Exercise C for Predictive Modeling

Pablo Vidal Fernández 100483812
José Ignacio Díez Ruiz 100487766
Carlos Roldán Piñero 100484904

2023-01-08

Introduction

In the search of a dataset of true interest for ourselves, we were sadly met with failure. After exhaustive indagations in public repositories, we were not able to find a suitable one. We did, nonetheless, stumbled upon intriguing datasets, which, unfortunately, were time series.

As such, and although it was against our primary desire, we ended up using the mysterious.RData. This, presented some challenges, which will be explored on the next section, which made it certainly more appealing than the majority of exploited online datasets.

Let us detail briefly the contents of the selected data. It is comprised of one binary prediction variable `y` and 903 predictors, seggregated by numericals, 893 of them, and categoricals, the remaining 10. It has 224 observations, which already tells us that we will need to consider dimensional reduction techniques. It has some NaNs present in it, a total of 81 values spanning 26 rows. On the next section we will detail our analysis process and buildup of the prediction model.

Model Building

Preprocessing

We first start by analysing the distributions of NaNs on the various variables:

```
cols_with_na <- sort(sapply(df, function(x) sum(is.na(x))), decreasing = TRUE)
cols_with_na[cols_with_na > 0]
```

```
##   fac.8   fac.6   fac.7 num.148
##      23      20      20      18
```

We can see that the 81 missing values are within 4 variables. Three of them are categorical. Before deciding whether we should impute or not, we want to be sure if this variables are important for the prediction.

For the numerical variable, a simple t-test is enough to determine that there is a significant difference between the groups. Hence, we will keep it and impute it using the median.

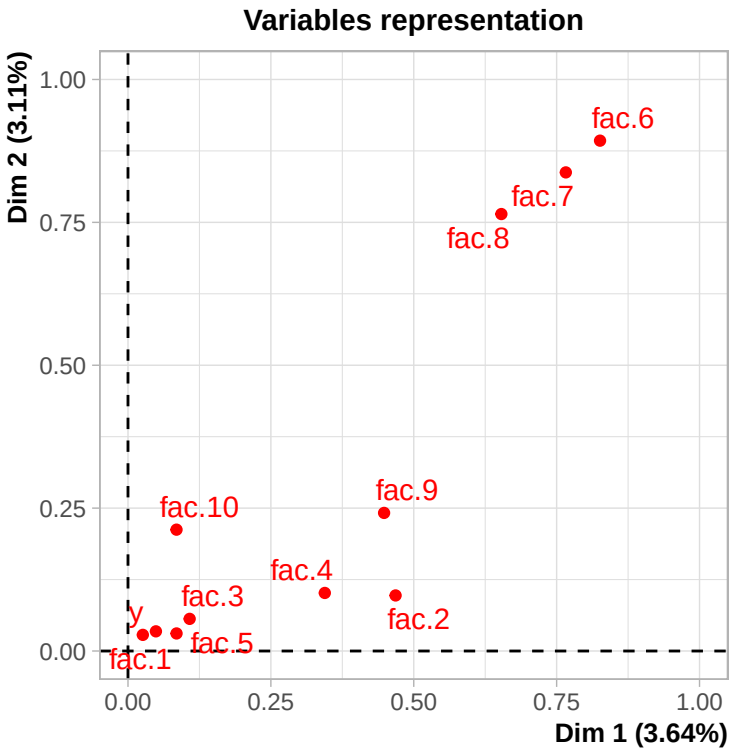
```
t.test(num.148 ~ y, data = df)
```

```
##
##  Welch Two Sample t-test
##
## data:  num.148 by y
## t = -2.5568, df = 198.47, p-value = 0.01131
## alternative hypothesis: true difference in means between group A and group B is not equal to 0
## 95 percent confidence interval:
##  -16.826000  -2.172864
```

```
## sample estimates:
## mean in group A mean in group B
##        87.71875        97.21818
```

The categorical variables we are going to look at them using a MCA. We can see that factors 6, 7 and 8 are not close to the other ones. Considering this, the fact that in the univariate fit for each of this factors no category has a significative coefficient, that factors 6 and 7 have a lot of categories, and that we still have other 900 variables we have decided to drop them. We will later return to the MCA. [1]

```
full_mca <- MCA(X = df[categorical_idx], graph = FALSE)
plot.MCA(full_mca, choix = "var", graph.type = "ggplot")
```



For the following part we will need two auxiliary functions. With the first, we will check whether there is (up to float accuracy), perfect multicollinearity between some of our variables. The second one checks if some column may be considered as constant (taking in account some variability for the numerical variables).

```
find_mulcol <- function(x) {
  # Handle factors
  if (all(sapply(x, class) == "factor")) {
    x <- dummy_cols(x, remove_first_dummy = TRUE,
            remove_selected_columns = TRUE)
  }

  # Fit each column with the rest
  problematicas <- c()
  for (i in seq_len(ncol(x))) {
    f <- lm(x[, i] ~ ., data = x[, -i])
    if (summary(f)$r.squared > 1 - 1e-10) {
      problematicas <- append(problematicas, colnames(x)[i])
```

[1] During the whole document, in the codeblocks we will refer to numerical variables as `numerical_idx`, factorials as `factorials_idx` and factorials together with the prediction variable as `categorical_idx`.

```
    }
  }

  return(problematicas)
}

check_if_constant_column <- function(x) {
  num <- x[1]
  c1  <- sum(x == num) > length(x) / 2
  if (class(x) == "numeric") {
    c2 <- all(near(x, num, tol = 0.001 * mean(x)))
    return(c1 & c2)
  } else {
    return(all(x == num))
  }
}
```

We now use them to remove problematic predictors.

```
# Problematic categoricals
find_mulcol(df[factorial_idx])
```

```
## [1] "fac.9_c1"  "fac.9_d1"  "fac.9_f1"  "fac.10_b1" "fac.10_c1"
```

```
df <- drop_columns(df, c("fac.9"))
find_mulcol(df[factorial_idx])
```

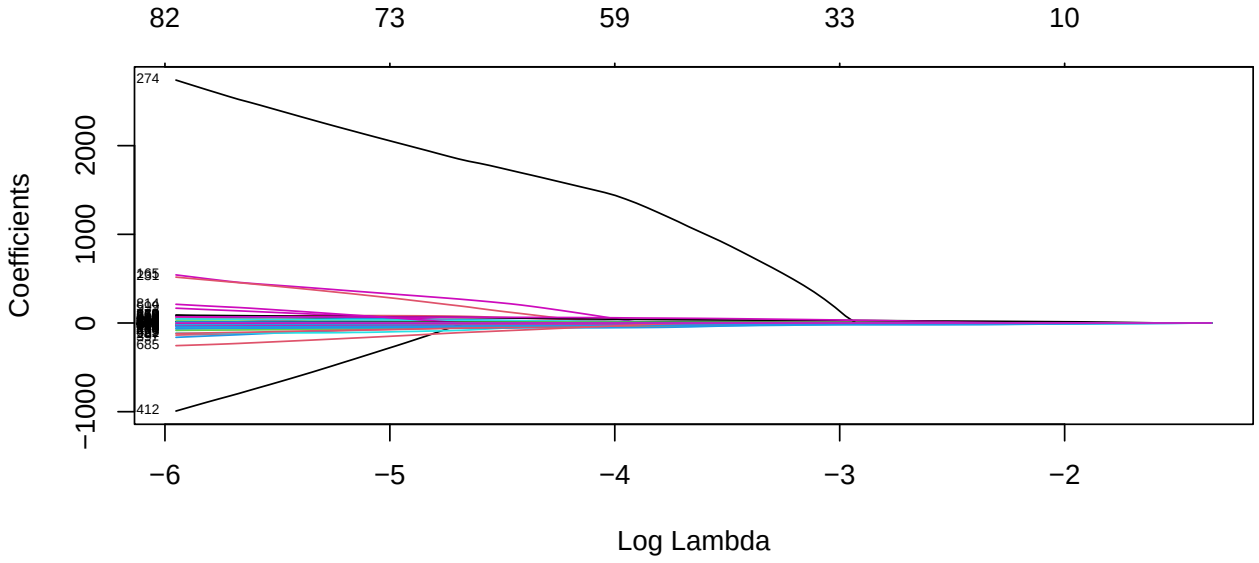
```
## NULL
```

```
# Drop constant columns
constant_columns_idx <- names(which(sapply(df, check_if_constant_column)))
df <- drop_columns(df, constant_columns_idx)
```

Dimensionality reduction

We had several ideas on how to approach dimensionality reduction. Mainly, we discussed whether to use PCA or Lasso with the numerical variables. Finally, we settled on Lasso as not only did it prove to be more numerically robust on the particularities of the dataset, but it also carries information of the prediction variable for the variable selection.

```
y     <- df$y
y_enc <- dplyr::recode(y, A = 0, B = 1)
x_num <- df[numerical_idx]
x_scl <- scale(x_num)
lasso <- glmnet(x_num, y_enc, family = "binomial", alpha = 1)

plot(lasso, xvar = "lambda", label = TRUE)
```



There seem to be some important variables as they converge slowly to zero when incrementing the value of λ. However, we do not know which is the best value for λ. We will use cross-validation and the rule of $\hat{\lambda}_{k-1SE}$. The idea behind it is visually seen in Figure 4.5, where it is evidenced that $\hat{\lambda}_{k-1SE}$ consistently predicts the true model for large enough datasets.

Due to cross validation in `glmnet` being random, we perform 10 7-fold [2] CV Lasso regressions and use the average obtained $\hat{\lambda}_{k-1SE}$.

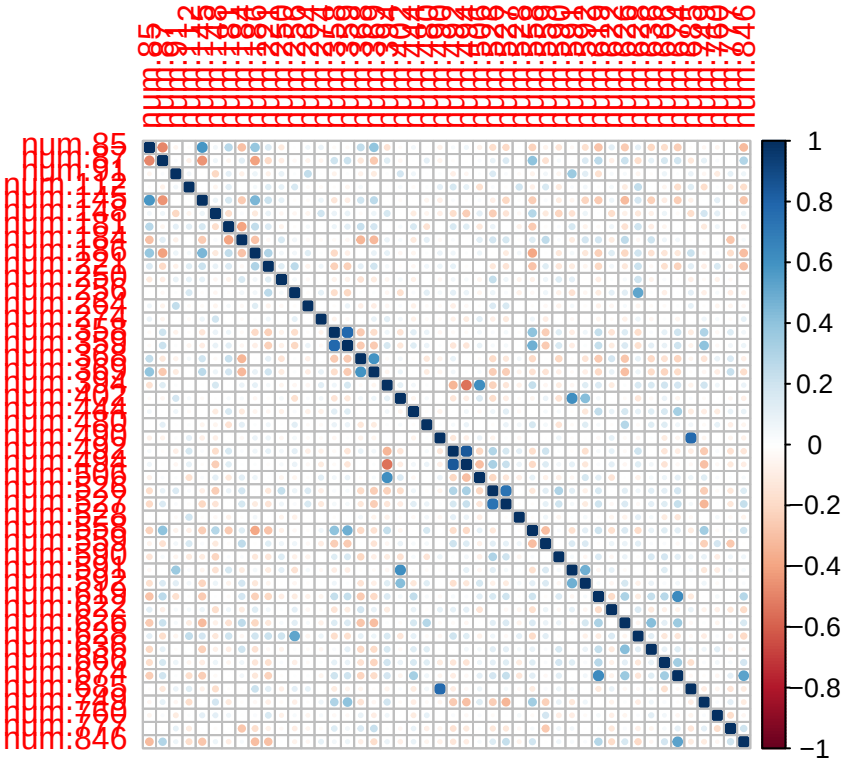
```
optimal_lambda <- 0.0
for (i in 1:10) {
  cv_lasso        <- cv.glmnet(x_scl, y_enc, family = "binomial",
                               alpha = 1, nfolds = 7)
  optimal_lambda  <- optimal_lambda + cv_lasso$lambda.1se
}
optimal_lambda    <- optimal_lambda / 10

mod_lasso <- glmnet(x_scl, y_enc, family = "binomial",
                    alpha = 1, lambda = optimal_lambda)
pred_nums <- predict(mod_lasso, type = "coefficients",
                             s = optimal_lambda)[-1, ] != 0

x_lasso   <- as.data.frame(x_scl[, pred_nums])
```

The best λ is 0.0246. It has 47 non-null coefficients.

Note that we have only reduced the number of variables. We still need to deal with multicollinearity between the selected predictors. We will do so using PCA, but first we filter out perfect multicollinearity as they will bring numerical inestability to the PCA. We illustrate with a corrplot that there is still huge correlations

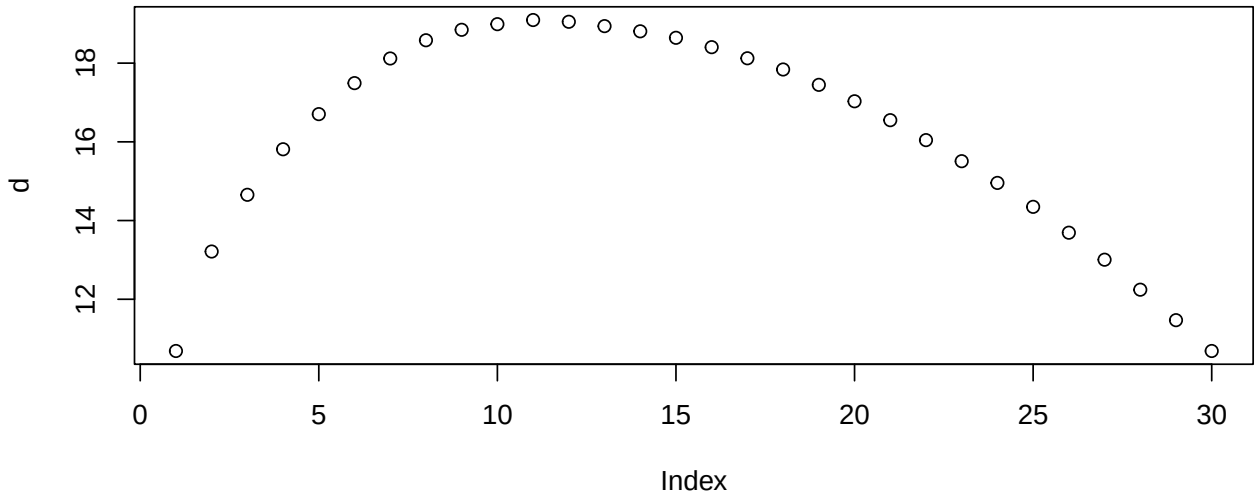
```
problematic <- find_mulcol(x_lasso)
while (length(problematic) > 0) {
  x_lasso <- subset(x_lasso, select = !(names(x_lasso) %in% problematic[1]))
  problematic <- find_mulcol(x_lasso)
}

corrplot(cor(x_lasso))
```

[2] The number 7 is set because it divides the number of observations.



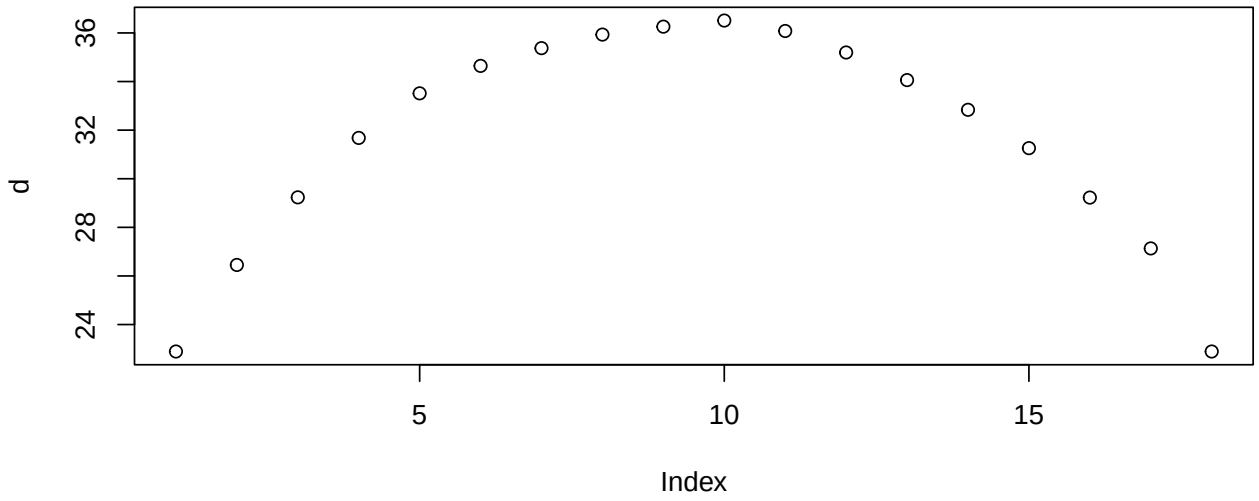
We can now safely proceed to the PCA.

```
pca <- princomp(x_lasso)

v <- pca$sdev^2
v <- v[v / sum(v) > 0.01]
c <- cumsum(v)
m <- (tail(c, 1) - c[1]) / (length(c) - 1)
d <- c - m * c(0:(length(c) - 1)) + c[1]

plot(d)
```




```
n_pcs     <- which.max(d)
new_x_num <- as.data.frame(pca$scores[, 1:n_pcs])
```

```r
mca <- MCA(X = df[factorial_idx], graph = FALSE, ncp = 20)
c <- mca$eig[, 3]
m <- (tail(c, 1) - c[1]) / (length(c) - 1)
d <- c - m * c(0:(length(c) - 1)) + c[1]

plot(d)
```




```r
n_mcs      <- which.max(d)
new_x_fac <- as.data.frame(mca$ind$coord[, 1:n_mcs])
```

```r
final_df  <- cbind(y_enc, new_x_num, new_x_fac)

mod_zero  <- glm(y_enc ~ 1, family = binomial, data = final_df)
mod_all   <- glm(y_enc ~ ., family = binomial, data = final_df)

both <- MASS::stepAIC(mod_zero, scope = list(lower = mod_zero, upper = mod_all),
                      direction = "both", trace = 0, k = log(nrow(final_df)))
```

```r
nfolds   <- 7

cross_val <- function(model, df, nfolds) {
  acc       <- 0.0
  fold_size <- floor(nrow(df) / nfolds)
  for (i in 1:nfolds) {
    sr  <- (i - 1) * fold_size
    er  <- i * fold_size
    f   <- glm(model$formula, family = binomial, data = df[-(sr:er), ])
    p   <- predict(f, newdata = df[sr:er, -1], type = "response")
    acc <- acc + mean((p > 0.5) == df[sr:er, 1])
  }
  return(acc / nfolds)
}

acc <- cross_val(both, final_df, nfolds)
print(acc)
```

```
## [1] 0.9433171
```