_worst
## [1] 0
##
## $concave.points_worst
## [1] 0
##
## $symmetry_worst
## [1] 0
##
## $fractal_dimension_worst
## [1] 0
```

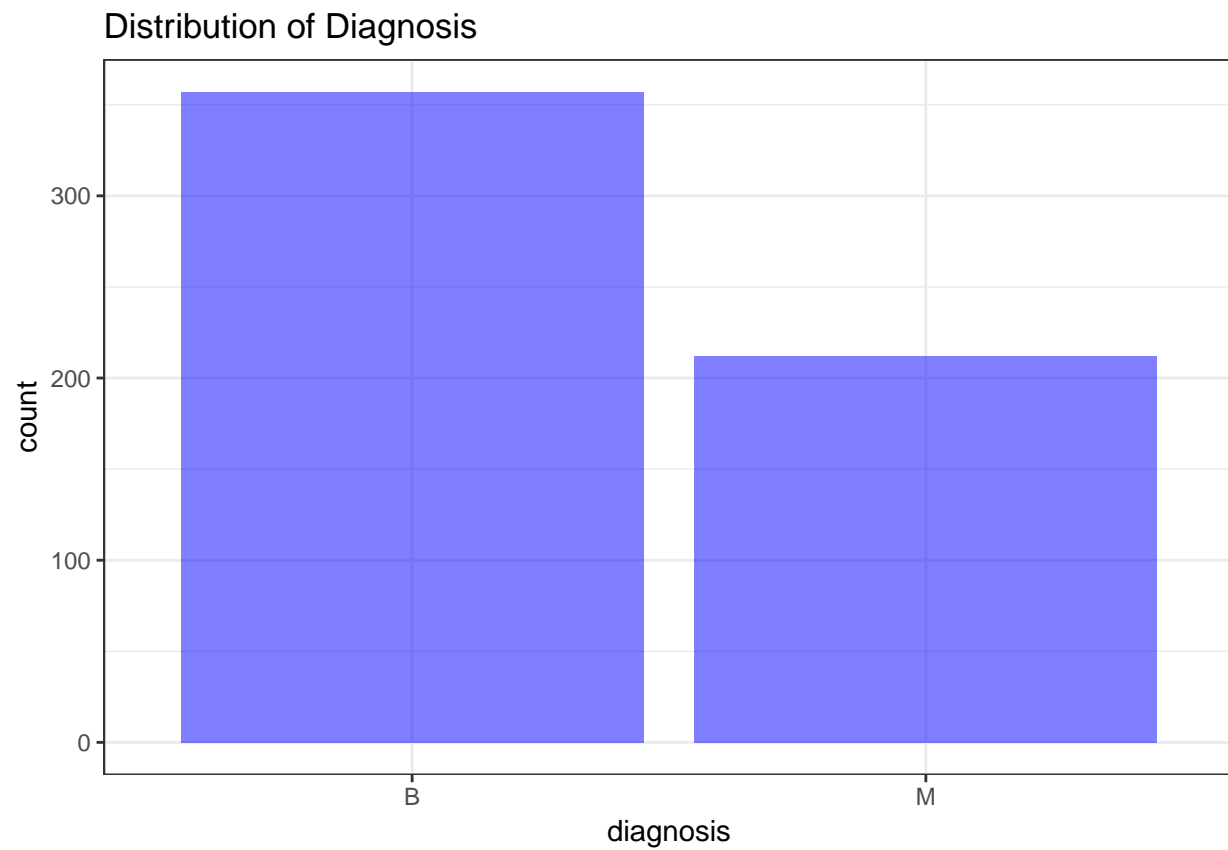
There aren't NA values. By analysing the dataset, it is found that it is a bit unbalanced in its proportions:

```
prop.table(table(data$diagnosis))
```

```
##
##           B           M
## 0.6274165 0.3725835
```

Also the plot of proportions confirms that the target variable is slightly unbalanced.

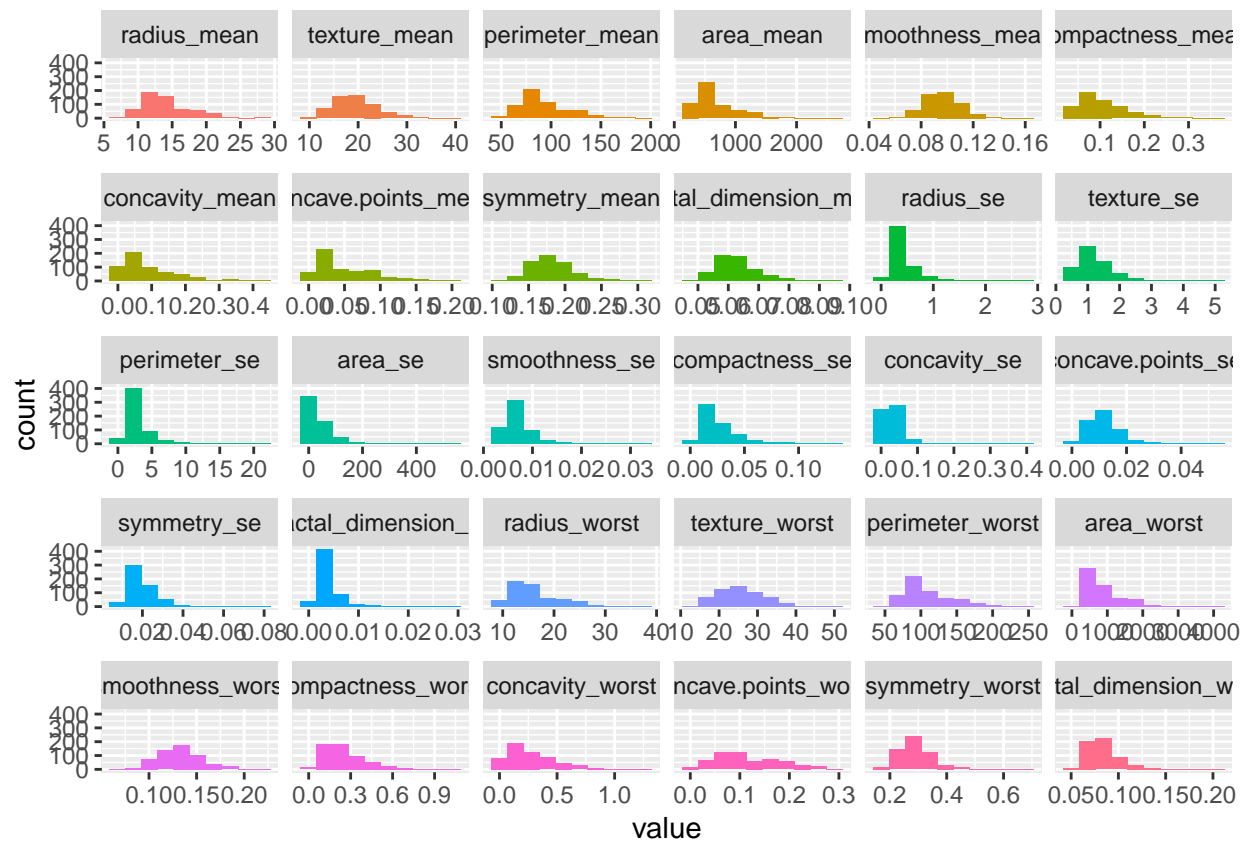
```
options(repr.plot.width=4, repr.plot.height=4)
ggplot(data, aes(x=diagnosis))+geom_bar(fill="blue",alpha=0.5)+theme_bw()+labs(title="Distribution of D
```



The most variables of the dataset are normally distributed as show with the below plot:

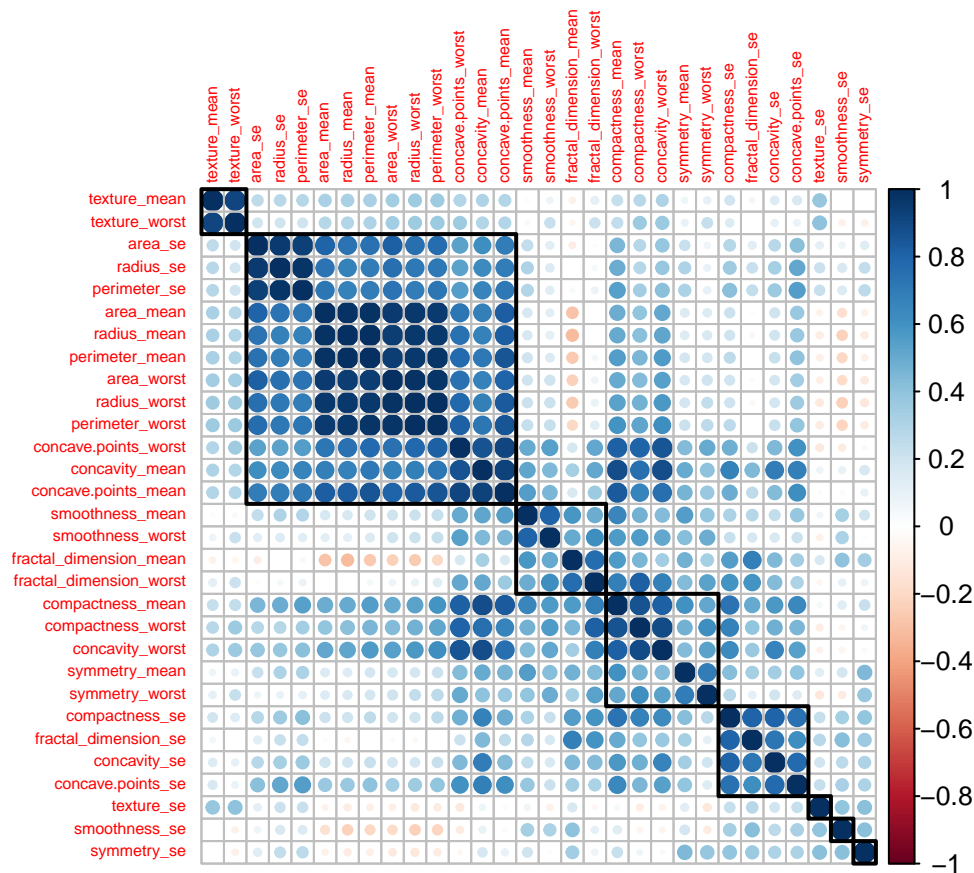
```
plot_num(data %>% select(-id), bins=10)
```





Check if there is any correlation between variables as machine learning algorithms assume that the predictor variables are independent from each other.

```
correlationMatrix <- cor(data[,3:ncol(data)])
corrplot(correlationMatrix, order = "hclust", tl.cex = 0.5, addrect = 8)
```



As shown by this plot, many variables are highly correlated with each others. Many methods perform better if highly correlated attributes are removed. The Caret R package provides the `findCorrelation` which will analyze a correlation matrix of your data's attributes report on attributes that can be removed. Because of much correlation some machine learning models could fail.

```
# find attributes that are highly correlated (ideally >0.90)
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.9)
# print indexes of highly correlated attributes
print(highlyCorrelated)
```

```
## [1] 7 8 23 21 3 24 1 13 14 2
```

Selecting the right features in data can mean the difference between mediocre performance with long training times and great performance with short training times.

```
# Remove correlated variables
data2 <- data %>%select(-highlyCorrelated)
# number of columns after removing correlated variables
ncol(data2)
```

```
## [1] 22
```

The new dataset has less 10 variables.

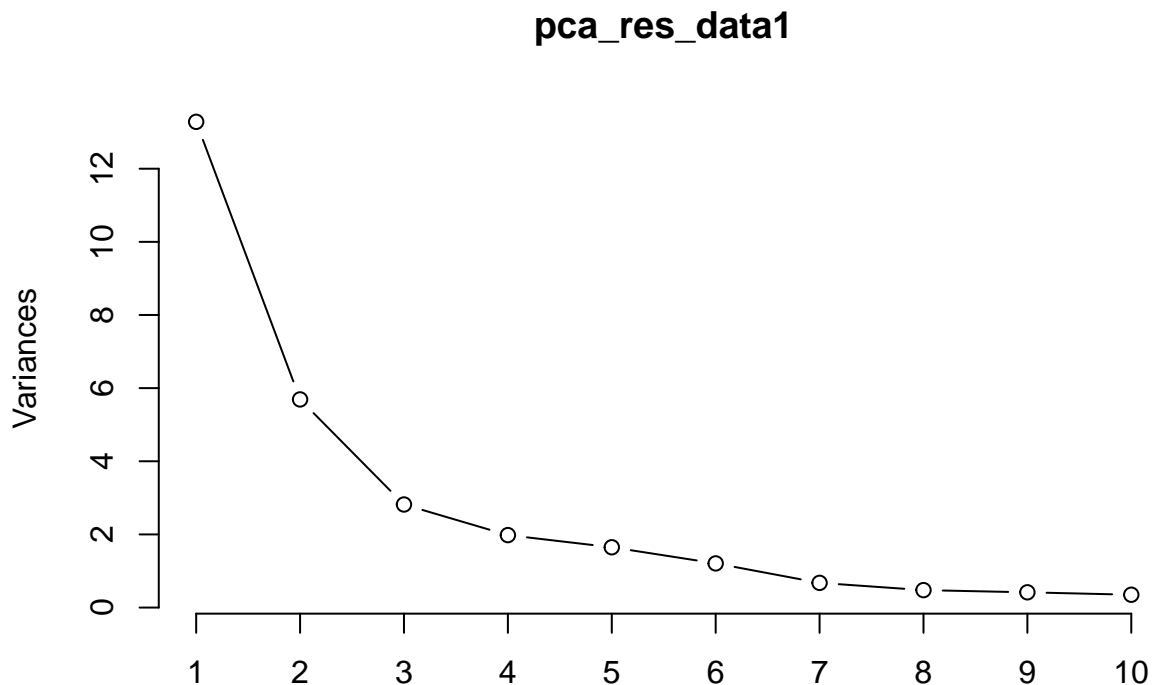
## Modelling Approach

### Modelling

Principal Component Analysis (PCA).

To avoid redundancy and relevancy, the function 'prncomp' was used to calculate the Principal Component Analysis (PCA) and select the rights components to avoid correlated variables that can be detrimental to our clustering analysis. One of the common problems in analysis of complex data comes from a large number of variables, which requires a large amount of memory and computation power. This is where PCA comes in. It is a technique to reduce the dimension of the feature space by feature extraction. The main idea of PCA is to reduce the dimensionality of a data set consisting of many variables correlated with each other, either heavily or lightly, while retaining the variation present in the dataset, up to the maximum extent. The same is done by transforming the variables to a new set of variables, which are known as the principal components (or simply, the PCs) and are orthogonal, ordered such that the retention of variation present in the original variables decrease as one moves down in the order.

```
pca_res_data1 <- prcomp(data[,3:ncol(data)], center = TRUE, scale = TRUE)
plot(pca_res_data1, type="l")
```



```
summary(pca_res_data1)
```

```
## Importance of components:
##              PC1    PC2    PC3    PC4    PC5    PC6
## Standard deviation  3.6444 2.3857 1.67867 1.40735 1.28403 1.09880
## Proportion of Variance 0.4427 0.1897 0.09393 0.06602 0.05496 0.04025
## Cumulative Proportion 0.4427 0.6324 0.72636 0.79239 0.84734 0.88759
##              PC7    PC8    PC9   PC10   PC11   PC12
## Standard deviation  0.82172 0.69037 0.6457 0.59219 0.5421 0.51104
## Proportion of Variance 0.02251 0.01589 0.0139 0.01169 0.0098 0.00871
## Cumulative Proportion 0.91010 0.92598 0.9399 0.95157 0.9614 0.97007
```

	PC13	PC14	PC15	PC16	PC17	PC18
## Standard deviation	0.49128	0.39624	0.30681	0.28260	0.24372	0.22939
## Proportion of Variance	0.00805	0.00523	0.00314	0.00266	0.00198	0.00175
## Cumulative Proportion	0.97812	0.98335	0.98649	0.98915	0.99113	0.99288

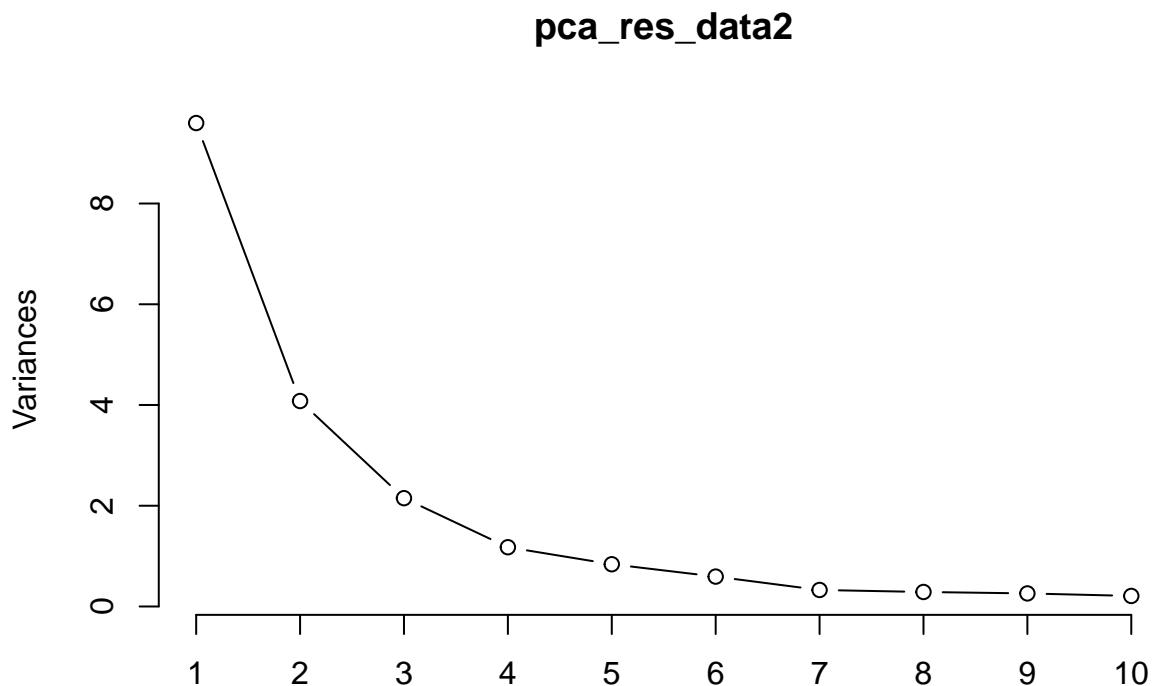
	PC19	PC20	PC21	PC22	PC23	PC24
## Standard deviation	0.22244	0.17652	0.1731	0.16565	0.15602	0.1344
## Proportion of Variance	0.00165	0.00104	0.0010	0.00091	0.00081	0.0006
## Cumulative Proportion	0.99453	0.99557	0.9966	0.99749	0.99830	0.9989

	PC25	PC26	PC27	PC28	PC29	PC30
## Standard deviation	0.12442	0.09043	0.08307	0.03987	0.02736	0.01153
## Proportion of Variance	0.00052	0.00027	0.00023	0.00005	0.00002	0.00000
## Cumulative Proportion	0.99942	0.99969	0.99992	0.99997	1.00000	1.00000

It can be observed from the above table that the two first components explain the 0.6324 of the variance. One needs 10 principal components to explain more than 0.95 of the variance and 17 to explain more than 0.99.

```
pca_res_data2 <- prcomp(data2[,3:ncol(data2)], center = TRUE, scale = TRUE)
plot(pca_res_data2, type="l")
```



```
summary(pca_res_data2)
```

## Importance of components:	PC1	PC2	PC3	PC4	PC5	PC6	PC7
## Standard deviation	3.0980	2.0196	1.4663	1.0845	0.91561	0.77019	0.57227
## Proportion of Variance	0.4799	0.2039	0.1075	0.0588	0.04192	0.02966	0.01637
## Cumulative Proportion	0.4799	0.6838	0.7913	0.8501	0.89205	0.92171	0.93808

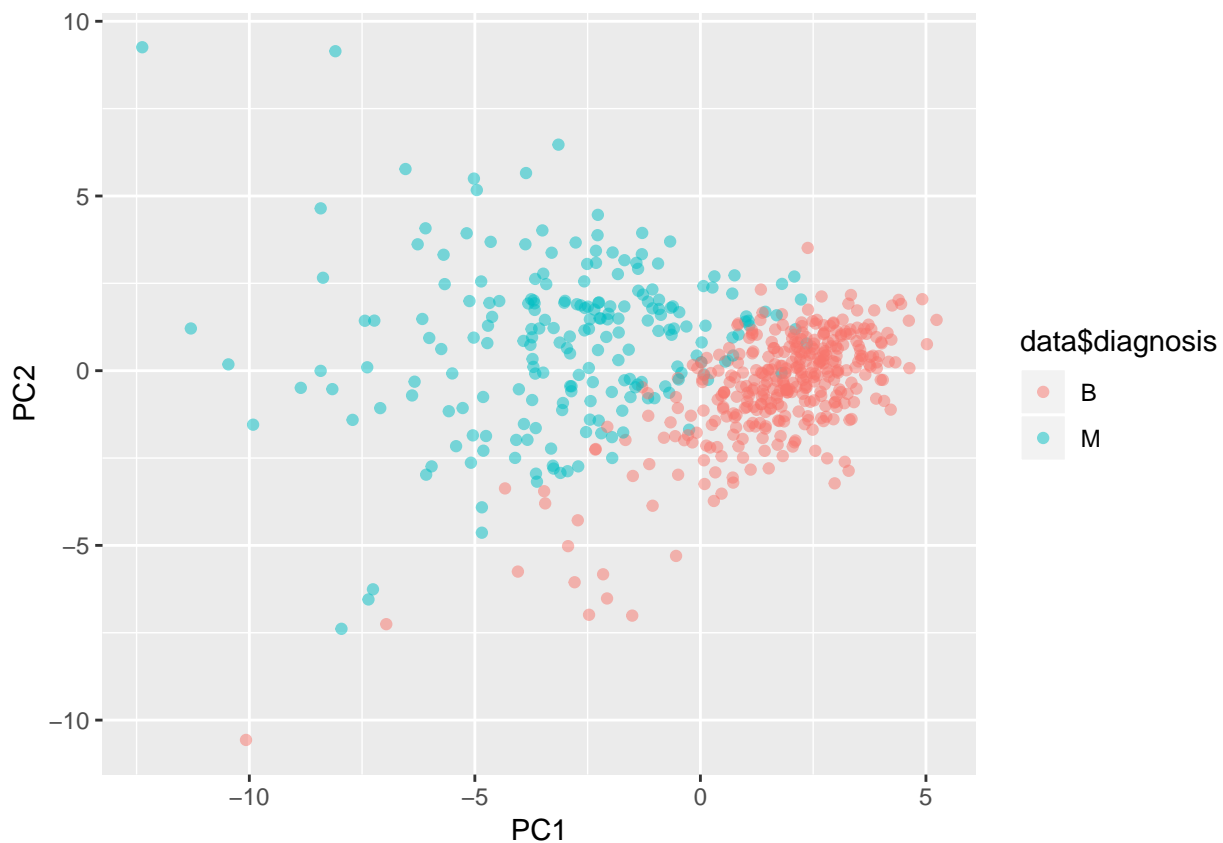
  

##	PC8	PC9	PC10	PC11	PC12	PC13
## Standard deviation	0.53641	0.50898	0.45726	0.36641	0.31778	0.28802
## Proportion of Variance	0.01439	0.01295	0.01045	0.00671	0.00505	0.00415
## Cumulative Proportion	0.95247	0.96542	0.97588	0.98259	0.98764	0.99179

```
##          PC14  PC15  PC16  PC17  PC18  PC19
## Standard deviation 0.21369 0.1846 0.15579 0.15393 0.14782 0.09636
## Proportion of Variance 0.00228 0.0017 0.00121 0.00118 0.00109 0.00046
## Cumulative Proportion 0.99407 0.9958 0.99699 0.99817 0.99926 0.99973
##          PC20
## Standard deviation 0.07375
## Proportion of Variance 0.00027
## Cumulative Proportion 1.00000
```

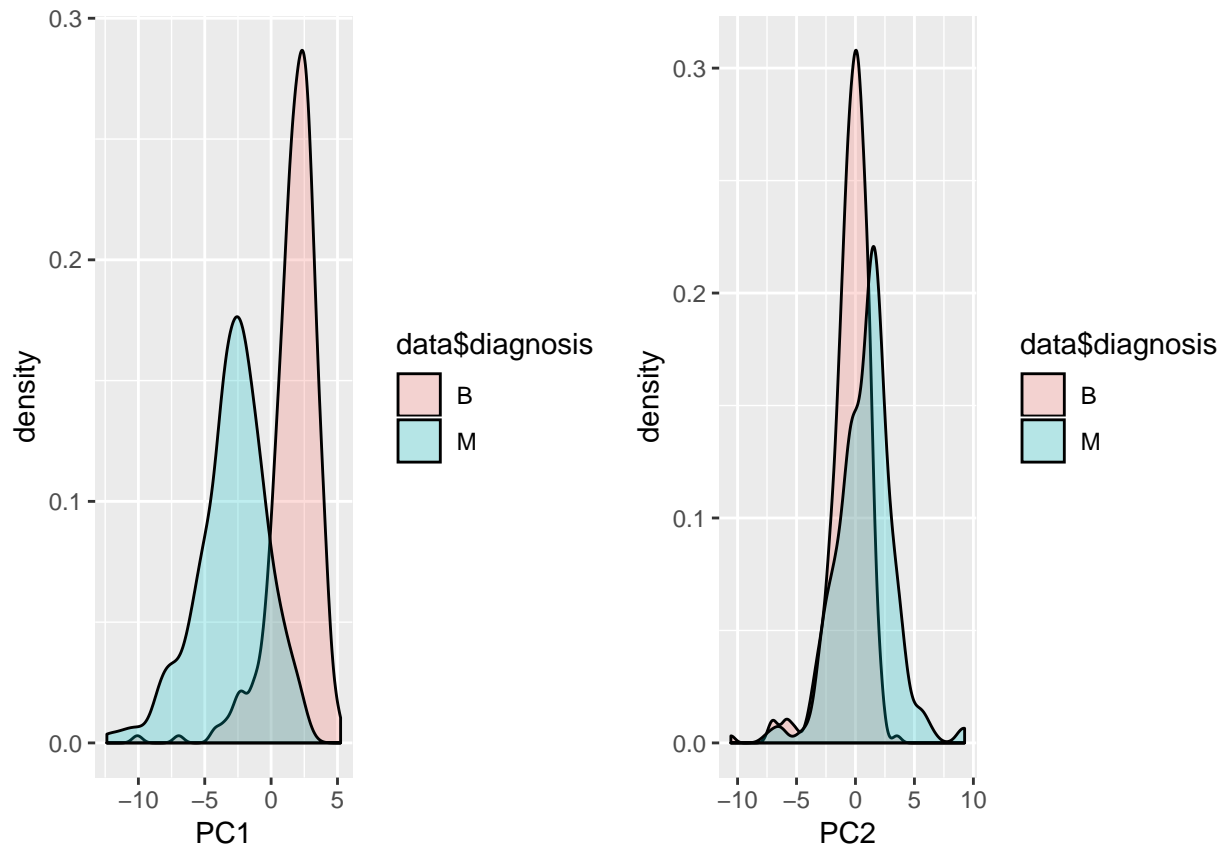
The above table shows that 95% of the variance is explained with 8 PC's in the transformed dataset data2.

```
pca_df <- as.data.frame(pca_res_data2$x)
ggplot(pca_df, aes(x=PC1, y=PC2, col=data$diagnosis)) + geom_point(alpha=0.5)
```



The data of the first 2 components can be easily separated into two classes. This is caused by the fact that the variance explained by these components is not large. The data can be easily separated.

```
g_pc1 <- ggplot(pca_df, aes(x=PC1, fill=data$diagnosis)) + geom_density(alpha=0.25)
g_pc2 <- ggplot(pca_df, aes(x=PC2, fill=data$diagnosis)) + geom_density(alpha=0.25)
grid.arrange(g_pc1, g_pc2, ncol=2)
```



### Linear Discriminant Analysis (LDA)

Another approach is to use the Linear Discriminant Analysis (LDA) instead of PCA. LDA takes in consideration the different classes and could get better results. The particularity of LDA is that it models the distribution of predictors separately in each of the response classes, and then it uses Bayes' theorem to estimate the probability. It is important to know that LDA assumes a normal distribution for each class, a class-specific mean, and a common variance.

```
lda_res_data <- MASS::lda(diagnosis~., data = data, center = TRUE, scale = TRUE)
lda_res_data
```

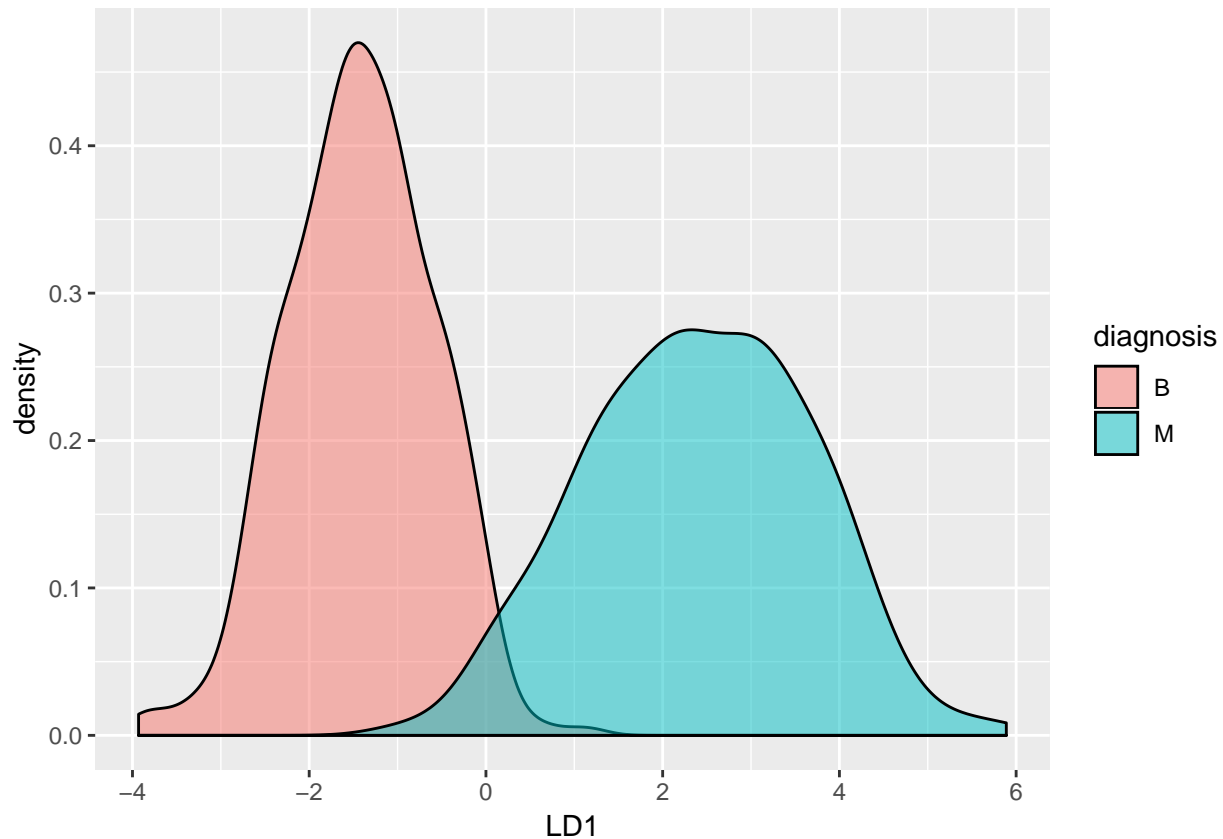
```
## Call:
## lda(diagnosis ~ ., data = data, center = TRUE, scale = TRUE)
##
## Prior probabilities of groups:
##      B      M
## 0.6274165 0.3725835
##
## Group means:
##      id radius_mean texture_mean perimeter_mean area_mean
## B 26543825    12.14652    17.91476     78.07541  462.7902
## M 36818050    17.46283    21.60491    115.36538  978.3764
##      smoothness_mean compactness_mean concavity_mean concave.points_mean
## B    0.09247765    0.08008462    0.04605762    0.02571741
## M    0.10289849    0.14518778    0.16077472    0.08799000
##      symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## B    0.174186    0.06286739 0.2840824    1.220380    2.000321
```

```
## M      0.192909      0.06268009 0.6090825  1.210915  4.323929
##      area_se smoothness_se compactness_se concavity_se concave.points_se
## B 21.13515  0.007195902    0.02143825  0.02599674    0.009857653
## M 72.67241  0.006780094    0.03228117  0.04182401    0.015060472
##      symmetry_se fractal_dimension_se radius_worst texture_worst
## B 0.02058381    0.003636051    13.37980    23.51507
## M 0.02047240    0.004062406    21.13481    29.31821
##      perimeter_worst area_worst smoothness_worst compactness_worst
## B      87.00594  558.8994    0.1249595    0.1826725
## M     141.37033 1422.2863    0.1448452    0.3748241
##      concavity_worst concave.points_worst symmetry_worst
## B      0.1662377    0.07444434    0.2702459
## M      0.4506056    0.18223731    0.3234679
##      fractal_dimension_worst
## B      0.07944207
## M      0.09152995
##
## Coefficients of linear discriminants:
##                               LD1
## id                        -2.512117e-10
## radius_mean              -1.080876e+00
## texture_mean              2.338408e-02
## perimeter_mean           1.172707e-01
## area_mean                 1.595690e-03
## smoothness_mean          5.251575e-01
## compactness_mean         -2.094197e+01
## concavity_mean            6.955923e+00
## concave.points_mean       1.047567e+01
## symmetry_mean             4.938898e-01
## fractal_dimension_mean    -5.937663e-02
## radius_se                 2.101503e+00
## texture_se                -3.979869e-02
## perimeter_se              -1.121814e-01
## area_se                   -4.083504e-03
## smoothness_se             7.987663e+01
## compactness_se            1.387026e-01
## concavity_se              -1.768261e+01
## concave.points_se         5.350520e+01
## symmetry_se               8.143611e+00
## fractal_dimension_se      -3.431356e+01
## radius_worst              9.677207e-01
## texture_worst             3.540591e-02
## perimeter_worst           -1.204507e-02
## area_worst                -5.012127e-03
## smoothness_worst          2.612258e+00
## compactness_worst         3.636892e-01
## concavity_worst           1.880699e+00
## concave.points_worst      2.218189e+00
## symmetry_worst            2.783102e+00
## fractal_dimension_worst   2.117830e+01
```

```
#Data frame of the LDA for visualization purposes
```

```
lda_df_predict <- predict(lda_res_data, data)$x %>% as.data.frame() %>% cbind(diagnosis=data$diagnosis)
```

```
ggplot(lda_df_predict, aes(x=LD1, fill=diagnosis)) + geom_density(alpha=0.5)
```



## Model Creation

It is necessary to have a training and a testing set when building some models. One can split the modified dataset into Train (80%) and Test (20%), in order to predict is whether a cancer cell is Benign or Malignant, by building machine learning classification models.

```
set.seed(1815)
data3 <- cbind (diagnosis=data$diagnosis, data2)
data_sampling_index <- createDataPartition(data$diagnosis, times=1, p=0.8, list = FALSE)
train_data <- data3[data_sampling_index, ]
test_data <- data3[-data_sampling_index, ]

fitControl <- trainControl(method="cv",      #Control the computational nuances of the train function
                           number = 15,     #Either the number of folds or number of resampling iterations
                           classProbs = TRUE,
                           summaryFunction = twoClassSummary)
```

## Naive Bayes Model

The Naive Bayesian classifier is based on Bayes' theorem with the independence assumptions between predictors. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which



makes it particularly useful for very large datasets. Bayes theorem provides a way of calculating the posterior probability,  $P(c|x)$ , from  $P(c)$ ,  $P(x)$ , and  $P(x|c)$ . Naive Bayes classifier assume that the effect of the value of a predictor ( $x$ ) on a given class ( $c$ ) is independent of the values of other predictors. This assumption is called class conditional independence.

```
model_naiveb <- train(diagnosis~.,
                      train_data,
                      method="nb",
                      metric="ROC",
                      preProcess=c('center', 'scale'), #in order to normalize the data
                      trace=FALSE,
                      trControl=fitControl)

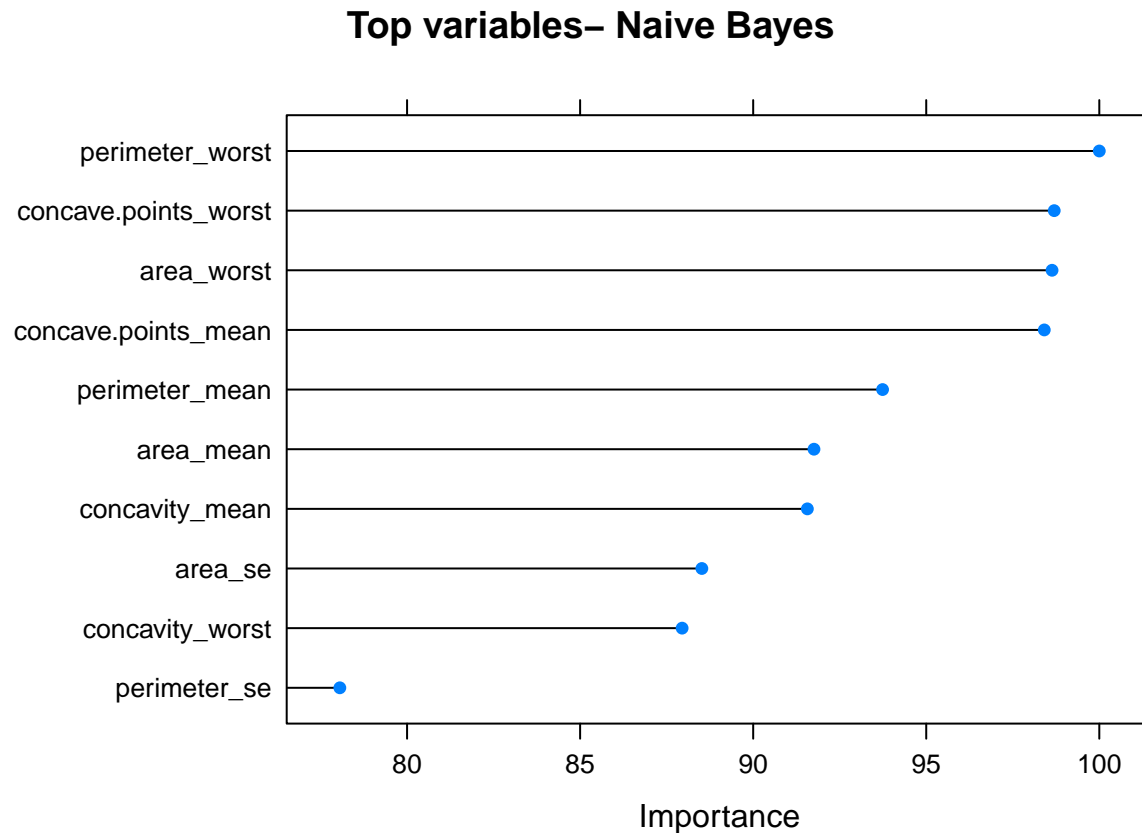
prediction_naiveb <- predict(model_naiveb, test_data)
confusionmatrix_naiveb <- confusionMatrix(prediction_naiveb, test_data$diagnosis, positive = "M")
confusionmatrix_naiveb
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##           B 69  5
##           M  2 37
##
##           Accuracy : 0.9381
##           95% CI : (0.8765, 0.9747)
##           No Information Rate : 0.6283
##           P-Value [Acc > NIR] : 1.718e-14
##
##           Kappa : 0.8654
##           Mcnemar's Test P-Value : 0.4497
##
##           Sensitivity : 0.8810
##           Specificity : 0.9718
##           Pos Pred Value : 0.9487
##           Neg Pred Value : 0.9324
##           Prevalence : 0.3717
##           Detection Rate : 0.3274
##           Detection Prevalence : 0.3451
##           Balanced Accuracy : 0.9264
##
##           'Positive' Class : M
##
```

One can note the accuracy with such model. These metrics will be described later, where: Sensitivity (recall) represent the true positive rate: the proportions of actual positives correctly identified. Specificity is the true negative rate: the proportion of actual negatives correctly identified. Accuracy is the general score of the classifier model performance as it is the ratio of how many samples are correctly classified to all samples. F1 score: the harmonic mean of precision and sensitivity. Accuracy and F1 score would be used to compare the result with the benchmark model. Precision: the number of correct positive results divided by the number of all positive results returned by the classifier.

The most important variables that permit the best prediction and contribute the most to the model are the following:

```
plot(varImp(model_naiveb), top=10, main="Top variables- Naive Bayes")
```



## Logistic Regression Model

Logistic Regression is widely used for binary classification like (0,1). The binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features).

```
model_logreg<- train(diagnosis ~., data = train_data, method = "glm",
                     metric = "ROC",

                     preProcess = c("scale", "center"), # in order to normalize the data
                     trControl= fitControl)
prediction_logreg<- predict(model_logreg, test_data)

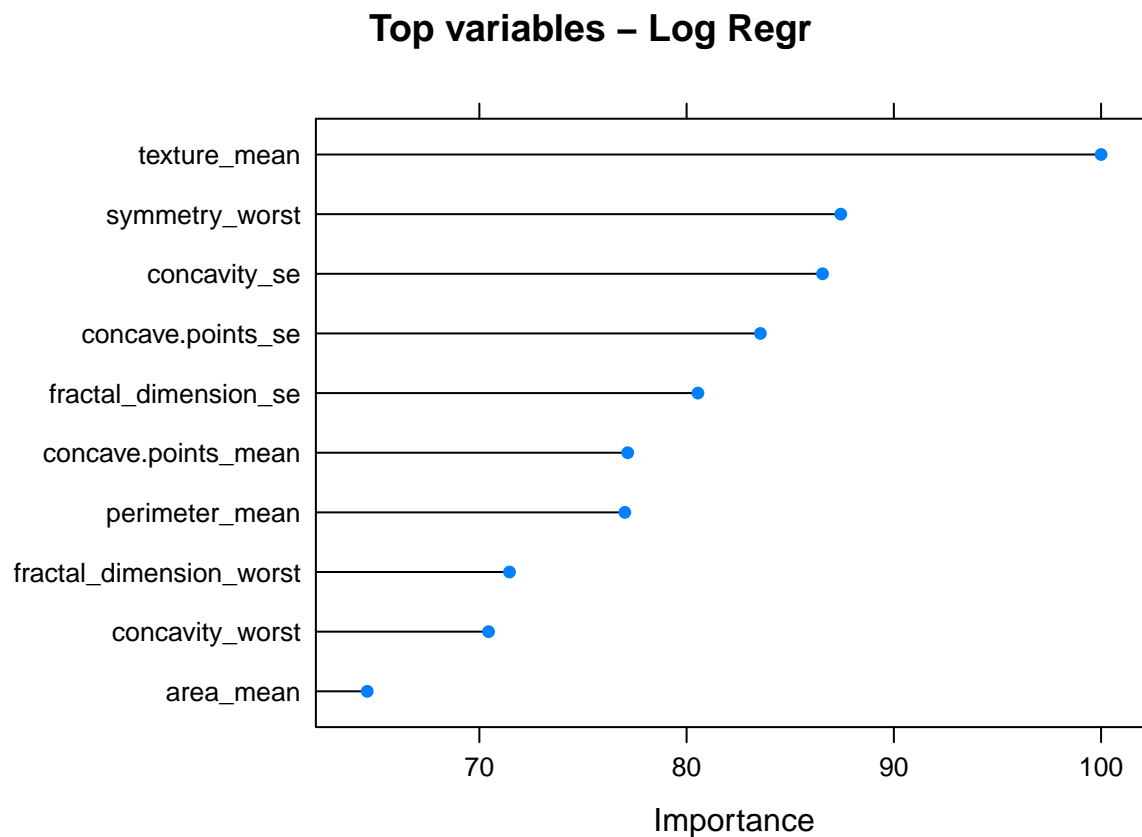
# Check results
confusionmatrix_logreg <- confusionMatrix(prediction_logreg, test_data$diagnosis, positive = "M")
confusionmatrix_logreg
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##           B 71  2
##           M  0 40
```

```
##
##           Accuracy : 0.9823
##           95% CI   : (0.9375, 0.9978)
##    No Information Rate : 0.6283
##    P-Value [Acc > NIR] : <2e-16
##
##           Kappa : 0.9617
##  Mcnemar's Test P-Value : 0.4795
##
##           Sensitivity : 0.9524
##           Specificity : 1.0000
##    Pos Pred Value : 1.0000
##    Neg Pred Value : 0.9726
##           Prevalence : 0.3717
##    Detection Rate : 0.3540
##    Detection Prevalence : 0.3540
##    Balanced Accuracy : 0.9762
##
##    'Positive' Class : M
##
```

The most important variables that permit the best prediction and contribute the most to the model are the following:

```
plot(varImp(model_logreg), top=10, main="Top variables - Log Regr")
```



## Random Forest Model

Random forests are a very popular machine learning approach that addresses the shortcomings of decision trees using a clever idea. The goal is to improve prediction performance and reduce instability by averaging multiple decision trees (a forest of trees constructed with randomness). Random forest is another ensemble method based on decision trees. It split data into sub-samples, trains decision tree classifiers on each sub-sample and averages prediction of each classifier. Splitting dataset causes higher bias but it is compensated by large decrease in variance. Random Forest is a supervised learning algorithm and it is flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because of it's simplicity and the fact that it can be used for both classification and regression tasks. Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

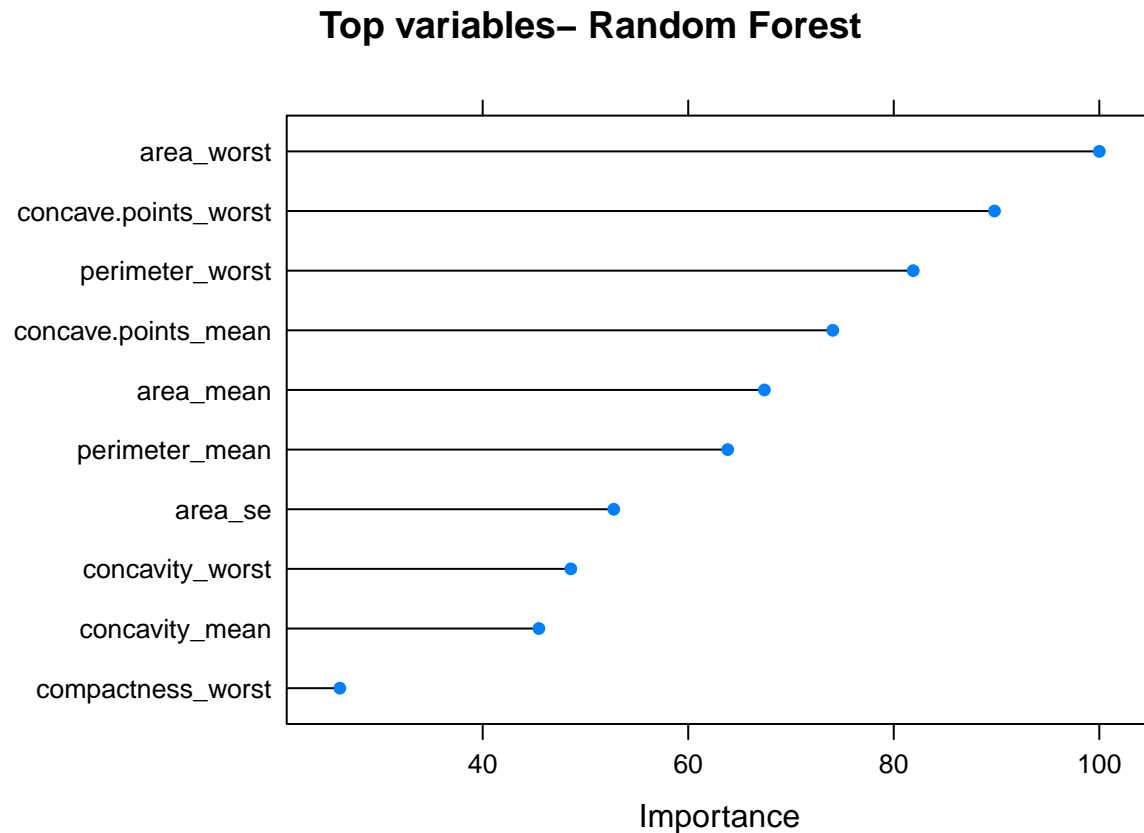
```
model_randomforest <- train(diagnosis~.,
                             train_data,
                             method="rf", #also recommended ranger, because it is a lot faster than ori
                             metric="ROC",
                             #tuneLength=10,
                             #tuneGrid = expand.grid(mtry = c(2, 3, 6)),
                             preProcess = c('center', 'scale'),
                             trControl=fitControl)

prediction_randomforest <- predict(model_randomforest, test_data)

#Check results
confusionmatrix_randomforest <- confusionMatrix(prediction_randomforest, test_data$diagnosis, positive = "M")
confusionmatrix_randomforest
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction  B  M
##          B 71  3
##          M  0 39
##
##              Accuracy : 0.9735
##              95% CI : (0.9244, 0.9945)
##      No Information Rate : 0.6283
##      P-Value [Acc > NIR] : <2e-16
##
##              Kappa : 0.9423
##  McNemar's Test P-Value : 0.2482
##
##              Sensitivity : 0.9286
##              Specificity : 1.0000
##              Pos Pred Value : 1.0000
##              Neg Pred Value : 0.9595
##              Prevalence : 0.3717
##              Detection Rate : 0.3451
##      Detection Prevalence : 0.3451
##              Balanced Accuracy : 0.9643
##
##              'Positive' Class : M
##
```

```
plot(varImp(model_randomforest), top=10, main="Top variables- Random Forest")
```



## K Nearest Neighbor (KNN) Model

KNN (K-Nearest Neighbors) is one of many (supervised learning) algorithms used in data mining and machine learning, it's a classifier algorithm where the learning is based "how similar" is a data from other. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions).

```
model_knn <- train(diagnosis~.,
  train_data,
  method="knn",
  metric="ROC",
  preProcess = c('center', 'scale'),
  tuneLength=10, #The tuneLength parameter tells the algorithm to try different default
  #In this case 10 default values are used
  trControl=fitControl)

prediction_knn <- predict(model_knn, test_data)
confusionmatrix_knn <- confusionMatrix(prediction_knn, test_data$diagnosis, positive = "M")
confusionmatrix_knn

## Confusion Matrix and Statistics
##
##           Reference
```

```

## Prediction  B  M
##           B 70  5
##           M  1 37
##
##           Accuracy : 0.9469
##           95% CI : (0.888, 0.9803)
##           No Information Rate : 0.6283
##           P-Value [Acc > NIR] : 1.866e-15
##
##           Kappa : 0.8841
##           McNemar's Test P-Value : 0.2207
##
##           Sensitivity : 0.8810
##           Specificity : 0.9859
##           Pos Pred Value : 0.9737
##           Neg Pred Value : 0.9333
##           Prevalence : 0.3717
##           Detection Rate : 0.3274
##           Detection Prevalence : 0.3363
##           Balanced Accuracy : 0.9334
##
##           'Positive' Class : M
##

```

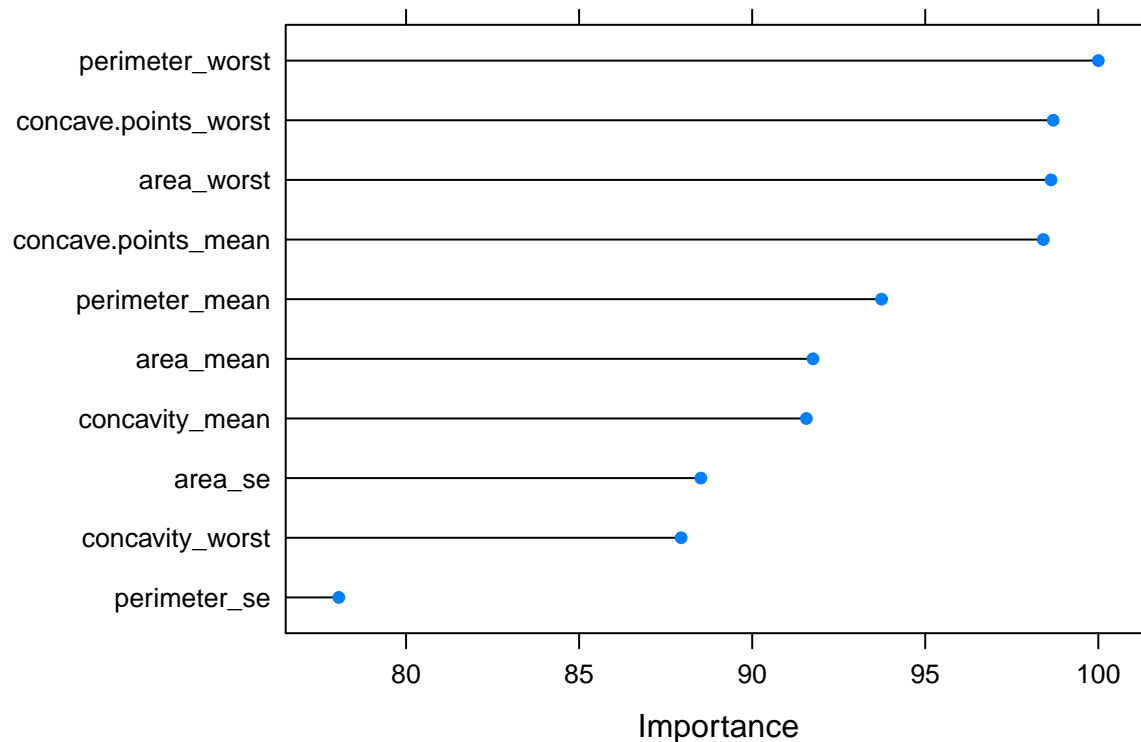
The most important variables that permit the best prediction and contribute the most to the model are the following:

```

plot(varImp(model_knn), top=10, main="Top variables - KNN")

```

## Top variables – KNN



## Neural Network with PCA Model

Artificial Neural Networks (NN) are a types of mathematical algorithms originating in the simulation of networks of biological neurons. An artificial Neural Network consists of nodes (called neurons) and edges (called synapses). Input data is transmitted through the weighted synapses to the neurons where calculations are processed and then either sent to further neurons or represent the output.

Neural Networks take in the weights of connections between neurons . The weights are balanced, learning data point in the wake of learning data point . When all weights are trained, the neural network can be utilized to predict the class or a quantity, if there should arise an occurrence of regression of a new input data point. With Neural networks, extremely complex models can be trained and they can be utilized as a kind of black box, without playing out an unpredictable complex feature engineering before training the model. Joined with the “deep approach” even more unpredictable models can be picked up to realize new possibilities.

```
model_nnet_pca <- train(diagnosis~.,
                        train_data,
                        method="nnet",
                        metric="ROC",
                        preProcess=c('center', 'scale', 'pca'),
                        tuneLength=10,
                        trace=FALSE,
                        trControl=fitControl)

prediction_nnet_pca <- predict(model_nnet_pca, test_data)
confusionmatrix_nnet_pca <- confusionMatrix(prediction_nnet_pca, test_data$diagnosis, positive = "M")
confusionmatrix_nnet_pca
```

```

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##           B 71  2
##           M  0 40
##
##           Accuracy : 0.9823
##           95% CI : (0.9375, 0.9978)
##           No Information Rate : 0.6283
##           P-Value [Acc > NIR] : <2e-16
##
##           Kappa : 0.9617
##           Mcnemar's Test P-Value : 0.4795
##
##           Sensitivity : 0.9524
##           Specificity : 1.0000
##           Pos Pred Value : 1.0000
##           Neg Pred Value : 0.9726
##           Prevalence : 0.3717
##           Detection Rate : 0.3540
##           Detection Prevalence : 0.3540
##           Balanced Accuracy : 0.9762
##
##           'Positive' Class : M
##

```

The most important variables that permit the best prediction and contribute the most to the model are the following:

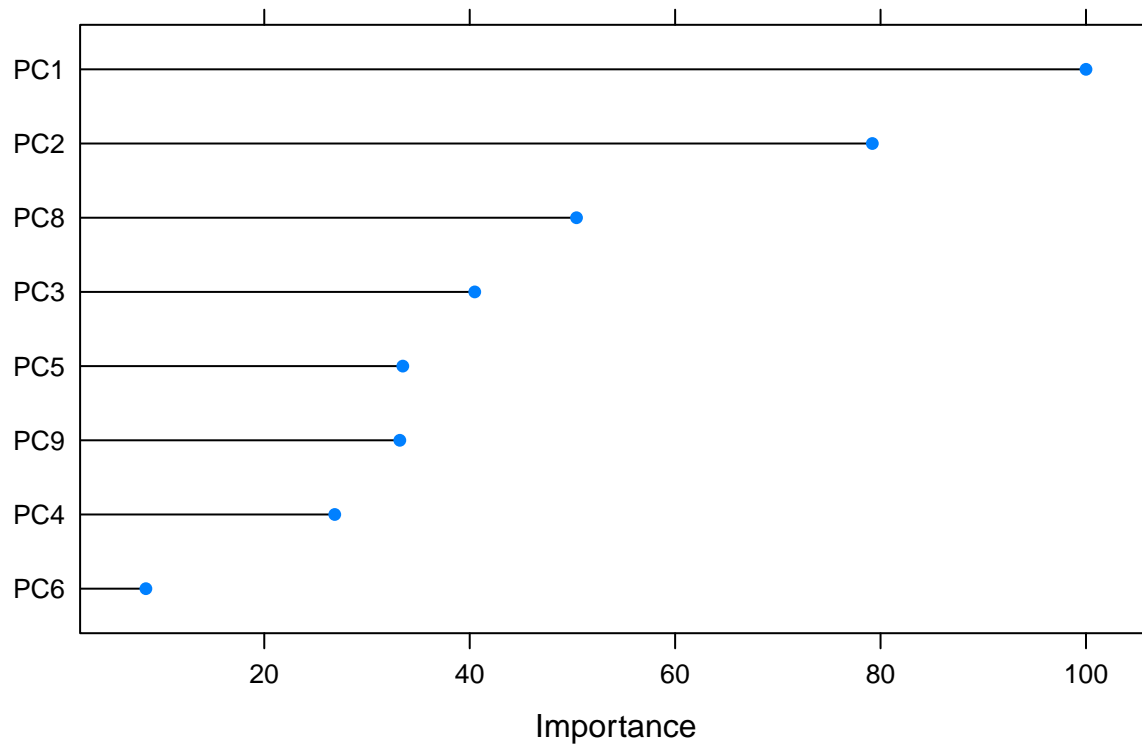
```

plot(varImp(model_nnet_pca), top=8, main="Top variables - NNET PCA")

```



## Top variables – NNET PCA



## Neural Network with LDA Model

Create a training and a test set of LDA data created in previous section:

```
train_data_lda <- lda_df_predict[data_sampling_index, ]
test_data_lda <- lda_df_predict[-data_sampling_index, ]
```

```
model_nnet_lda <- train(diagnosis~.,
                        train_data_lda,
                        method="nnet",
                        metric="ROC",
                        preProcess=c('center', 'scale'),
                        tuneLength=10,
                        trace=FALSE,
                        trControl=fitControl)
```

```
prediction_nnet_lda <- predict(model_nnet_lda, test_data_lda)
confusionmatrix_nnet_lda <- confusionMatrix(prediction_nnet_lda, test_data_lda$diagnosis, positive = "M")
confusionmatrix_nnet_lda
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##           B 71  1
```

```
##           M  0 41
##
##           Accuracy : 0.9912
##           95% CI : (0.9517, 0.9998)
##      No Information Rate : 0.6283
##      P-Value [Acc > NIR] : <2e-16
##
##           Kappa : 0.981
##  McNemar's Test P-Value : 1
##
##           Sensitivity : 0.9762
##           Specificity : 1.0000
##      Pos Pred Value : 1.0000
##      Neg Pred Value : 0.9861
##           Prevalence : 0.3717
##      Detection Rate : 0.3628
##      Detection Prevalence : 0.3628
##      Balanced Accuracy : 0.9881
##
##      'Positive' Class : M
##
```

## Results

One can now compare and evaluate the results obtained with the above calculations.

```
models_list <- list(Naive_Bayes=model_naiveb,
                    Logistic_regr=model_logreg,
                    Random_Forest=model_randomforest,
                    KNN=model_knn,
                    Neural_PCA=model_nnet_pca,
                    Neural_LDA=model_nnet_lda)
models_results <- resamples(models_list)

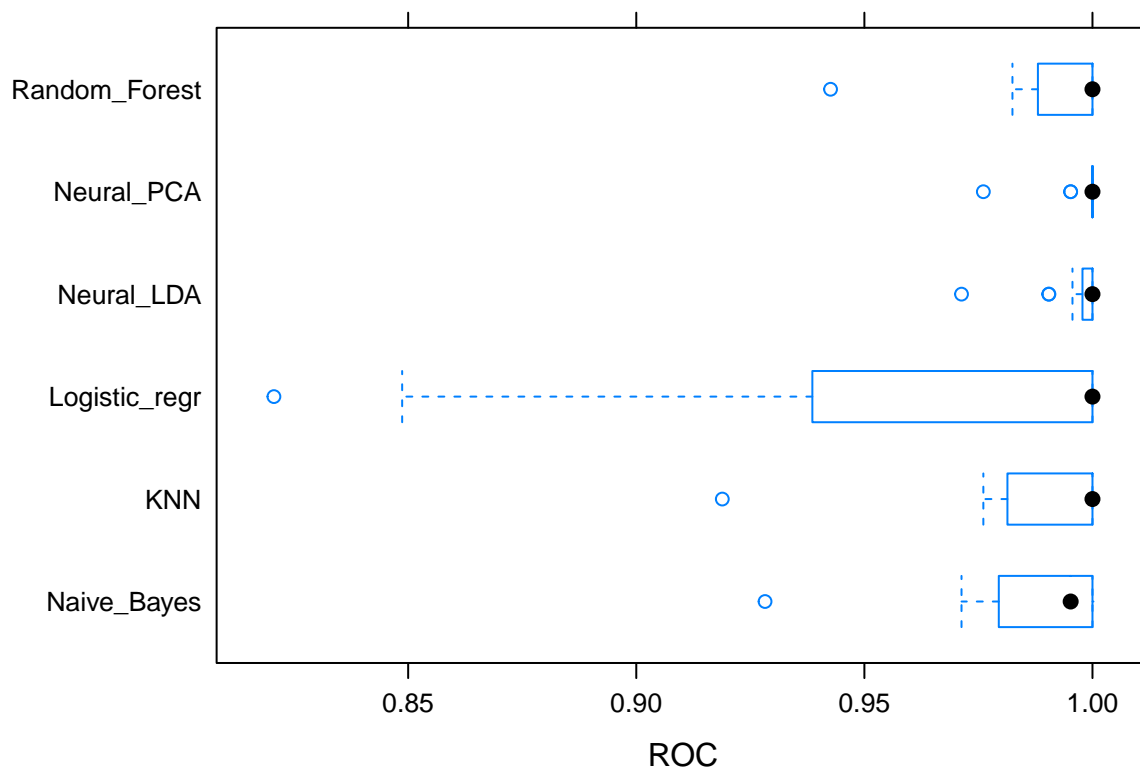
summary(models_results)
```

```
##
## Call:
## summary.resamples(object = models_results)
##
## Models: Naive_Bayes, Logistic_regr, Random_Forest, KNN, Neural_PCA, Neural_LDA
## Number of resamples: 15
##
## ROC
##           Min.    1st Qu.    Median    Mean 3rd Qu.  Max. NA's
## Naive_Bayes  0.9282297 0.9794657 0.9952153 0.9863636      1      1      0
## Logistic_regr 0.8205742 0.9385965 1.0000000 0.9603535      1      1      0
## Random_Forest 0.9425837 0.9880383 1.0000000 0.9916680      1      1      0
## KNN          0.9188596 0.9813596 1.0000000 0.9885965      1      1      0
## Neural_PCA    0.9760766 1.0000000 1.0000000 0.9977671      1      1      0
## Neural_LDA    0.9712919 0.9978070 1.0000000 0.9965178      1      1      0
##
```

```
## Sens
##           Min.    1st Qu.    Median    Mean 3rd Qu.  Max. NA's
## Naive_Bayes 0.8421053 0.9210526 0.9473684 0.9508772      1      1      0
## Logistic_regr 0.8421053 0.9473684 0.9473684 0.9508772      1      1      0
## Random_Forest 0.8947368 0.9473684 1.0000000 0.9719298      1      1      0
## KNN          0.9473684 1.0000000 1.0000000 0.9964912      1      1      0
## Neural_PCA   0.9473684 0.9736842 1.0000000 0.9859649      1      1      0
## Neural_LDA   0.8947368 0.9736842 1.0000000 0.9824561      1      1      0
##
## Spec
##           Min.    1st Qu.    Median    Mean 3rd Qu.  Max. NA's
## Naive_Bayes 0.7272727 0.8257576 0.9090909 0.8994949      1      1      0
## Logistic_regr 0.7500000 0.8712121 1.0000000 0.9419192      1      1      0
## Random_Forest 0.6363636 0.8636364 0.9166667 0.9116162      1      1      0
## KNN          0.7272727 0.9090909 0.9166667 0.9131313      1      1      0
## Neural_PCA   0.8181818 0.9090909 1.0000000 0.9525253      1      1      0
## Neural_LDA   0.7272727 0.9090909 1.0000000 0.9520202      1      1      0
```

From the following plot, one can observe two models, Naive\_bayes and Logistic\_regr have great variability, depending of the processed sample :

```
bwplot(models_results, metric="ROC")
```



The Neural Network LDA model achieve a great auc (Area Under the ROC Curve) with some variability. The ROC (Receiver Operating characteristic Curve is a graph showing the performance of a classification model at all classification thresholds) metric measure the auc of the roc curve of each model. This metric is independent of any threshold. Let's remember how these models result with the testing dataset. Prediction classes are obtained by default with a threshold of 0.5 which could not be the best with an unbalanced dataset like this.

```

confusionmatrix_list <- list(
  Naive_Bayes=confusionmatrix_naiveb,
  Logistic_regr=confusionmatrix_logreg,
  Random_Forest=confusionmatrix_randomforest,
  KNN=confusionmatrix_knn,
  Neural_PCA=confusionmatrix_nnet_pca,
  Neural_LDA=confusionmatrix_nnet_lda)
confusionmatrix_list_results <- sapply(confusionmatrix_list, function(x) x$byClass)
confusionmatrix_list_results %>% knitr::kable()

```

	Naive_Bayes	Logistic_regr	Random_Forest	KNN	Neural_PCA	Neural_LDA
Sensitivity	0.8809524	0.9523810	0.9285714	0.8809524	0.9523810	0.9761905
Specificity	0.9718310	1.0000000	1.0000000	0.9859155	1.0000000	1.0000000
Pos Pred Value	0.9487179	1.0000000	1.0000000	0.9736842	1.0000000	1.0000000
Neg Pred Value	0.9324324	0.9726027	0.9594595	0.9333333	0.9726027	0.9861111
Precision	0.9487179	1.0000000	1.0000000	0.9736842	1.0000000	1.0000000
Recall	0.8809524	0.9523810	0.9285714	0.8809524	0.9523810	0.9761905
F1	0.9135802	0.9756098	0.9629630	0.9250000	0.9756098	0.9879518
Prevalence	0.3716814	0.3716814	0.3716814	0.3716814	0.3716814	0.3716814
Detection Rate	0.3274336	0.3539823	0.3451327	0.3274336	0.3539823	0.3628319
Detection Prevalence	0.3451327	0.3539823	0.3451327	0.3362832	0.3539823	0.3628319
Balanced Accuracy	0.9263917	0.9761905	0.9642857	0.9334339	0.9761905	0.9880952

## Discussion

Below the metrics that will compare in this section have been described.

Accuracy is our starting point. It is the number of correct predictions made divided by the total number of predictions made, multiplied by 100 to turn it into a percentage.

Precision is the number of True Positives divided by the number of True Positives and False Positives. Put another way, it is the number of positive predictions divided by the total number of positive class values predicted. It is also called the Positive Predictive Value (PPV). A low precision can also indicate a large number of False Positives.

Recall (Sensitivity) is the number of True Positives divided by the number of True Positives and the number of False Negatives. Put another way it is the number of positive predictions divided by the number of positive class values in the test data. It is also called Sensitivity or the True Positive Rate. Recall can be thought of as a measure of a classifiers completeness. A low recall indicates many False Negatives.

The F1 Score is the  $2 \times ((\text{precision} \times \text{recall}) / (\text{precision} + \text{recall}))$ . It is also called the F Score or the F Measure. Put another way, the F1 score conveys the balance between the precision and the recall.

The best results for sensitivity (detection of breast cancer malign cases) is Neural Network with LDA model which also has a great F1 score.

```

confusionmatrix_results_max <- apply(confusionmatrix_list_results, 1, which.is.max)

output_report <- data.frame(metric=names(confusionmatrix_results_max),
                             best_model=colnames(confusionmatrix_list_results)[confusionmatrix_results_max[,1]],
                             value=mapply(function(x,y) {confusionmatrix_list_results[x,y]},
                                             names(confusionmatrix_results_max),
                                             confusionmatrix_results_max))

```

```
rownames(output_report) <- NULL
output_report
```

##		metric	best_model	value
## 1		Sensitivity	Neural_LDA	0.9761905
## 2		Specificity	Logistic_regr	1.0000000
## 3		Pos Pred Value	Neural_LDA	1.0000000
## 4		Neg Pred Value	Neural_LDA	0.9861111
## 5		Precision	Logistic_regr	1.0000000
## 6		Recall	Neural_LDA	0.9761905
## 7		F1	Neural_LDA	0.9879518
## 8		Prevalence	Logistic_regr	0.3716814
## 9		Detection Rate	Neural_LDA	0.3628319
## 10		Detection Prevalence	Neural_LDA	0.3628319
## 11		Balanced Accuracy	Neural_LDA	0.9880952

## Conclusion

This paper treats the Wisconsin Madison Breast Cancer diagnosis problem as a pattern classification problem. In this report several machine learning model were investigated and the optimal model was selected by choosing a high accuracy level combined with a low rate of false-negatives (the means that the metric is high sensitivity).

The Neural Network with LDA model had the optimal results for F1 (0.9879518), Sensitivity (0.9761905) and Balanced Accuracy (0.9880952)