Breast Cancer Project

Lucas Perondi Kist

2024-01-31

Introduction

This report aims to provide application of the concepts learned in the courses Data Science: Machine Learning (from HarvardX) and Supervised Machine Learning (ME906, from Unicamp). Thus, it is presented an application on the dataset `brca`, from the package `dslabs`, with biopsy features for the classification of 569 malignant (cancer) and benign (not cancer) breast masses.

The objective is to build a classifier that, based on the values of the features, tries to predict if a given breast mass is malignant or benign. For this purpose, GLMs and several machine learning models were adjusted, whose parameters were tuned on a training set maximizing the *F1* metric. After the training, all the models were evaluated on a test set and some were combined to build ensembles. # Dataset The dataset `brca` contains 569 rows and 30 features. The outcome `y` is a factor with two levels: "M" (malignant) and "B" (benign). The predictors are the mean, standard error and worst value of the following measurements on the slide:

- Radius: nucleus radius (mean of distances from the center to points on the perimeter);
- Texture: nucleus texture (standard deviation of grayscale values).
- Perimeter: nucleus perimeter.
- Area: nucleus area.
- Smoothness: nucleus smoothness (local variation in radius lengths).
- Compactness: nucleus compactness ($perimeter^2/area - 1$).
- Concavity: nucleus concavity (severity of concave portions of the contour).
- Concave_pts: number of concave portions of the nucleus contour.
- Symmetry: nucleus symmetry.
- Fractal_dim: nucleus fractal dimension ("coastline approximation" -1).

```r
data(brca)
brca <- data.frame(brca$x, y = factor(brca$y,
                                      levels = c("M", "B"))) %>%
  tibble()
#summary(brca) %>% kable()
```

Pre-processing

The `brca` dataset was divided into two sets: training and test, in a proportion of approximately 80% and 20%. Then, the predictors were standardized in each one.

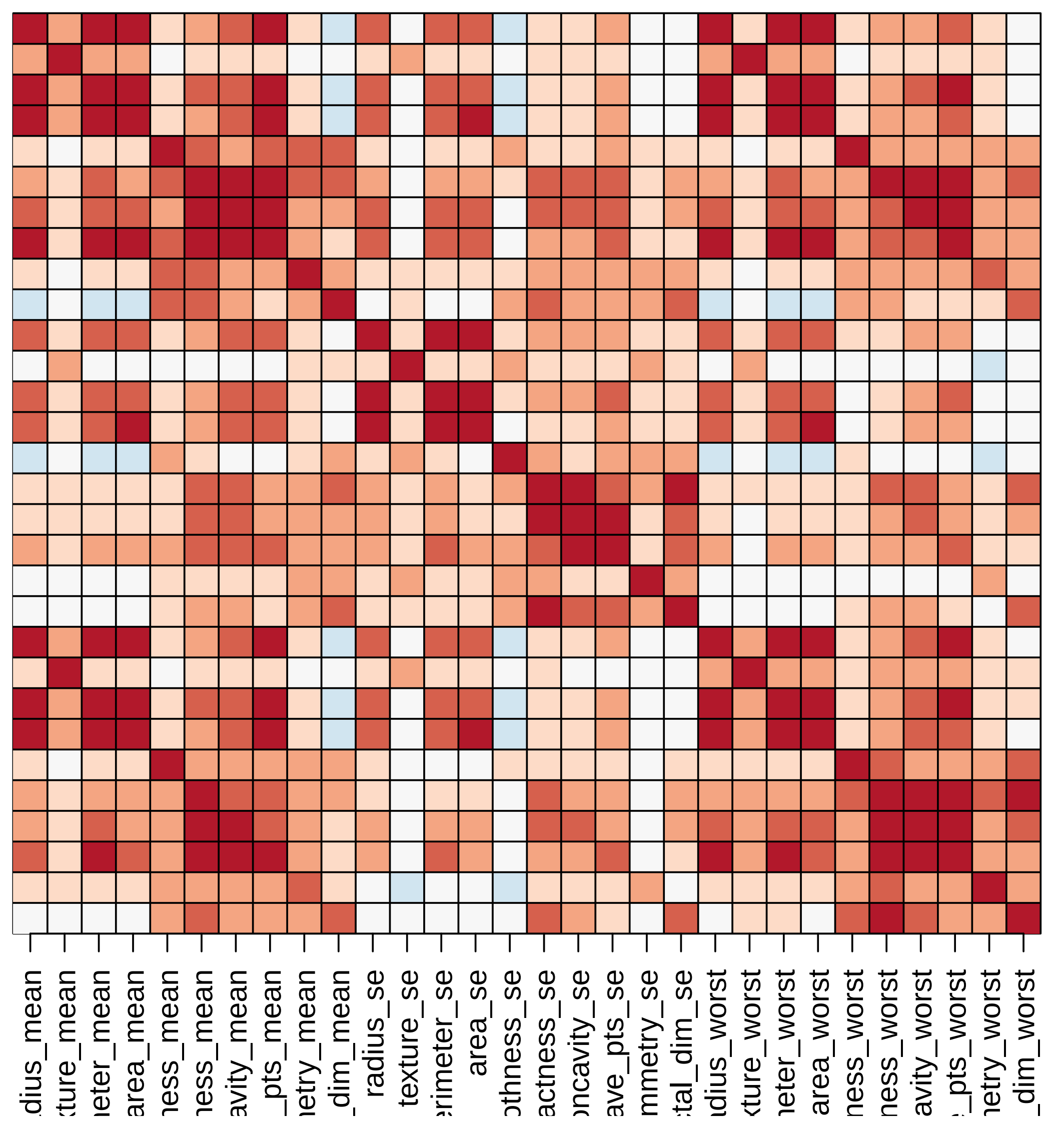
```
set.seed(3)
test_idx <- createDataPartition(brca$y, times = 1, p = 0.2, list = F)
train <- brca[-test_idx, ]
test <- brca[test_idx, ]
p <- ncol(brca)
train[,1:(p-1)] <- apply(train[,1:(p-1)], 2,
                                   function(x) scale(x))
test[,1:(p-1)] <- apply(test[,1:(p-1)], 2,
                                   function(x) scale(x))
```

Descriptive Analysis

The following analysis was done considering only the training set. The correlation of the features is shown below. Based on it, it is possible to see that there are some groups with very high correlation (>0.99) and, then, it is needed to select one from each group to improve the performance of the methods (especially in terms of computational time).

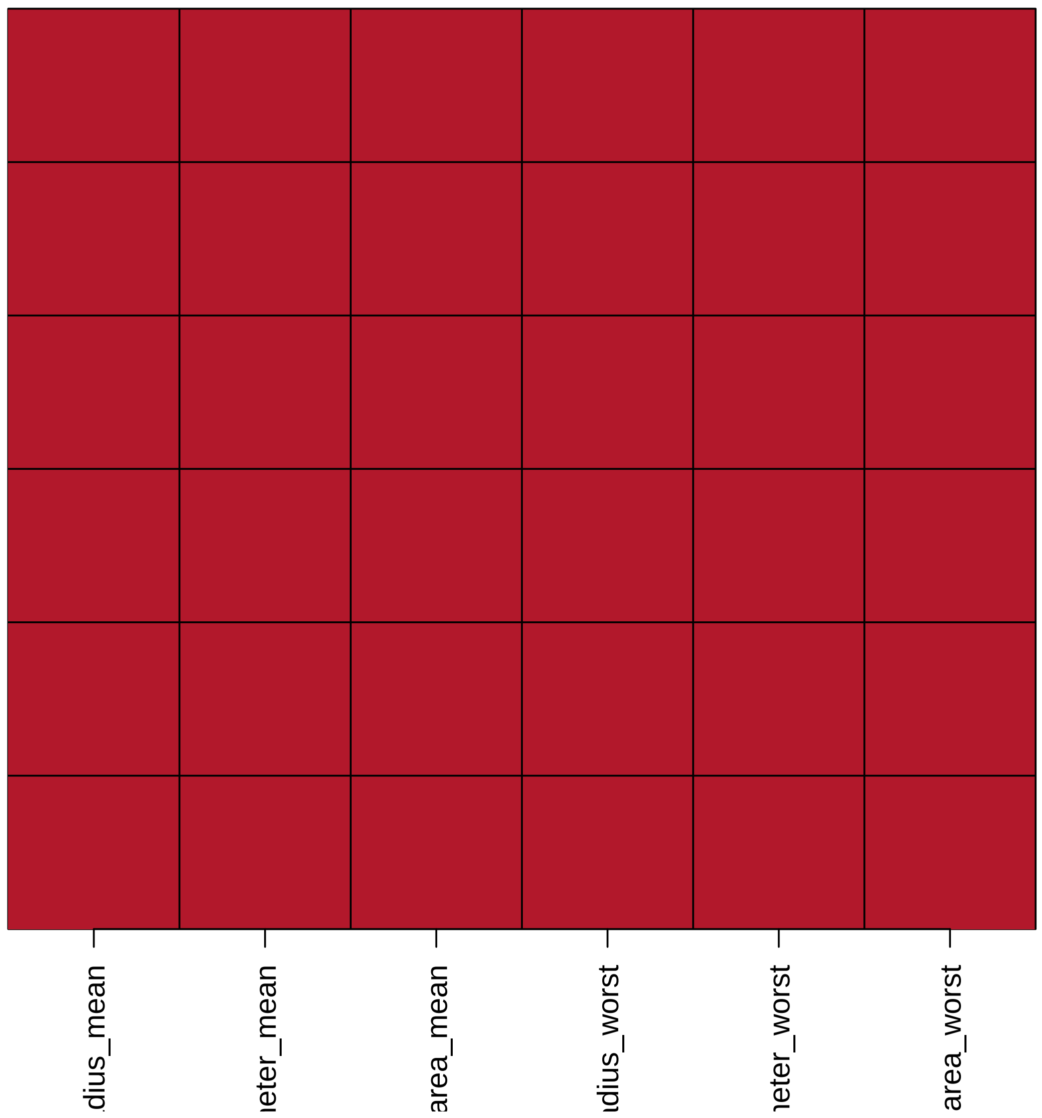
```
my_image <- function(x, zlim = range(x), ...){
  colors = rev(RColorBrewer::brewer.pal(9, "RdBu"))
  cols <- 1:ncol(x)
  rows <- 1:nrow(x)
  image(cols, rows, t(x[rev(rows),,drop=FALSE]), xaxt = "n", yaxt = "n",
        xlab="", ylab="",  col = colors, zlim = zlim, ...)
  abline(h=rows + 0.5, v = cols + 0.5)
  axis(side = 1, cols, colnames(x), las = 2)
}

my_image(cor(train[, 1:(p-1)]), zlim = c(-1,1))
```



The criteria used to select a feature from a group was to keep the one that has the biggest absolute value of the *t statistic*, that is, it is the predictor with the greater difference in the means of different outcomes. The first group is "radius_mean", "perimeter_mean", "area_mean", "radius_worst", "perimeter_worst" and "area_worst". The correlation is shown below and the selected feature is "perimeter_worst".

```r
stat_t <- function(col){
  t.test(col[train$y == "B"],
         col[train$y == "M"])$statistic
}
stats_t <- apply(train[,1:(p-1)], 2, stat_t)
my_image(cor(train[,c("radius_mean", 'perimeter_mean', 'area_mean',
                  'radius_worst', 'perimeter_worst', 'area_worst')]),
         zlim = c(-1, 1))
```



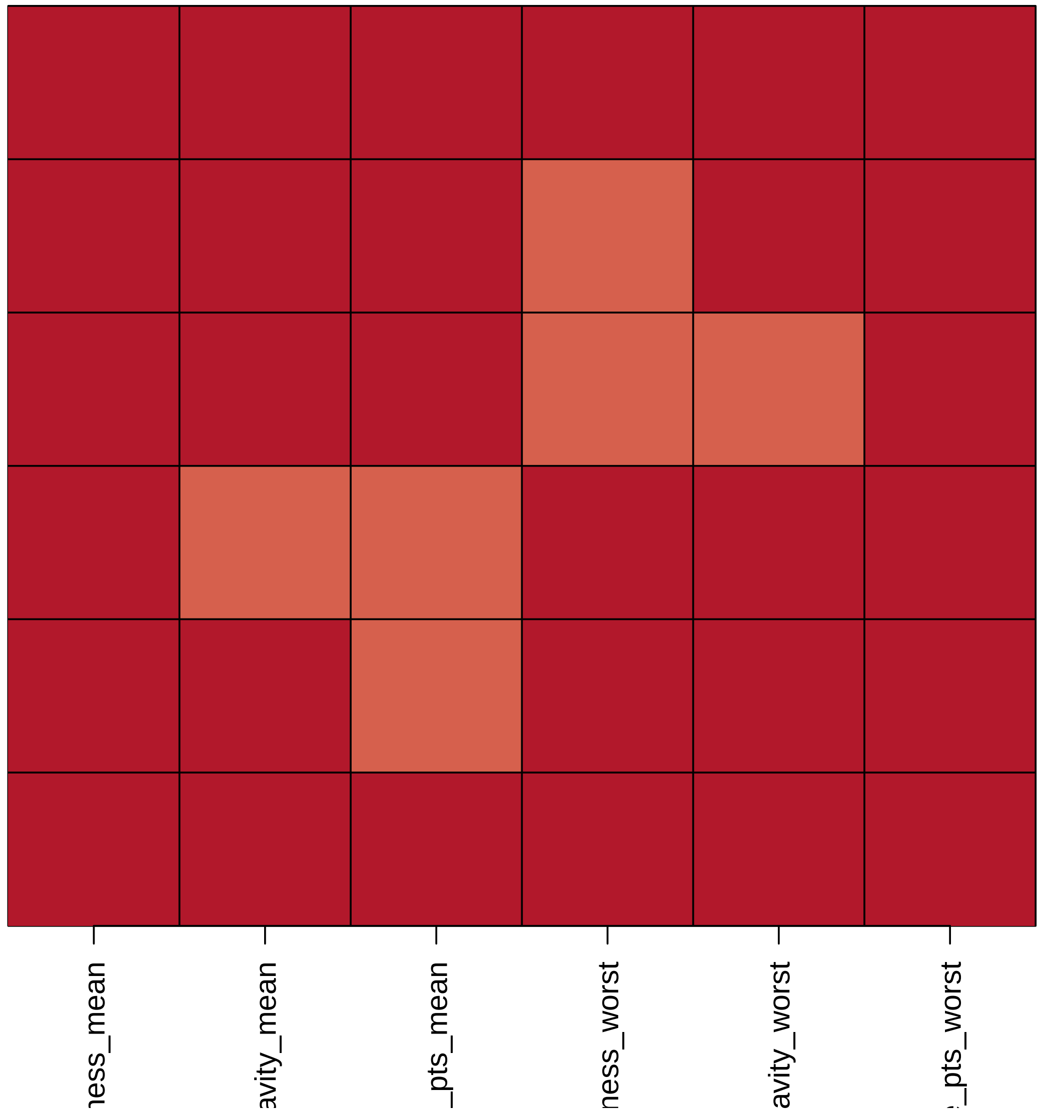
```
which.max(abs(stats_t[c("radius_mean", 'perimeter_mean', 'area_mean',
                   'radius_worst', 'perimeter_worst', 'area_worst')]))
```

```
## perimeter_worst
##               5
```

```
train <- train[, !colnames(train) %in% c("radius_mean", 'perimeter_mean',
                                          'area_mean','radius_worst',
                                          'area_worst')]
```

The second group is compactness_mean", "concavity_mean", "concave_pts_mean", "compactness_worst", "concavity_worst" and "concave_pts_worst". The correlation is shown below and the selected feature is "concave_pts_worst".

```r
my_image(cor(train[, c("compactness_mean", "concavity_mean",
                       "concave_pts_mean", "compactness_worst",
                       "concavity_worst", "concave_pts_worst")]),
         zlim = c(-1,1))
```



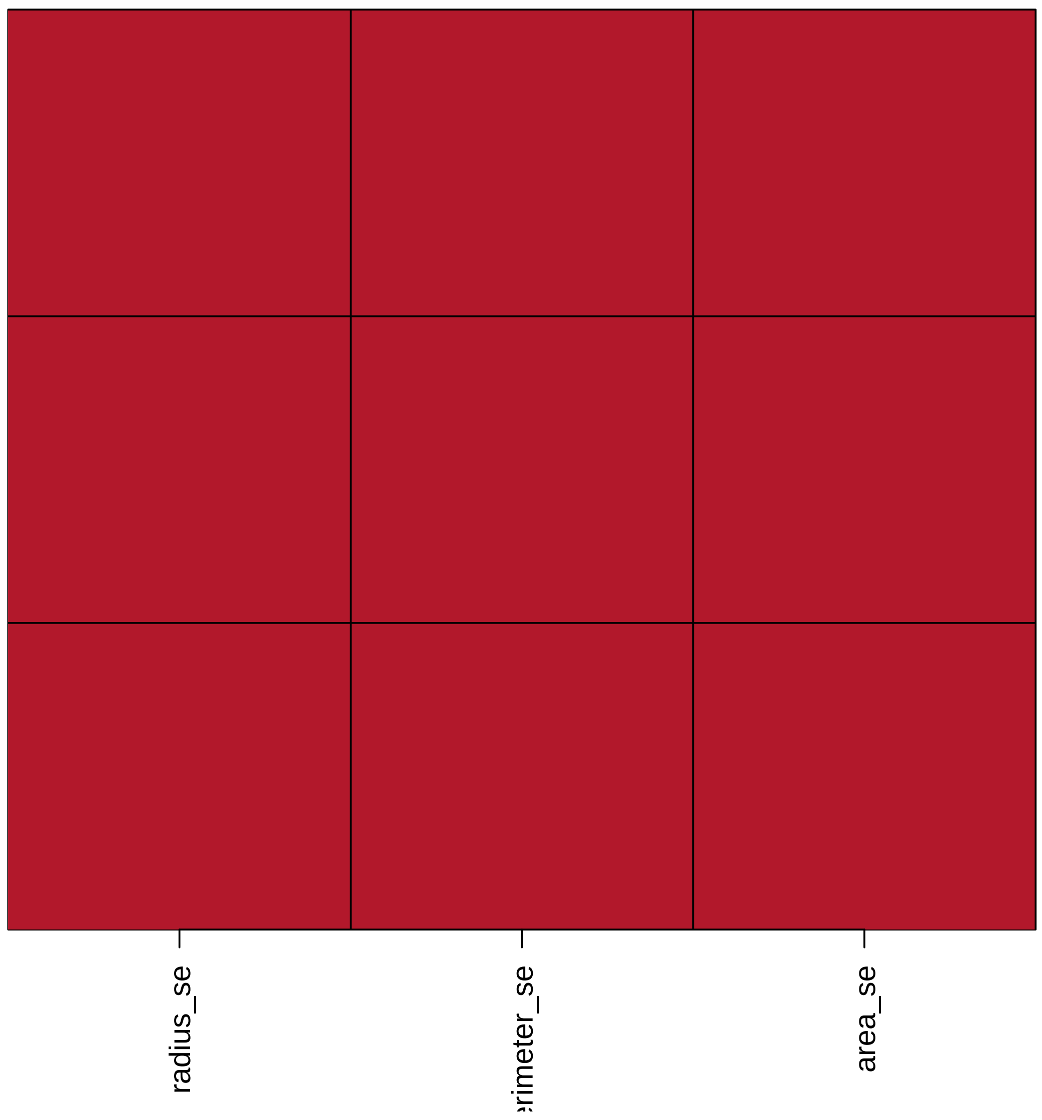
```r
which.max(abs(stats_t[c("compactness_mean", "concavity_mean",
                       "concave_pts_mean", "compactness_worst",
                       "concavity_worst", "concave_pts_worst")]))
```

```
## concave_pts_worst
##                 6
```

```r
train <- train[, !colnames(train) %in%
                 c("compactness_mean", "concavity_mean",
                   "concave_pts_mean", "compactness_worst",
                   "concavity_worst")]
```

The third group is “radius_se”, “perimeter_se” and “area_se”. The correlation is shown below and the selected feature is “radius_se”.

```r
my_image(cor(train[, c("radius_se", "perimeter_se",
                       "area_se")]),
         zlim = c(-1,1))
```



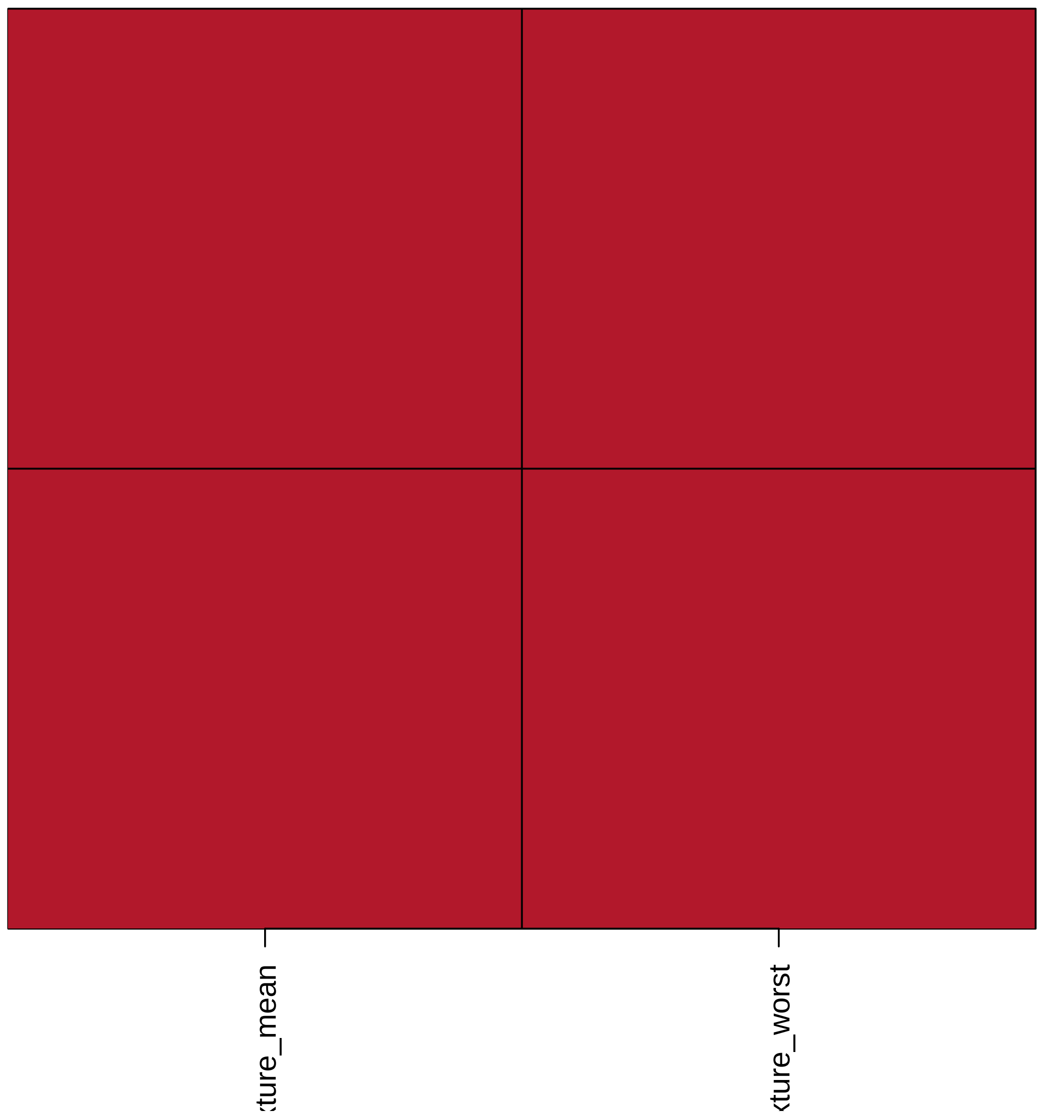
```r
which.max(abs(stats_t[c("radius_se", "perimeter_se",
                       "area_se")]))
```

```
## radius_se
##         1
```

```r
train <- train[, !colnames(train) %in%
                 c("perimeter_se",
                       "area_se")]
```

The fourth group is "texture_mean" and "texture_worst". The correlation is shown below and the selected feature is "texture_worst".

```r
my_image(cor(train[,c("texture_mean", "texture_worst")]),
         z = c(-1,1))
```



```r
which.max(abs(stats_t[c("texture_mean", "texture_worst")]))
```

```
## texture_worst
##             2
```

```r
train <- train[, !colnames(train) %in%
                 c("texture_mean")]
```

After removing the highly correlated predictors referred to above, the 3 biggest correlations are the following. It is possible to note that, although they are still high, it is not needed to remove them from the features set. After this selection, the same predictors were kept in the test set.

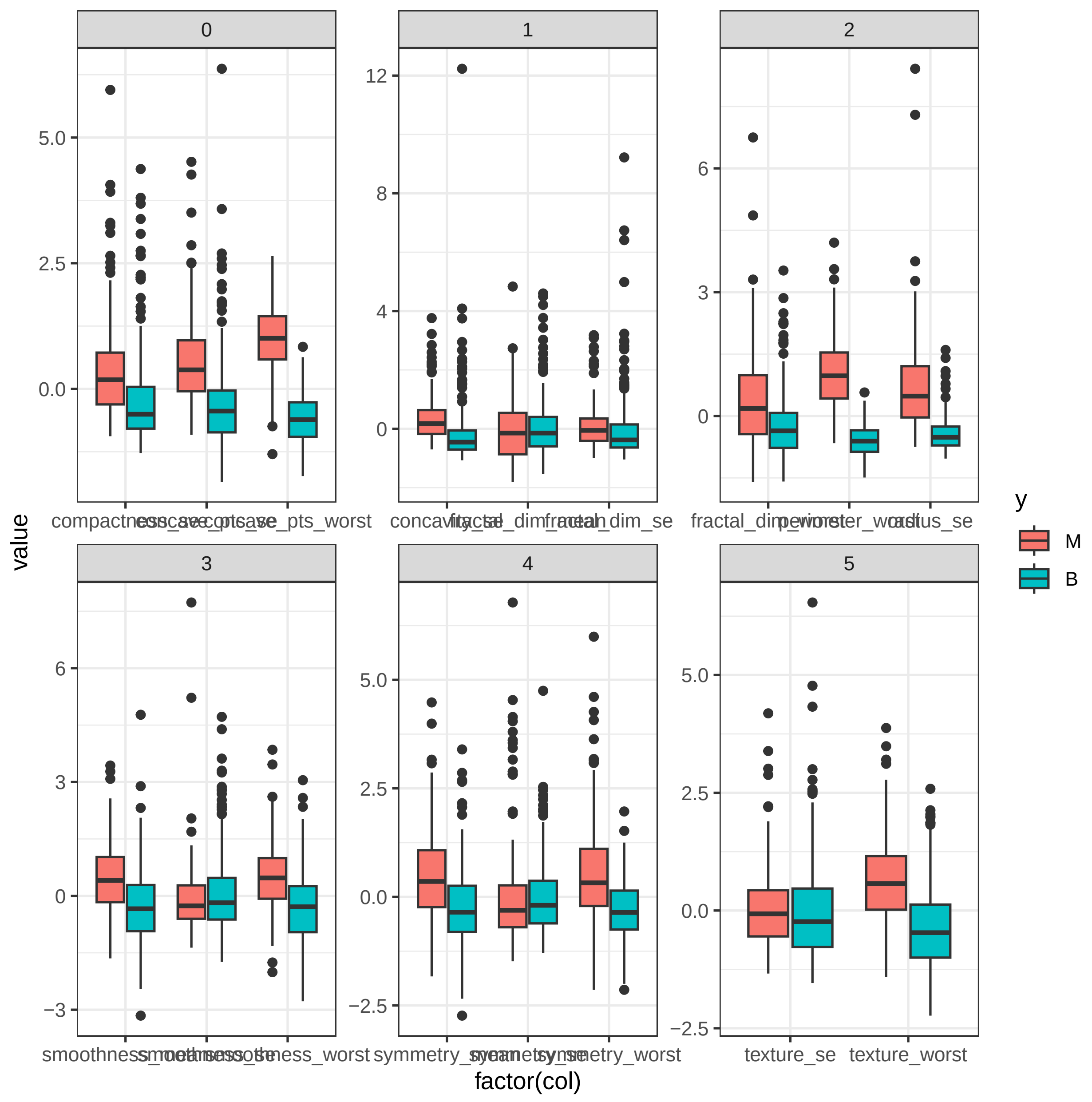
```r
(cor(train[,1:17]) %>% data.frame(c1 = colnames(train)[1:17]) %>%
  pivot_longer(cols = 1:17, names_to = "c2", values_to = "cor") %>%
  filter(cor!=1) %>%
  arrange(-abs(cor)) %>% head())[c(1,3,5),] %>% kable()
```

c1	c2	cor
perimeter_worst	concave_pts_worst	0.8228342
compactness_se	concavity_se	0.8101561
compactness_se	fractal_dim_se	0.8061178

```r
p <- ncol(train)
test <- test[, colnames(test) %in% colnames(train)]
```

The boxplots of the remaining features by the value of the outcome are below. Based on them, it is possible to see there is a big difference in the values according to y. Thus, it is expected it is possible to build a good classifier based on them.

```r
train %>% pivot_longer(cols = 1:(p-1), names_to = "col",
                       values_to = "value") %>%
  mutate(col = factor(col),
         idx = factor((as.numeric(col)-1)%/%3)) %>%
  ggplot(aes(x = factor(col), y = value, fill = y))+
  geom_boxplot()+
  facet_wrap(~idx, scales = "free")+
  theme_bw()
```



Adjusting models

The best tuning parameters of the models were selected based on the *F1* value. Then, a function that calculates it was defined and used to compute the *F1* for each combination of the tuning grid (when it was needed). Also, `train.control` is defined, which contains the details of the way the validation would be performed: a ten-fold cross-validation.

```
f1 <- function(data, lev = NULL, model = NULL) {
        f1_val <- MLmetrics::F1_Score(y_pred = data$pred,
                                      y_true = data$obs,
                                      positive = lev[1])
        c(F1 = f1_val)
}
train.control <- trainControl(method = "cv",
                              number = 10,
                              p = 0.9,
                              summaryFunction = f1)
```

GLMs

Firstly, Generalized Linear Models were adjusted using the link functions *logit*, *probit* and *cloglog*. The models were sequentially fitted to keep only the significant predictors and tested again to the original model to compare the fits (the deviance was compared to the difference of degrees of freedom).

```
logit <- glm(y ~ ., data = train, family = binomial(link = "logit"))
coef(summary(logit)) %>% kable()
```

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-1.2373327	1.070178	-1.1561929	0.2476023
smoothness_mean	-0.0765922	2.418022	-0.0316756	0.9747308
symmetry_mean	2.5462528	1.775343	1.4342318	0.1515061
fractal_dim_mean	1.2222961	2.125622	0.5750300	0.5652710
radius_se	-17.4070536	6.608068	-2.6342121	0.0084333
texture_se	0.3595423	1.174614	0.3060941	0.7595330
smoothness_se	0.2620612	1.132747	0.2313501	0.8170428
compactness_se	4.5113647	2.134585	2.1134619	0.0345613
concavity_se	-3.7966989	1.996238	-1.9019269	0.0571807
concave_pts_se	-6.5522290	4.030449	-1.6256819	0.1040173
symmetry_se	-1.0026415	1.707051	-0.5873530	0.5569667
fractal_dim_se	9.6352401	3.901342	2.4697243	0.0135217
texture_worst	-4.5998336	1.647313	-2.7923245	0.0052331
perimeter_worst	-15.4467505	6.331762	-2.4395659	0.0147049
smoothness_worst	-4.0011877	2.822610	-1.4175489	0.1563225
concave_pts_worst	2.8056450	4.214759	0.6656715	0.5056211
symmetry_worst	-2.5353162	2.352744	-1.0775997	0.2812124
fractal_dim_worst	-9.7121441	4.088377	-2.3755498	0.0175228

```
logit2 <- glm(y ~ radius_se + smoothness_se + fractal_dim_se + texture_worst + perimeter_worst + concave
              data = train,
              family = binomial(link = "logit"))
coef(summary(logit2)) %>% kable()
```

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	0.0280895	0.5221691	0.0537938	0.9570994
radius_se	-7.9834749	2.3601229	-3.3826522	0.0007179
smoothness_se	-1.9130951	0.7214354	-2.6517899	0.0080066
fractal_dim_se	5.3111709	1.8067719	2.9395912	0.0032865
texture_worst	-2.3712812	0.6547223	-3.6218122	0.0002925
perimeter_worst	-6.5557605	1.8276700	-3.5869498	0.0003346
concave_pts_worst	-4.3190765	1.3388553	-3.2259471	0.0012556

	Estimate	Std. Error	z value	Pr(>|z|)
fractal_dim_worst	-3.5265944	1.4383625	-2.4518121	0.0142139

```r
anova(logit2, logit)
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ radius_se + smoothness_se + fractal_dim_se + texture_worst +
##     perimeter_worst + concave_pts_worst + fractal_dim_worst
## Model 2: y ~ smoothness_mean + symmetry_mean + fractal_dim_mean + radius_se +
##     texture_se + smoothness_se + compactness_se + concavity_se +
##     concave_pts_se + symmetry_se + fractal_dim_se + texture_worst +
##     perimeter_worst + smoothness_worst + concave_pts_worst +
##     symmetry_worst + fractal_dim_worst
##   Resid. Df Resid. Dev Df Deviance
## 1       446     44.429
## 2       436     30.005 10   14.424
```

```r
probit <- glm(y ~ ., data = train, family = binomial(link = "probit"))
coef(summary(logit)) %>% kable()
```

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-1.2373327	1.070178	-1.1561929	0.2476023
smoothness_mean	-0.0765922	2.418022	-0.0316756	0.9747308
symmetry_mean	2.5462528	1.775343	1.4342318	0.1515061
fractal_dim_mean	1.2222961	2.125622	0.5750300	0.5652710
radius_se	-17.4070536	6.608068	-2.6342121	0.0084333
texture_se	0.3595423	1.174614	0.3060941	0.7595330
smoothness_se	0.2620612	1.132747	0.2313501	0.8170428
compactness_se	4.5113647	2.134585	2.1134619	0.0345613
concavity_se	-3.7966989	1.996238	-1.9019269	0.0571807
concave_pts_se	-6.5522290	4.030449	-1.6256819	0.1040173
symmetry_se	-1.0026415	1.707051	-0.5873530	0.5569667
fractal_dim_se	9.6352401	3.901342	2.4697243	0.0135217
texture_worst	-4.5998336	1.647313	-2.7923245	0.0052331
perimeter_worst	-15.4467505	6.331762	-2.4395659	0.0147049
smoothness_worst	-4.0011877	2.822610	-1.4175489	0.1563225
concave_pts_worst	2.8056450	4.214759	0.6656715	0.5056211
symmetry_worst	-2.5353162	2.352744	-1.0775997	0.2812124
fractal_dim_worst	-9.7121441	4.088377	-2.3755498	0.0175228

```r
probit2 <- glm(y ~ radius_se + smoothness_se + fractal_dim_se + texture_worst + perimeter_worst + conca
               data = train, family = binomial(link = "probit"))
coef(summary(probit2))
```

```
##                       Estimate Std. Error     z value     Pr(>|z|)
## (Intercept)       -0.005064185  0.2743815 -0.01845673 9.852745e-01
## radius_se         -4.153544089  1.1229730 -3.69870343 2.167036e-04
## smoothness_se     -0.955607316  0.3551530 -2.69069183 7.130403e-03
## fractal_dim_se     2.686902322  0.8754223  3.06926418 2.145867e-03
## texture_worst     -1.306039018  0.3270960 -3.99283087 6.528914e-05
## perimeter_worst   -3.303494644  0.8775314 -3.76453141 1.668617e-04
## concave_pts_worst -2.309175614  0.7054422 -3.27337315 1.062721e-03
## fractal_dim_worst -1.719299901  0.7007520 -2.45350686 1.414708e-02
```

```
anova(probit2, probit)
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ radius_se + smoothness_se + fractal_dim_se + texture_worst +
##     perimeter_worst + concave_pts_worst + fractal_dim_worst
## Model 2: y ~ smoothness_mean + symmetry_mean + fractal_dim_mean + radius_se +
##     texture_se + smoothness_se + compactness_se + concavity_se +
##     concave_pts_se + symmetry_se + fractal_dim_se + texture_worst +
##     perimeter_worst + smoothness_worst + concave_pts_worst +
##     symmetry_worst + fractal_dim_worst
##   Resid. Df Resid. Dev Df Deviance
## 1       446     45.578
## 2       436     31.063 10   14.515
```

```
cloglog <- glm(y ~ ., data = train, family = binomial(link = "cloglog"))
cloglog2 <- glm(y ~ radius_se + smoothness_se + fractal_dim_se + texture_worst + perimeter_worst + conca
                data = train, family = binomial(link = "cloglog"))
coef(summary(cloglog2))
```

```
##                     Estimate Std. Error   z value     Pr(>|z|)
## (Intercept)       -0.5094783  0.2729843 -1.866328 6.199543e-02
## radius_se         -4.1560635  0.9769537 -4.254105 2.098871e-05
## smoothness_se     -0.9246067  0.3367811 -2.745424 6.043282e-03
## fractal_dim_se     2.7428156  0.8160096  3.361254 7.758936e-04
## texture_worst     -1.4138603  0.3336736 -4.237256 2.262686e-05
## perimeter_worst   -3.1332338  0.8093771 -3.871167 1.083156e-04
## concave_pts_worst -2.2343383  0.6926811 -3.225638 1.256924e-03
## fractal_dim_worst -1.6862979  0.6200867 -2.719455 6.538961e-03
```

```
anova(cloglog2, cloglog)
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ radius_se + smoothness_se + fractal_dim_se + texture_worst +
##     perimeter_worst + concave_pts_worst + fractal_dim_worst
## Model 2: y ~ smoothness_mean + symmetry_mean + fractal_dim_mean + radius_se +
##     texture_se + smoothness_se + compactness_se + concavity_se +
##     concave_pts_se + symmetry_se + fractal_dim_se + texture_worst +
##     perimeter_worst + smoothness_worst + concave_pts_worst +
##     symmetry_worst + fractal_dim_worst
##   Resid. Df Resid. Dev Df Deviance
## 1       446     50.489
## 2       436     33.480 10   17.009
```

LDA and QDA

All the other models were fitted using the ‘`caret::train`function, which performs a ten-fold cross-validation. Then, Linear and Quadratic Discriminant Analyses were done. The results were saved and the values of *F1* after the validation are shown in the table below.

```r
lda <- train(y~., data = train,
             method = "lda",
             metric = "F1",
             trControl = train.control)
qda <- train(y~., data = train,
             method = "qda",
             metric = "F1",
             trControl = train.control)
kable(rbind(cbind(model = "LDA", lda$results[2:3]),
            cbind(model = "QDA", qda$results[2:3])))
```

model	F1	F1SD
LDA	0.9366752	0.0460897
QDA	0.9464129	0.0196392

Naive Bayes

The Naive Bayes model was fitted using and not using kernel. The results are shown in the table below.

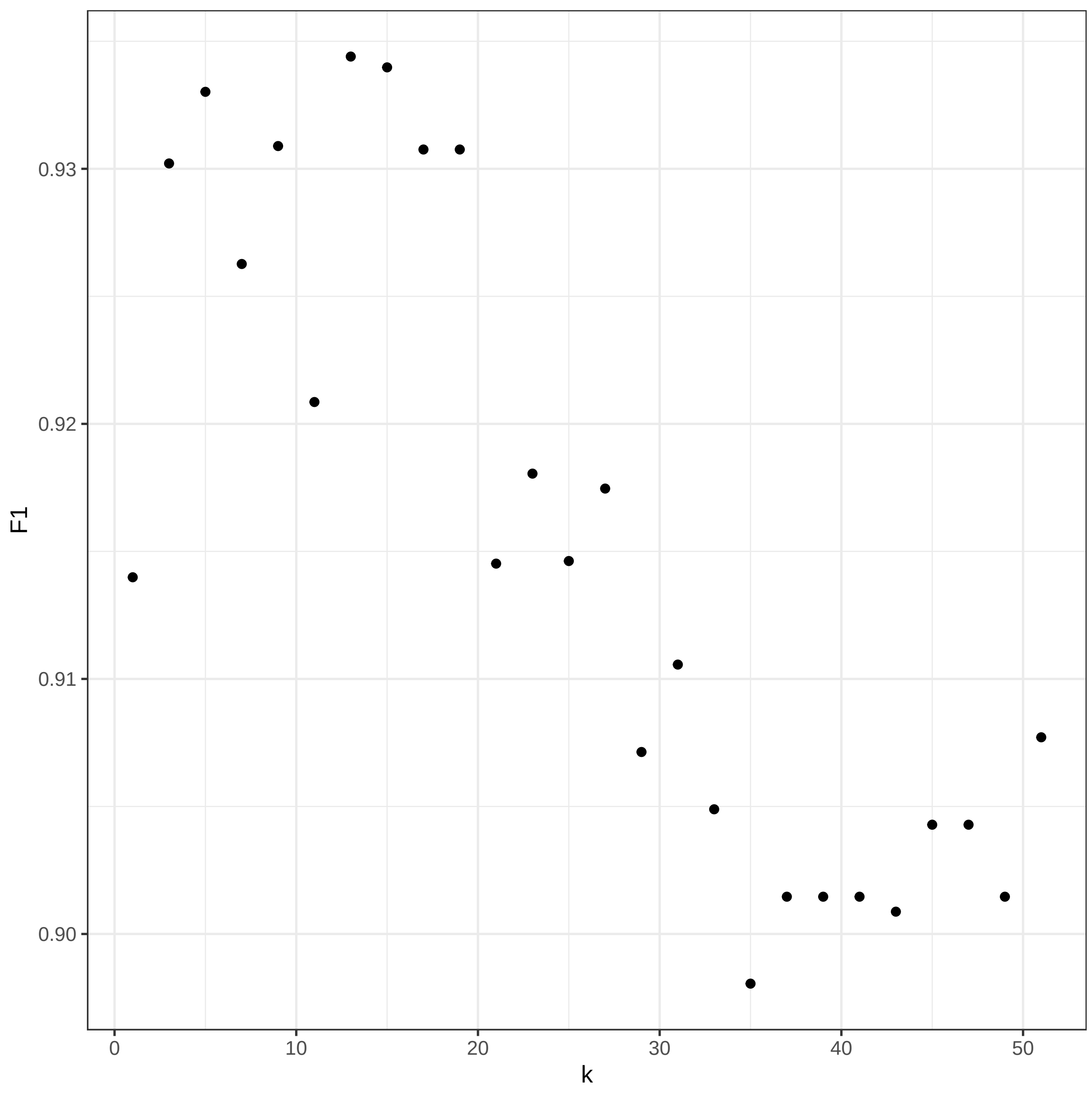
```r
naive_bayes <- train(y~., data = train,
             method = "naive_bayes",
             metric = "F1",
             trControl = train.control)
kable(cbind(model = "Naive_bayes", naive_bayes$results))
```

model	usekernel	laplace	adjust	F1	F1SD
Naive_bayes	FALSE	0	1	0.9049022	0.0666567
Naive_bayes	TRUE	0	1	0.8918421	0.0659453

KNN

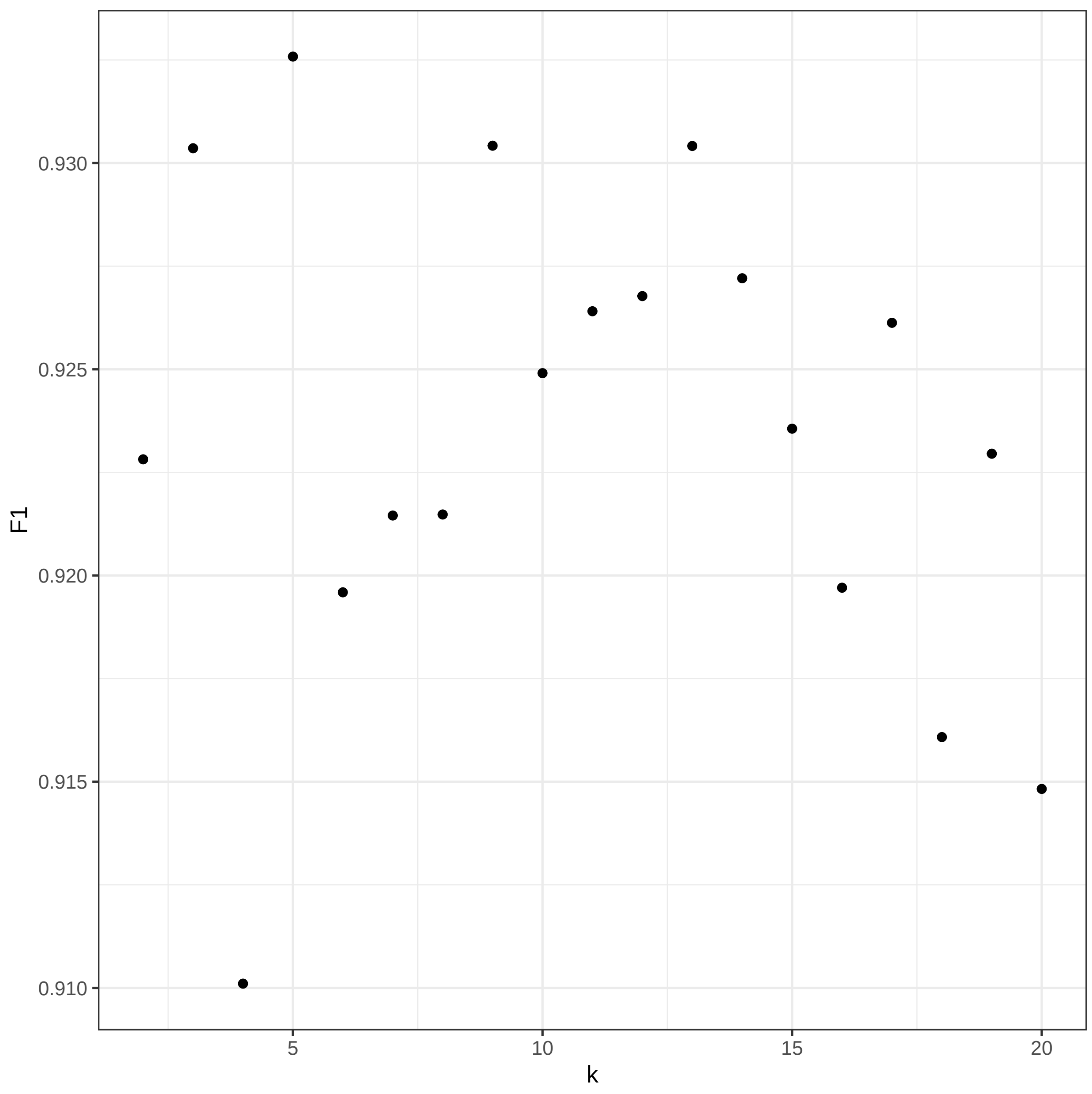
The k-nearest-neighbors model was fit. As a first approach, the grid used to tune the value of k was `seq(1, 51, by=2)`. The results are shown in the plot. From it, it is possible to see that the best values of k are between 2 and 20. Because of this, another grid was used to tune.

```r
knn <- train(y~., data = train,
             method = "knn",
             metric = "F1",
             tuneGrid = data.frame(k = seq(1, 51, by=2)),
             trControl = train.control)
ggplot(knn$results, aes(x = k, y = F1))+
  geom_point()+theme_bw()
```



After changing the grid, the best value of k is 5. The results are shown in the plot.

```r
knn <- train(y~., data = train,
             method = "knn",
             metric = "F1",
             tuneGrid = data.frame(k = 2:20),
             trControl = train.control)
ggplot(knn$results, aes(x = k, y = F1))+
  geom_point()+theme_bw()
```



GamLoess

The Generalized Additive Model using LOESS was fitted to the train data. The parameter values used are shown in the table.

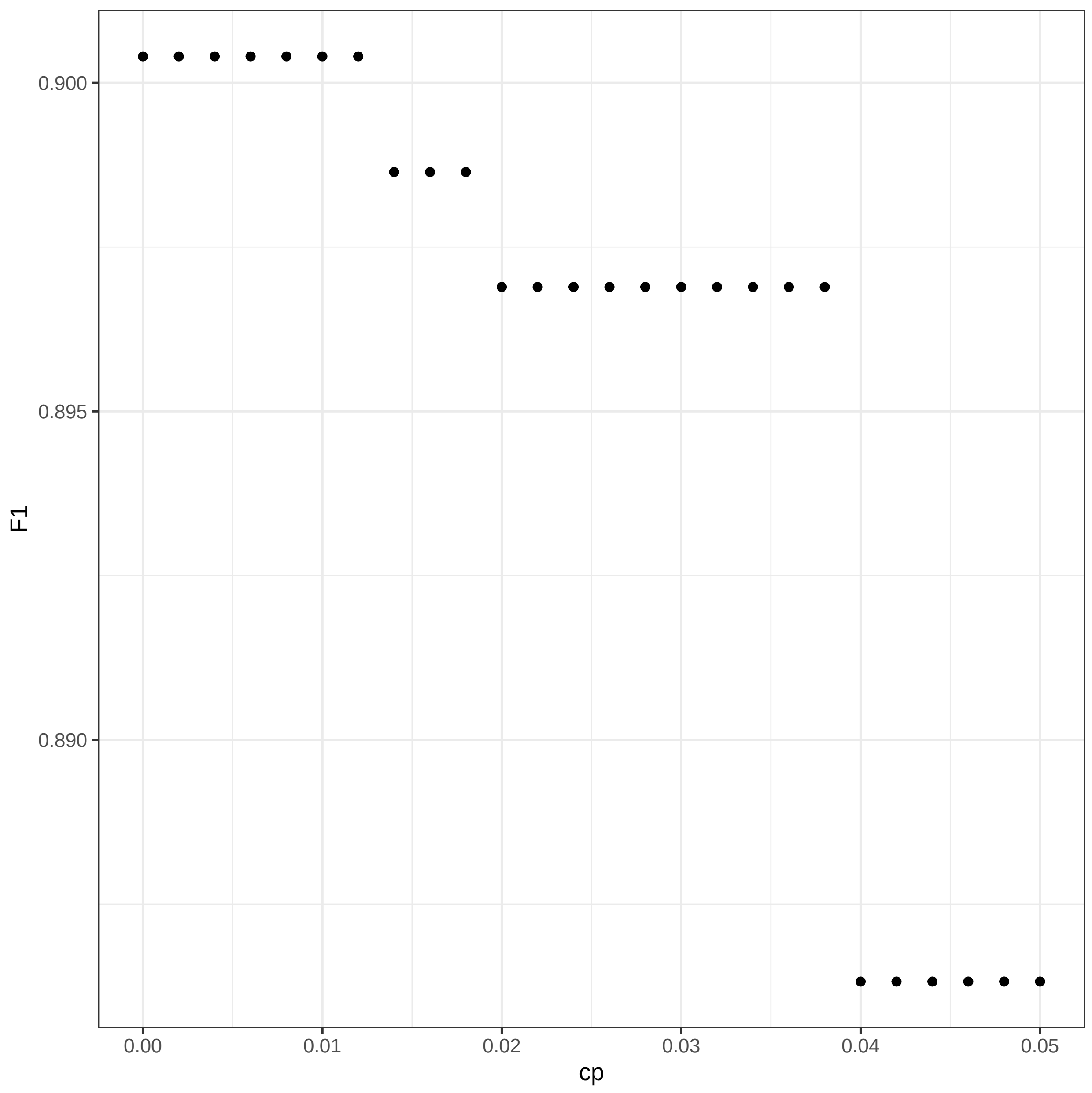
```r
gamLoess <- train(y~., data = train,
                  method = "gamLoess",
                  metric = "F1",
                  trControl = train.control)
kable(gamLoess$results)
```

span	degree	F1	F1SD
0.5	1	0.9409057	0.024787

Classification tree

A classification tree was fitted, with the complexity parameter varying in `seq(0, 0.05, 0.002)`. After the validation, the best tune is reached with $cp = 0.012$.

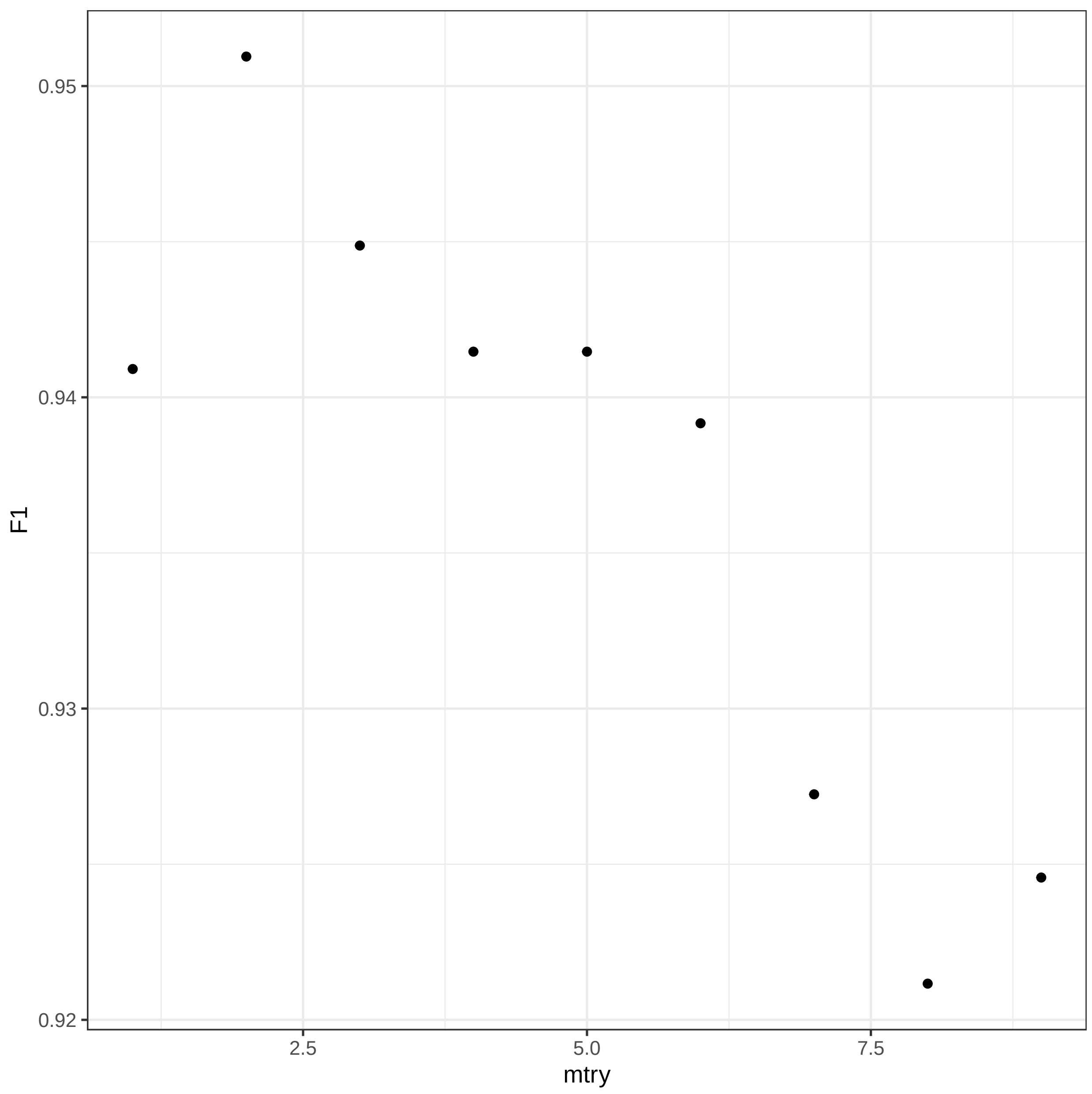
```r
rpart <- train(y~., data = train,
               method = "rpart",
               metric = "F1",
               tuneGrid = data.frame(cp = seq(0, 0.05, 0.002)),
               trControl = train.control)
ggplot(rpart$results, aes(x=cp, y=F1))+
  geom_point()+
  theme_bw()
```



Random forest

Then, a random forest was fitted, with the number of predictors varying from 1 to 9. The results are shown below, with the best tune being $mtry = 2$.

```
rf <- train(y~., data = train,
            method = "rf",
            metric = "F1",
            tuneGrid = data.frame(mtry = 1:9),
            trControl = train.control)
ggplot(rf$results, aes(x=mtry, y=F1))+
  geom_point()+
  theme_bw()
```



Gradient Boosting

The Stochastic Gradient Boosting was fitted to the data, firstly using a grid with the number of trees in `c(100,500,1000)`, the maximum number of nodes in each tree equal 1 or 2, shrinkage in `c(0.01,0.05,0.1)` and minimal terminal node size equals 10 or 20. From the results, it can be seen that *ntrees* = 1000 (maximum) is the best tune. Thus, another grid was built.

```
output <- capture.output(gbm <- train(y~., data = train,
             method = "gbm",
             metric = "F1",
             tuneGrid = data.frame(expand_grid(
               n.trees = c(100, 500, 1000),
               interaction.depth = 1:3,
               shrinkage = c(0.01,0.05,0.1,0.15),
               n.minobsinnode = c(10,20,30)
             )),
             trControl = train.control))
kable(gbm$results %>% arrange(-F1) %>% head())
```

shrinkage	interaction.depth	n.minobsinnode	n.trees	F1	F1SD
0.05	2	10	1000	0.9697938	0.0325620
0.10	3	20	1000	0.9697938	0.0325620
0.15	2	10	1000	0.9696401	0.0328656
0.05	3	10	1000	0.9696401	0.0328656
0.10	1	20	500	0.9671242	0.0363855
0.10	2	20	1000	0.9671047	0.0361054

Based on this new grid, one can see that the best tune values were found: $shrinkage = 0.1$, $interaction.depth = 3$, $n.minobsinnode = 20$ and $n.trees = 1000$ or $interaction.depth = 2$, $n.minobsinnode = 10$ and $n.trees = 2000$, meaning that these values are very similar, but better than the first ones used.

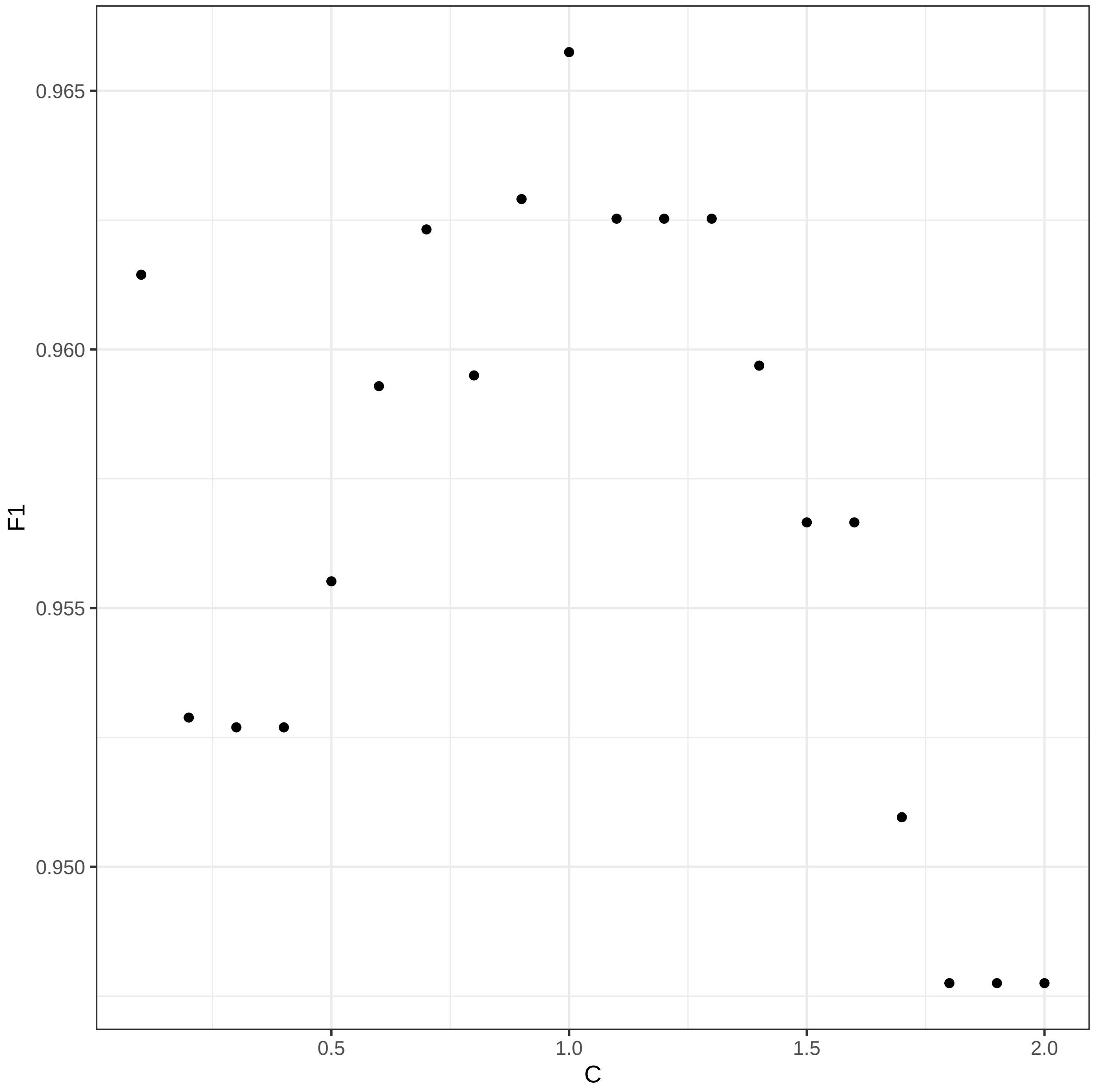
```
output <- capture.output(gbm <- train(y~., data = train,
              method = "gbm",
              metric = "F1",
              tuneGrid = data.frame(expand_grid(
                n.trees = seq(1000,2500, by = 500),
                interaction.depth = 2:4,
                shrinkage = seq(0.05, 0.15, by = 0.05),
                n.minobsinnode = c(10, 20)
              )),
              trControl = train.control))
kable(gbm$results %>% arrange(-F1) %>% head())
```

shrinkage	interaction.depth	n.minobsinnode	n.trees	F1	F1SD
0.10	3	20	1000	0.9641386	0.0333443
0.10	2	10	2000	0.9641386	0.0333443
0.15	2	10	2000	0.9639756	0.0308124
0.10	3	20	1500	0.9639603	0.0302853
0.10	3	20	2000	0.9639603	0.0302853
0.10	3	10	2500	0.9631974	0.0244947

SVM

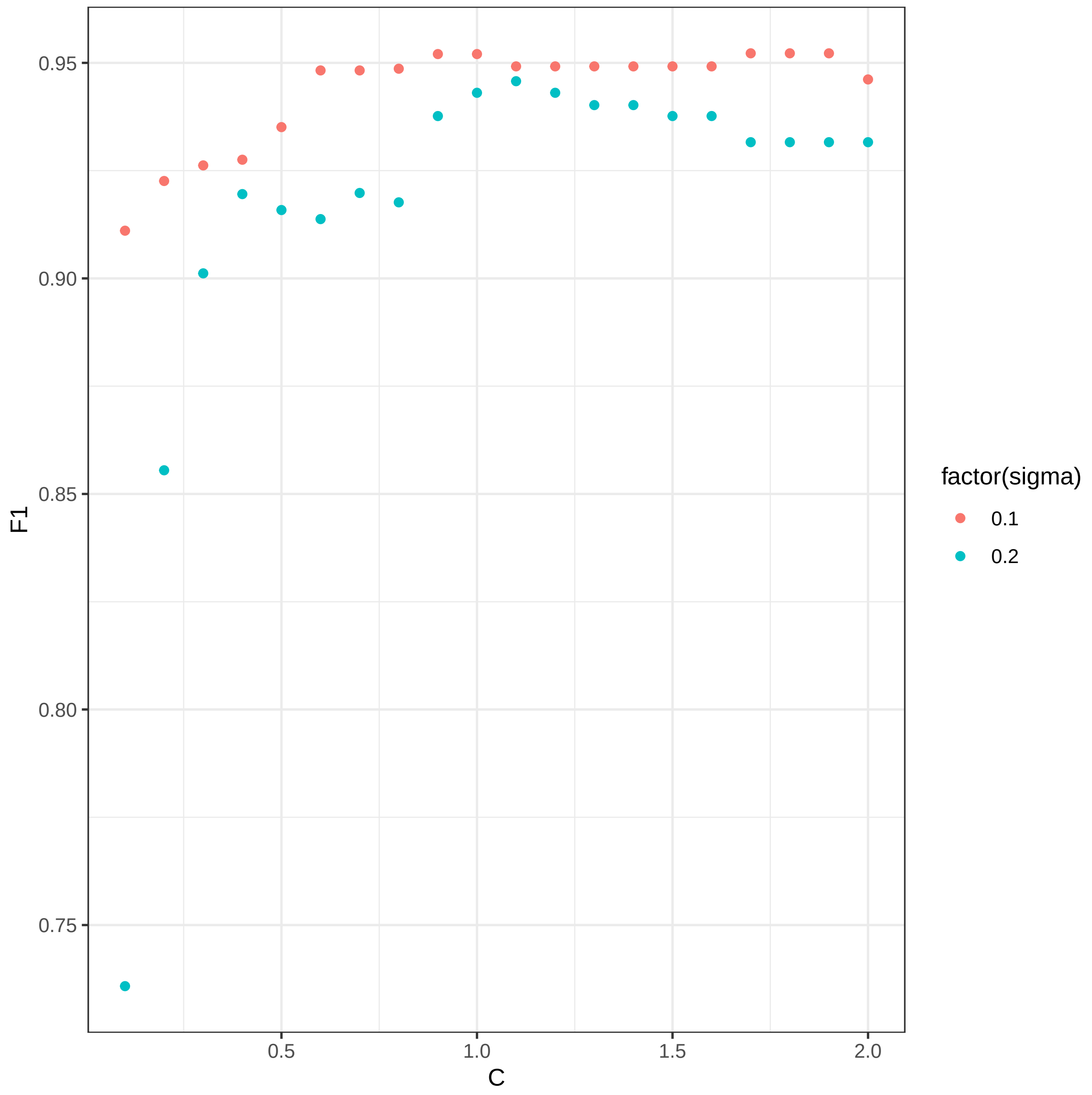
Then, Support Vector Machines with different kernels were fitted. Firstly, was used a Linear Kernel, with the best value of the cost being 1.

```
svmLinear <- train(y~., data = train,
                   method = "svmLinear",
                   metric = "F1",
                   tuneGrid = expand.grid(C = seq(0.1, 2, by = 0.1)),
                   trControl = train.control)
ggplot(svmLinear$results, aes(x=C, y = F1))+
  geom_point()+
  theme_bw()
```



Then, the Radial Basis Function Kernel was used. The best result was achieved with $sigma = 0.1$ and $cost = 1.7$.

```
svmRadial <- train(y~., data = train,
                   method = "svmRadial",
                   metric = "F1",
                   tuneGrid = expand.grid(
                     sigma = c(0.1, 0.2),
                     C = seq(0.1, 2, by = 0.1)),
                   trControl = train.control)
ggplot(svmRadial$results, aes(x=C, y = F1, color = factor(sigma)))+
  geom_point()+
  theme_bw()
```



Finally, a Polynomial Kernel was used to fit the model. The best tune is with *degree* $= 1$, *scale* $= 0.1$ and *cost* $= 0.9$.

```
svmPoly <- train(y~., data = train,
                   method = "svmPoly",
                   metric = "F1",
                   tuneGrid = expand.grid(
                     degree = 1:2,
                     scale = c(0.1,0.01, 0.001),
                     C = seq(0.1, 2, by = 0.1)),
                   trControl = train.control)
kable(svmPoly$results %>% arrange(-F1) %>% head(5))
```

degree	scale	C	F1	F1SD
1	0.1	0.9	0.9658009	0.0444917
1	0.1	0.4	0.9656166	0.0421306
2	0.1	0.9	0.9632281	0.0353416
2	0.1	1.0	0.9632281	0.0353416
2	0.1	1.6	0.9632281	0.0353416

Ensembles

After adjusting all the models, some ensembles were considered: one with all models, only GLMs, all but GLMs (caret models) and the 3, 5 and 9 best caret models.

```
models <- c("cloglog", "logit", "probit",
            "cloglog2", "logit2", "probit2",
            "gamLoess", "gbm", "knn","lda",
            "naive_bayes", "qda", "rf", "rpart",
            "svmLinear", "svmPoly", "svmRadial")
models_caret <- models[7:length(models)]
F1s <- sapply(models_caret, function(model) (eval(parse(text = model))$results %>% arrange(-F1))[1,"F1"
which.max(F1s)
```

```
## svmPoly
##      10
```

```
all <- models
all_caret <- models_caret
all_glm <- c("logit", "probit", "cloglog",
             "cloglog2", "logit2", "probit2")
top_3 <- names(F1s[order(F1s, decreasing = TRUE)[1:3]])
top_5 <- names(F1s[order(F1s, decreasing = TRUE)[1:5]])
top_9 <- names(F1s[order(F1s, decreasing = TRUE)[1:9]])
```

Evaluating in the test set

After training all the models and tuning their parameters in the training set,

```
preds <- sapply(models, function(mod){
  if(mod %in% models_caret){
  predict(eval(parse(text = mod)), newdata = test)}
  else ifelse(predict(eval(parse(text = mod)), newdata = test,
               type = "response")>0.5, 2,1)
})

cms <- apply(preds, 2, function(pred){
  confusionMatrix(factor(pred,
                         levels = c(1,2),
                         labels = c("M", "B")), test$y)
  })
names(cms) <- models
metrics <- sapply(cms, function(cm){
  c(cm$overall["Accuracy"],
    cm$byClass["Sensitivity"],
    cm$byClass["Specificity"],
    cm$byClass["Balanced Accuracy"],
    cm$byClass["F1"])
})
t(metrics)
```

```
##              Accuracy Sensitivity Specificity Balanced Accuracy        F1
## cloglog     0.9217391   0.9302326   0.9166667         0.9234496 0.8988764
## logit       0.9217391   0.9302326   0.9166667         0.9234496 0.8988764
## probit      0.9217391   0.9302326   0.9166667         0.9234496 0.8988764
## cloglog2    0.9130435   0.9069767   0.9166667         0.9118217 0.8863636
## logit2      0.9304348   0.9069767   0.9444444         0.9257106 0.9069767
## probit2     0.9217391   0.9069767   0.9305556         0.9187661 0.8965517
## gamLoess    0.9043478   0.9069767   0.9027778         0.9048773 0.8764045
## gbm         0.9565217   0.9302326   0.9722222         0.9512274 0.9411765
## knn         0.9391304   0.8372093   1.0000000         0.9186047 0.9113924
## lda         0.9478261   0.8604651   1.0000000         0.9302326 0.9250000
## naive_bayes 0.9478261   0.9069767   0.9722222         0.9395995 0.9285714
## qda         0.9304348   0.9302326   0.9305556         0.9303941 0.9090909
## rf          0.9130435   0.8837209   0.9305556         0.9071382 0.8837209
## rpart       0.9217391   0.9534884   0.9027778         0.9281331 0.9010989
## svmLinear   0.9478261   0.9302326   0.9583333         0.9442829 0.9302326
## svmPoly     0.9565217   0.9302326   0.9722222         0.9512274 0.9411765
## svmRadial   0.9391304   0.9069767   0.9583333         0.9326550 0.9176471
```

```
apply(t(metrics), 2, function(x)which.max(x))
```

```
##          Accuracy       Sensitivity       Specificity Balanced Accuracy
##                 8                14                 9                 8
##                F1
##                 8
```

```
t(metrics)[which.max(t(metrics)[,5]),]
```

```
##          Accuracy       Sensitivity       Specificity Balanced Accuracy
##         0.9565217         0.9302326         0.9722222         0.9512274
##                F1
##         0.9411765
```

```
cms[[which.max(t(metrics)[,5])]]
```

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

```
##           Reference
## Prediction  M  B
##          M 40  2
##          B  3 70
##
##                Accuracy : 0.9565
##                  95% CI : (0.9015, 0.9857)
##     No Information Rate : 0.6261
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9067
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9302
##             Specificity : 0.9722
##          Pos Pred Value : 0.9524
##          Neg Pred Value : 0.9589
##              Prevalence : 0.3739
##          Detection Rate : 0.3478
##    Detection Prevalence : 0.3652
##       Balanced Accuracy : 0.9512
##
##        'Positive' Class : M
##
```

```
# Ensembles
ensembles <- c('all', 'all_caret', 'all_glm',
               'top_3', 'top_5', 'top_9')
preds_e <- sapply(ensembles, function(x){
  mods <- eval(parse(text = x))
  ifelse(rowMeans(preds[, mods]==1)>0.5, 1,2)
})
cms_e <- apply(preds_e, 2, function(pred){
  confusionMatrix(factor(pred,
                         levels = c(1,2),
                         labels = c("M", "B")), test$y)
  })
names(cms_e) <- ensembles
metrics_e <- sapply(cms_e, function(cm){
  c(cm$overall["Accuracy"],
    cm$byClass["Sensitivity"],
    cm$byClass["Specificity"],
    cm$byClass["Balanced Accuracy"],
    cm$byClass["F1"])
})
t(metrics_e)
```

```
##            Accuracy Sensitivity Specificity Balanced Accuracy        F1
## all       0.9478261   0.9302326   0.9583333         0.9442829 0.9302326
## all_caret 0.9478261   0.9069767   0.9722222         0.9395995 0.9285714
## all_glm   0.9217391   0.9069767   0.9305556         0.9187661 0.8965517
## top_3     0.9652174   0.9534884   0.9722222         0.9628553 0.9534884
## top_5     0.9391304   0.9069767   0.9583333         0.9326550 0.9176471
## top_9     0.9391304   0.8837209   0.9722222         0.9279716 0.9156627
```

```
apply(t(metrics_e), 2, function(x)which.max(x))
```

```
##          Accuracy       Sensitivity       Specificity Balanced Accuracy
##                 4                 4                 2                 4
##                F1
##                 4
```

```
cms_e[[which.max(t(metrics_e)[,5])]]
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  M  B
##          M 41  2
##          B  2 70
##
##                Accuracy : 0.9652
##                  95% CI : (0.9133, 0.9904)
##     No Information Rate : 0.6261
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9257
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9535
##             Specificity : 0.9722
##          Pos Pred Value : 0.9535
##          Neg Pred Value : 0.9722
##              Prevalence : 0.3739
##          Detection Rate : 0.3565
##    Detection Prevalence : 0.3739
##       Balanced Accuracy : 0.9629
##
##        'Positive' Class : M
##
```

```
data.frame(rbind(t(metrics), t(metrics_e))) %>%
  arrange(-F1)
```

```
##              Accuracy Sensitivity Specificity Balanced.Accuracy        F1
## top_3       0.9652174   0.9534884   0.9722222         0.9628553 0.9534884
## gbm         0.9565217   0.9302326   0.9722222         0.9512274 0.9411765
## svmPoly     0.9565217   0.9302326   0.9722222         0.9512274 0.9411765
## svmLinear   0.9478261   0.9302326   0.9583333         0.9442829 0.9302326
## all         0.9478261   0.9302326   0.9583333         0.9442829 0.9302326
## naive_bayes 0.9478261   0.9069767   0.9722222         0.9395995 0.9285714
## all_caret   0.9478261   0.9069767   0.9722222         0.9395995 0.9285714
## lda         0.9478261   0.8604651   1.0000000         0.9302326 0.9250000
## svmRadial   0.9391304   0.9069767   0.9583333         0.9326550 0.9176471
## top_5       0.9391304   0.9069767   0.9583333         0.9326550 0.9176471
## top_9       0.9391304   0.8837209   0.9722222         0.9279716 0.9156627
## knn         0.9391304   0.8372093   1.0000000         0.9186047 0.9113924
## qda         0.9304348   0.9302326   0.9305556         0.9303941 0.9090909
## logit2      0.9304348   0.9069767   0.9444444         0.9257106 0.9069767
## rpart       0.9217391   0.9534884   0.9027778         0.9281331 0.9010989
## cloglog     0.9217391   0.9302326   0.9166667         0.9234496 0.8988764
## logit       0.9217391   0.9302326   0.9166667         0.9234496 0.8988764
## probit      0.9217391   0.9302326   0.9166667         0.9234496 0.8988764
## probit2     0.9217391   0.9069767   0.9305556         0.9187661 0.8965517
## all_glm     0.9217391   0.9069767   0.9305556         0.9187661 0.8965517
## cloglog2    0.9130435   0.9069767   0.9166667         0.9118217 0.8863636
## rf          0.9130435   0.8837209   0.9305556         0.9071382 0.8837209
## gamLoess    0.9043478   0.9069767   0.9027778         0.9048773 0.8764045
```

```
a <- rbind(t(metrics)[which.max(t(metrics)[,5]),],
      t(metrics_e)[which.max(t(metrics_e)[,5]),])
rownames(a) <- c(names(which.max(t(metrics)[,5])),
                 names(which.max(t(metrics_e)[,5])))
a %>% kable()
```

	Accuracy	Sensitivity	Specificity	Balanced Accuracy	F1
gbm	0.9565217	0.9302326	0.9722222	0.9512274	0.9411765
top_3	0.9652174	0.9534884	0.9722222	0.9628553	0.9534884