Excercise: binary classification

Fabio Morea

2022-02-10

Contents

1. Introduction

Description of the problem and a motivation for addressing it, why relevant # 2. Formal statement # 3. Indexes for measuring the solutions # 4. Data exploration and feature engineering (most relevant features, why) # 5. Proposed solution # 6. Experimental procedure for measuring effectiveness (k-fold cross validation procedure) is an experimental procedure # 7. Discussion

Business case

Business case for this excercise Area Science Park is interested in predicting a label "isHealty". used for monitoring the performance of companis within homogeneous groups. Each company is associated with a performance level (isHealthy = True/False) that is used for further activities (e.g. "Top"healthy" companies are featured in the newsletter, others are offered services to boost innovation and performance). The classification algorithm is based on financial ratings, financial indicators and employment indicators. The goal is to predict the label *without* using financial indicators, that is expansive. (purchased at 5,00€ per company).

Objective: to develop a new method, leading to the same classification, replacing financial ratings with balance sheet data (which can be purchased at a significantly lower cost, € 0.75 per company). The expected result is a model (learning and prediction modules) and a detailed report describilng the model performance (error rate, robustness to unbalanced data, . . .)

Constraints: training and classification will be performed on a laptop, twice a year. No specific constraints on time or computation effort (even if it takes hours, it's ok).

Workflow - insert picture here - Classification tree

Understanding the data

TODO insert figure here

Data is available from 4 sourecs: * cmp.csv * bsd.csv * rating.csv * empl_flow.csv [TODO]

and sould be pre-processed to obtain two vectors: X and y.

Feature selection

This section is dedicated to load and preprocess data

The relevant files are

- *cmp.csv* and *codes.csv*: . . . Each observation is . . . there are p attributes. We will use . . . . to filter companies that belong to a specific sector () and of a specific type.
- *bsd.csv.* Each observation is a summary of balance sheet data (bsd) of a company (identified by *cf*) for a given year. Column labels need some improvement to remove whitespaces and possibly short english names.
- *rating.csv*
- *empl-flow.csv* and *empl-stock.csv*: . . . .

```
pathTidyData = './../../_data/tidy/'
companies <-  read_csv( paste0(pathTidyData,"cmp.csv"),    show_col_types = FALSE )
bsd <-        read_csv( paste0(pathTidyData,"bsd.csv"),    show_col_types = FALSE )
rating <-     read_csv( paste0(pathTidyData,"rating.csv"), show_col_types = FALSE )
codes <-      read_csv( paste0(pathTidyData,"nace.csv"),   show_col_types = FALSE )
```

Selection of sample

Select NACE code (see _data/ino/ for a complete list of codes https://ec.europa.eu/eurostat/web/products-manuals-and-guidelines/-/ks-ra-07-015 codes are organized by: Division / Group / Class

We are interested in Division 22 22 Manufacture of rubber and plastic products 22.1 Manufacture of rubber products 22.11 Manufacture of rubber tyres and tubes; retreading and rebuilding of rubber tyres 2211 22.19 Manufacture of other rubber products 2219 22.2 Manufacture of plastics products 22.21 Manufacture of plastic plates, sheets, tubes and profiles 2220* 22.22 Manufacture of plastic packinggoods 2220* 22.23 Manufacture of builders' ware of plastic 2220* 22.29 Manufacture of other plastic products

And only companies that have a duty of disclosure of financial information. SOCIETA' DI CAPITALE|SU|SOCIETA' A RESPONSABILITA' LIMITATA CON UNICO SOCIO SOCIETA' DI CAPITALE|SR|SOCIETA' A RESPONSABILITA' LIMITATA SOCIETA' DI CAPITALE|SP|SOCIETA' PER AZIONI SOCIETA' DI CAPITALE|SD|SOCIETA' EUROPEA SOCIETA' DI CAPITALE|RS|SOCIETA' A RESPONSABILITA' LIMITATA SEMPLIFICATA SOCIETA' DI CAPITALE|RR|SOCIETA' A RESPONSABILITA' LIMITATA A CAPITALE RIDOTTO SOCIETA' DI CAPITALE|AU|SOCIETA' PER AZIONI CON SOCIO UNICO SOCIETA' DI CAPITALE|AA|SOCIETA' IN ACCOMANDITA PER AZIONI

```r
#select only some types of ng2 (natura giuridica)
selectedNg = c("SU", "SR", "SP", "SD", "RS", "RR", "AU", "AA")
#selectedNg = c( "SR")
companies <- companies %>% filter(ng2 %in% selectedNg)

#select only division 28
divs = c( 28)
selectedCf <- codes %>% filter(division %in% divs) %>% select(cf)
companies  <- companies %>% semi_join(selectedCf) #semi_join() return all rows from x with a match in y
```

```
## Joining, by = "cf"
```

```r
bsd        <- bsd       %>% semi_join(selectedCf)
```

```
## Joining, by = "cf"
```

```r
rating     <- rating    %>% semi_join(selectedCf)
```

```
## Joining, by = "cf"
```

```r
checkDuplicates <- companies %>% filter(duplicated(.[["cf"]]))#check duplicates (none expected)
```

Now we have a sample of {r length(companies)} companies. Duplicates are {r length(checkDuplicates)}.

Labels based on financial ratings

TODO: improve description: > The credit rating and commercial credit limit available within the s-peek application are evaluated through an innovative methodology called Multi Objective Rating Evaluation which is owned by modeFinance. This innovative methodology studies a corporation as a complex system and deepens the analysis on its different aspects: solvency, debt coverage, liquidity, cash conversion cycle, profitability, fixed asset coverage, compared with the sector which it belongs to and so on. With effect from July 10th, 2015, modeFinance Srl is registered as a credit rating agency in accordance with Regulation (EC) No 1060/2009 of the European Parliament ad of the Council of 16 September 2009 (the Credit Rating Agencies Regulation link). MORE Methodology is used by modeFinance also as part of the process of issuance of Credit Ratings in compliance with Regulation (EC) No 1060/2009 of the European Parliament ad of the Council of 16 September 2009 (the Credit Rating Agencies Regulation).

```
bsd <- bsd %>% filter(year == 2019)
rating <- rating %>%  filter(year == 2019)

companies <- companies %>%
        inner_join(bsd, by = "cf") %>%
        inner_join(rating, by = "cf")

checkDuplicates <- companies %>% filter(duplicated(.[["cf"]]))#check duplicates (none expected)
```

create X and y

```
# names can be used for debug purposes to check the identiy of each company
names <- companies %>% select(name,cf,idCompany)

X     <- companies %>% select(idCompany, is.sme, is.startup, is.fem, is.young, is.fore, yearsInBusiness

y     <- companies %>%
          mutate(isHealthy = (noi >0) & (rating010 >=8)&(yearsInBusiness>10))  %>%
          select(idCompany,isHealthy)
```

Create a label - at the current stage a binary label is enough to test out method

```
data <- X %>%
       inner_join(y)%>%
        select(-idCompany)%>%
        mutate(isHealthy = as.factor(isHealthy))
```

```
## Joining, by = "idCompany"
```

feature engineering

select most relevant features for prediction, based on domain knowledge

we start with a few atttributes, that are deemed more relevant included: - totAssess = totale attivo = total assets - noi = (ron) reddito operativo netto = (noi) net operating income - personnel = totale costi del personale = total personnel costs - debts = debiti esigibili entro l'esercizio successivo = debts due within the following financial year

not included:

- totEquity = totale patrimonio netto = total equity
- profLoss = uile/perdita esercizio ultimi = profit / loss for the last financial year
- accounts = crediti esigibili entro l'esercizio successivo = accounts receivables
- totIntang = totale immobilizzazioni immateriali = total intangible fixed assets
- prod = totale valore della produzione = total production value

- revenues = ricavi delle vendite = revenues from sales
- valCost = differenza tra valore e costi della produzione = difference between production value and production costs
- ammort = ammortamento immobilizzazione immateriali = amortisation
- valAdded = valore aggiunto = value added
- deprec = tot.aam.acc.svalutazioni = total amortisation, depreciation and write-downs

check for correlation:

Question: a general overview of the selected variables.
Design: pairs plot = matrix of scatterplots that lets you understand the pairwise relationship between different variables in a dataset

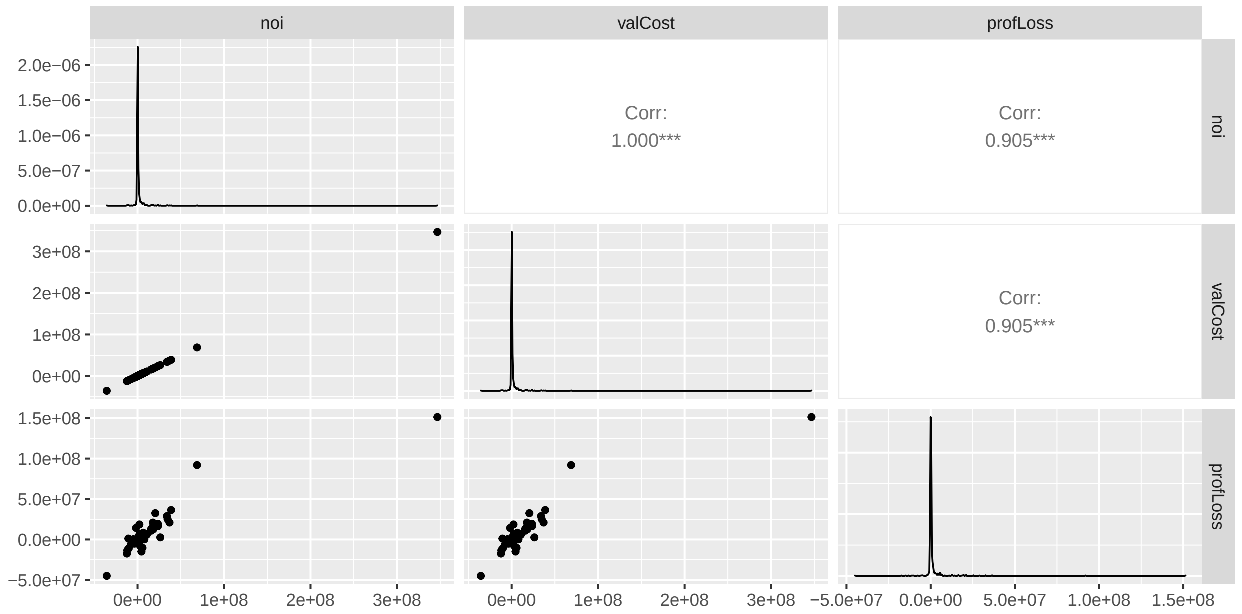
```
require(GGally)
```

```
## Caricamento del pacchetto richiesto: GGally
```

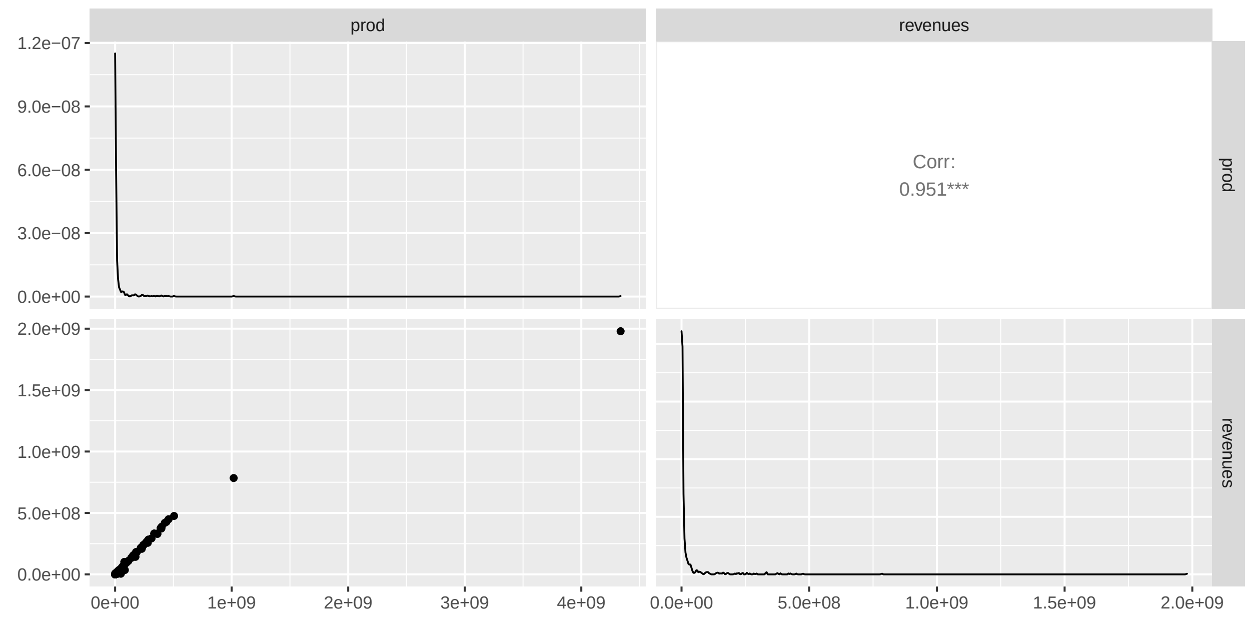
```
## Warning: il pacchetto 'GGally' è stato creato con R versione 4.1.2
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

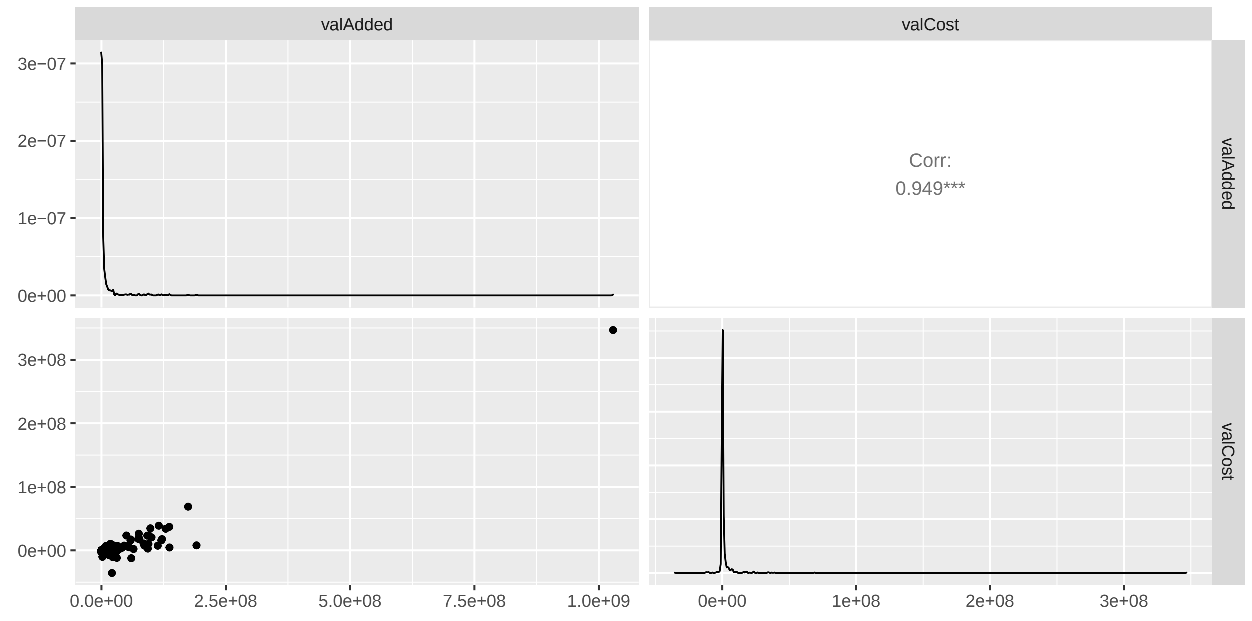
```
data %>%  select(noi, valCost, profLoss) %>% ggpairs()
```



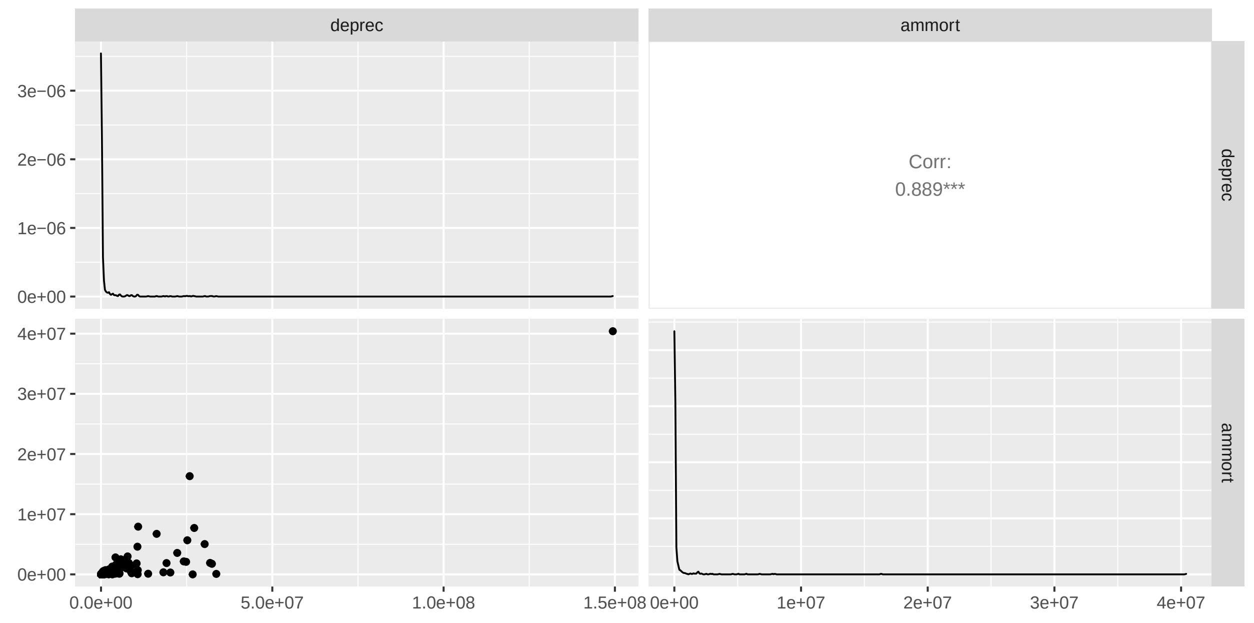
```
data %>%  select(prod, revenues) %>% ggpairs()
```



```
data %>%  select(valAdded, valCost) %>% ggpairs()
```

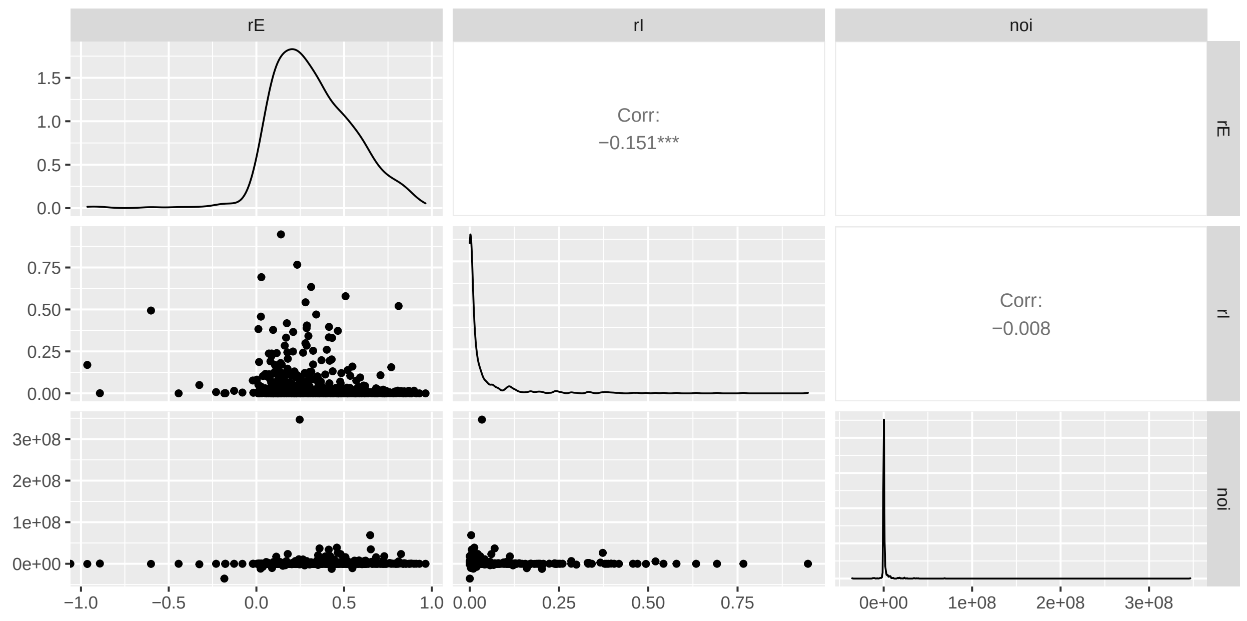


```
data %>%  select(deprec, ammort) %>% ggpairs()
```



```
data %>%  select(totAssets, totEquity, totIntang, noi) %>%
  mutate(rE = totEquity/totAssets) %>%
  mutate(rI = totIntang/totAssets) %>%
  select(rE, rI, noi) %>%
  ggpairs()
```

```
## Warning: Removed 1 rows containing non-finite values (stat_density).

## Warning in ggally_statistic(data = data, mapping = mapping, na.rm = na.rm, :
## Removing 1 row that contained a missing value

## Warning: Removed 1 rows containing missing values (geom_text).

## Warning: Removed 1 rows containing missing values (geom_point).

## Warning: Removed 1 rows containing non-finite values (stat_density).

## Warning in ggally_statistic(data = data, mapping = mapping, na.rm = na.rm, :
## Removing 1 row that contained a missing value

## Warning: Removed 1 rows containing missing values (geom_point).
```



we check for the scale of each variable

```
# gruop1 <- c(is.sme, is.startup, is.fem, is.young, is.fore)
# group2 <- c(yearsInBusiness)
# group3 <- c(totAssets, totIntang,accounts,totEquity,debts,prod,revenues,personnel,valCost, ammort, pr
#require(GGally)
#data %>%  select(is.sme:is.fore, isHealthy) %>% ggpairs()
```

```
#data %>%  select(yearsInBusiness, totAssets, noi, isHealthy) %>% ggpairs()
#data %>%  select(totEquity, totAssets, totIntang,noi, profLoss, valCost , isHealthy) %>% ggpairs()
```

```
# data <- data %>% select(totAssets,noi,personnel, debts,totIntang, isHealthy) %>%
#             rowwise() %>%
#             mutate(relNoi =       log10(noi)/log10(totAssets)) %>%
#             mutate(relPers =      log10(personnel)/log10(totAssets)) %>%
#             mutate(relDebts =     log10(debts)/log10(totAssets)) %>%
#             mutate(relIntang =    log10(totIntang)/log10(totAssets))%>%
#
#             select(-totAssets,-noi,-personnel,-debts, -totIntang)
#
# data %>%
#   pivot_longer(cols=!isHealthy) %>%
#   ggplot(aes(x=isHealthy, y=value, color = isHealthy)) + geom_boxplot() + facet_grid(.~name, scales="
```

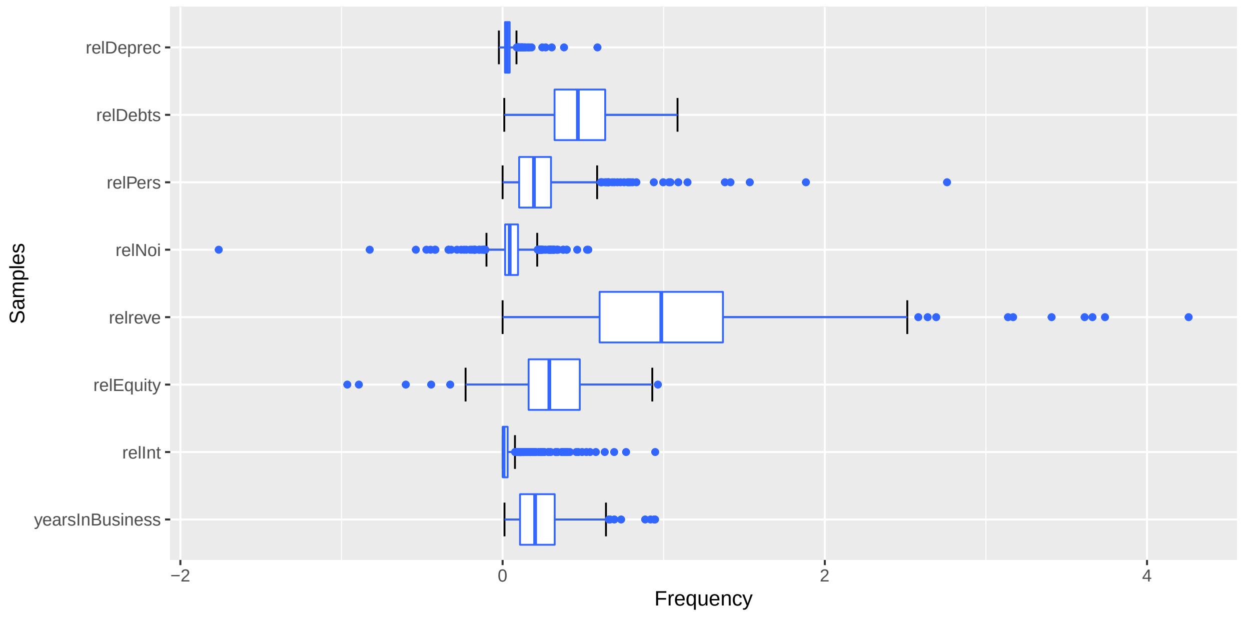
```
#(totAssets, totIntang,accounts,totEquity,debts,prod,revenues,personnel,valCost, ammort, profLoss, valA

data <- data %>% select(totAssets,totEquity, noi,personnel,prod, debts,deprec,valCost,totIntang, revenu
            mutate(relInt =   totIntang/totAssets) %>%
            mutate(relEquity =totEquity/totAssets) %>%
            mutate(yearsInBusiness =   yearsInBusiness/100) %>%
            mutate(relreve = revenues/totAssets)%>%
            mutate(relNoi =       noi/totAssets) %>%
            mutate(relPers =      personnel/totAssets) %>%
            mutate(relDebts =       debts/totAssets) %>%
            mutate(relDeprec =     deprec/totAssets)%>%


            select(-totAssets,-totEquity, -noi,-personnel,-debts,-prod, -deprec,-revenues,-valCost,-tot
```

create normalized variables `{r names(data)}` and scaled

Questions: are the variable of the same order of magnitude?

```
ggplot(stack(data), aes(x = ind, y = values)) +
 stat_boxplot(geom = "errorbar", width = 0.5) +
 labs(x="Samples", y="Frequency") +
 geom_boxplot(fill = "white", colour = "#3366FF") + coord_flip()
```

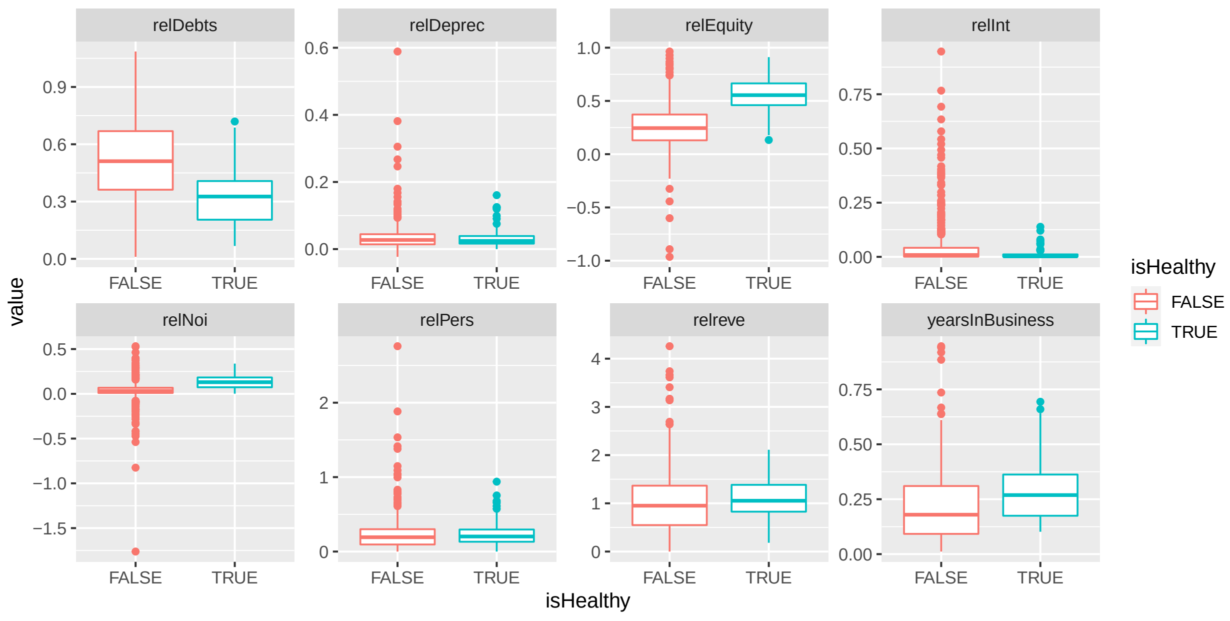
```
## Warning in stack.data.frame(data): non-vector columns will be ignored

## Warning: Removed 7 rows containing non-finite values (stat_boxplot).

## Warning: Removed 7 rows containing non-finite values (stat_boxplot).
```



Questions: are the variables useful to predict the given label? are the distributions different acocrding to the label? Answet: bowplot by laber

```
plot <- data %>% pivot_longer(cols=!isHealthy) %>% ggplot(aes(x=isHealthy, y=value, color = isHealthy))
plot + facet_wrap(~name, ncol = 4,     scales="free")
```

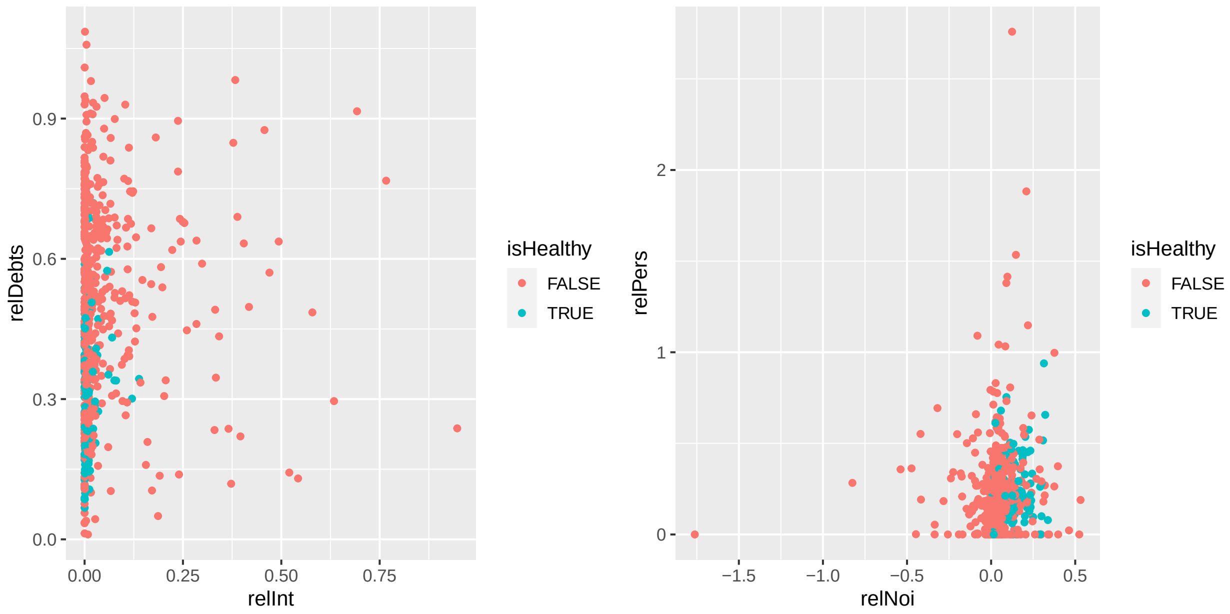
```
## Warning: Removed 7 rows containing non-finite values (stat_boxplot).
```



another plot to answer the same question: can we spot at a glance the decision boundary in a scatterplot? No

```r
plot1 <- data %>% ggplot(aes(x=relInt, y=relDebts,  color=isHealthy)) + geom_point()
plot2 <- data %>% ggplot(aes(x=relNoi, y=relPers, color=isHealthy)) + geom_point()
ggarrange(plot1,plot2)
```

```
## Warning: Removed 1 rows containing missing values (geom_point).

## Warning: Removed 1 rows containing missing values (geom_point).
```



the figure above higlights shows at a glance that is no trivial decision boundary.

Question: Are the distribution of variables different between Healty and non-healty companies? Design: violin boxplots

```
# data %>% pivot_longer(cols=!isHealthy) %>% ggplot(aes(x=isHealthy, y=value,  color=isHealthy)) + geom
# data %>% pivot_longer(cols=!isHealthy) %>% ggplot(aes(x=isHealthy, y=value,  color=isHealthy)) + geom
```

we can see small differences.

Conclusions - Healty and non healty appear hard to tell apart with the given attributes - Which attributes appear more useful for inferring health? relDebts and relNoi

Decision Trees

[short intro to tree learning] We use "tree" that provides: - a function for doing the learning `tree()` - a function for doing the prediction, usually named `predict()`

The learning function in tree requires a dataframe and an indication of the dependency, using a data type, peculiar of R, known as **formula** as in the examples `a ~ b+c` (variable `a` depends on `b` and `c`) or `a ~ .` (variable `a` depends on all the other variables). The following is an example of tree with a limited complexity:

a simple tree

- mincut: The minimum number of observations to include in either child node. This is a weighted quantity; the observational weights are used to compute the 'number'. The default is 5.
- minsize: The smallest allowed node size: a weighted quantity. The default is 10.

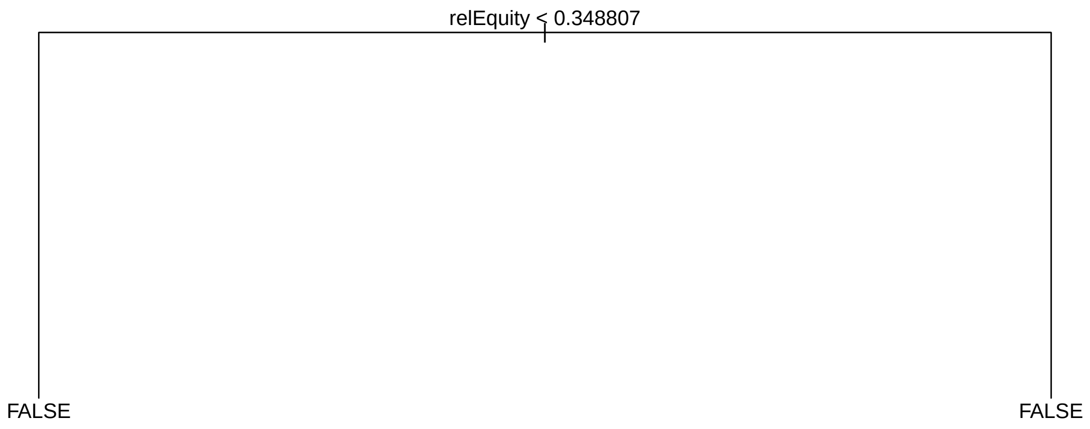
```
require(tree)
```

```
## Caricamento del pacchetto richiesto: tree
```

```
## Warning: il pacchetto 'tree' è stato creato con R versione 4.1.2
```

```
## Registered S3 method overwritten by 'tree':
##   method     from
##   print.tree cli
```

```
model <- tree(isHealthy~.,data,  mindev = .100 )
plot(model)
text(model)
```



```
print(model)
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
## 1) root 637 628.00 FALSE ( 0.80534 0.19466 )
##   2) relEquity < 0.348807 368  77.08 FALSE ( 0.97826 0.02174 ) *
##   3) relEquity > 0.348807 269 367.80 FALSE ( 0.56877 0.43123 ) *
```

measure the performance of the tree

We have a few options:

- measure error on the learning data
- measure error on test data statically left out (e.g., 20% of the overall data)
- measure error with a k-fold CV
- measure error with a **leave-one-out** CV (LOOCV), i.e., a k-fold CV with $k = n$, n being the number of observations in the available data

measure error on the learning data

It's an easy option but we know that, when $k_{\min} = 1$, the error on the learning data is 0 by definition. It would hence pointless to compare a set of 0s… We need to ascertain which is the default value of $k_{\min}$ in `tree()`, that requires to consume the documentation: [I leave this for your enjoiment.[TODO]

measure error on test data

split (e.g., 20% of the overall data) We use `sample()` for shuffling the set of row-indexes of `d` and take a subset of this set that will act as the indexes of the learning data.

```r
fraction <- .8
indexes.learning = sample(c(1:nrow(data)))[1:(nrow(data)*fraction)]
model <- tree(isHealthy~.,data, subset = indexes.learning, mincut = 50 , minsize = 100)
```

The `predict()` function takes a dataframe with possibly new observations and predict the corresponding labels: the results is hence a vector.

```r
predicted.health = predict(model, data[-indexes.learning,], type = "class")
```

Note that:

- the `-` preceding `indexes.learning` means "select all but those"
- `type="class"` is needed to obtain a vector of factors, rather than a more complex thing: see the documentation of `predict.tree()`
- `predict()` doesn't cheat: even if `d[-indexes.learning,]` actually contains also the correct y value, it is not using it

Now we can compute the classification error rate by comparing `predicted.y` against the expected y:

```r
classification.error <- length(which(predicted.health!=data$isHealthy[-indexes.learning]))/length(predi
classification.error
```

```
## [1] 0.15625
```

```r
predicted.learn <- predict(model, data[indexes.learning,], type = "class")
errors.learn    <- tibble (pL = predicted.learn, aL = data[indexes.learning,]$isHealthy) %>%
                    mutate(err = (pL != aL))
err.rate.learn  <- errors.learn %>% filter(err == TRUE) %>%
                    nrow()/length(predicted.learn)

predicted.test  <- predict(model, data[-indexes.learning,], type = "class")
errors.test     <- tibble (pT = predicted.test, aL = data[-indexes.learning,]$isHealthy) %>%
                    mutate(err = (pT != aL))
err.rate.test <-     errors.test  %>% filter(err == TRUE) %>% nrow()/length(predicted.test)

print(paste('Error rate on learning data: ',round(err.rate.learn,3 )))
```

```
## [1] "Error rate on learning data:  0.12"
```

```r
print(paste('Error rate on test data:     ',round(err.rate.test, 3 )))
```

```
## [1] "Error rate on test data:      0.156"
```

```r
# classification.error <- length(which(predicted.test!=data$isHealthy[-indexes.learning]))/length(predi
# classification.error
```

Another way is to "compute" the **confusion matrix** and then obtaining the error from that. The confusion matrix shows the number misclassifications, class by class:

```
table(predicted.test, data$isHealthy[-indexes.learning])
```

```
##
## predicted.test FALSE TRUE
##          FALSE    89   12
##          TRUE      8   19
```

Given that matrix, the accuracy of classification is:

```
conf.matrix = table(predicted.test, data$isHealthy[-indexes.learning])
sum(diag(conf.matrix))/sum(conf.matrix)
```

```
## [1] 0.84375
```

and the error rate can be computed as:

```
error = 1-sum(diag(conf.matrix))/sum(conf.matrix)
round(error,3)
```

```
## [1] 0.156
```

Out of simplicity, we might build a function that does all those operations together, with some parameters:

```
computeErrorRate = function(categorical.y.name, data,  learner, p.learn = 0.8, ...) {

  print(paste("learning a model for ", categorical.y.name, "using parameter = ", p.learn))

  indexes.learning = sample(c(1:nrow(data)))[1:(nrow(data)*p.learn)]
  model = learner(formula(paste0(categorical.y.name,"~.")), data[indexes.learning,], ...)

  predicted.y = predict(model, data[-indexes.learning, ], type="class")
  errors <- data[-indexes.learning,] %>%
    select(categorical.y.name) %>%
    mutate(ph = predicted.y) %>%
    mutate(err = (categorical.y.name = ph)) %>%
    filter(err == TRUE) %>%
    nrow()

    errors/length(predicted.y)
}

print(computeErrorRate("isHealthy", data, tree))
```

```
## [1] "learning a model for  isHealthy using parameter =  0.8"
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use ‘all_of(categorical.y.name)‘ instead of ‘categorical.y.name‘ to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```
## [1] 0.171875
```

measure error ratio over different values of the learn to test rario. The values are stocastic due to the random selection of learn and test datasets

```
#computeErrorRate = function(categorical.y.name, data, learner, p.learn = 0.8, ...) {
#
# ratio = seq(0.1, 0.99, 0.05)
# error = ratio %>% map_dbl(function(r){computeErrorRate("isHealthy", data, tree, r)})
# tibble(ratio=ratio, error=error) %>% ggplot(aes(x=ratio,y=error))+geom_line()+ylim(0,1)
```

check overfitting

compare errors on learning and training dataset

```
p.learn <- .8
indexes.learning = sample(c(1:nrow(data)))[1:(nrow(data)*p.learn)]

model <- tree(isHealthy~.,data[indexes.learning,])

predicted.y <- predict(model, data[indexes.learning, ], type="class")
err.rate.learning <- data[indexes.learning,] %>%
    select(isHealthy) %>%
    mutate(ph = predicted.y) %>%
    mutate(err = (categorical.y.name = ph)) %>%
    filter(err == TRUE) %>%
    nrow()/length(predicted.y)

predicted.y <- predict(model, data[-indexes.learning, ], type="class")
err.rate.training <- data[-indexes.learning,] %>%
    select(isHealthy) %>%
    mutate(ph = predicted.y) %>%
    mutate(err = (categorical.y.name = ph)) %>%
    filter(err == TRUE) %>%
    nrow()/length(predicted.y)


err.rate.learning
```

```
## [1] 0.1980392
```

```
err.rate.training
```

```
## [1] 0.1640625
```

Compute the learning error and test error for different values of the learning-to-test data ratio

```
# computeErrorRate2 = function(categorical.y.name, data, learner, p.learn = 0.8, ...) {
#
#   print(paste("learning a model for ", categorical.y.name, "using parameter = ", p.learn))
```

```
#
#   indexes.learning = sample(c(1:nrow(data)))[1:(nrow(data)*p.learn)]
#   model = learner(formula(paste0(categorical.y.name,"~.")), data[indexes.learning,], ...)
#
#   predicted.y.learning = predict(model, data[indexes.learning, ], type="class")
#   errors.learning <- data[indexes.learning,] %>%
#     select(categorical.y.name) %>%
#     mutate(ph = predicted.y.learning) %>%
#     mutate(err = (categorical.y.name = ph)) %>%
#     filter(err == TRUE) %>%
#     nrow()/length(predicted.y.learning)
#
#   predicted.y.test = predict(model, data[-indexes.learning, ], type="class")
#   errors.test <- data[-indexes.learning,] %>%
#     select(categorical.y.name) %>%
#     mutate(ph = predicted.y.test) %>%
#     mutate(err = (categorical.y.name = ph)) %>%
#     filter(err == TRUE) %>%
#     nrow()/length(predicted.y.test)
#
#   c(errors.learning,errors.test)
# }
#
# ratio = seq(0.1, 0.99, 0.02)
# error1 = ratio %>% map_dbl(function(r){computeErrorRate2("isHealthy", data, tree, r, mindev = 0, mins
# error2 = ratio %>% map_dbl(function(r){computeErrorRate2("isHealthy", data, tree, r, mindev = 0, mins
# tibble(ratio=ratio, error1=error1, error2=error2) %>%
#   ggplot()+geom_line(aes(x=ratio,y=error1, color = 'blue'))+geom_line(aes(x=ratio,y=error2, color = '
```

Compute the learning error and test error for different values of the complexity parameter

Arguments nobs
The number of observations in the training set.

mincut = The minimum number of observations to include in either child node. This is a weighted quantity; the observational weights are used to compute the 'number'. The default is 5.

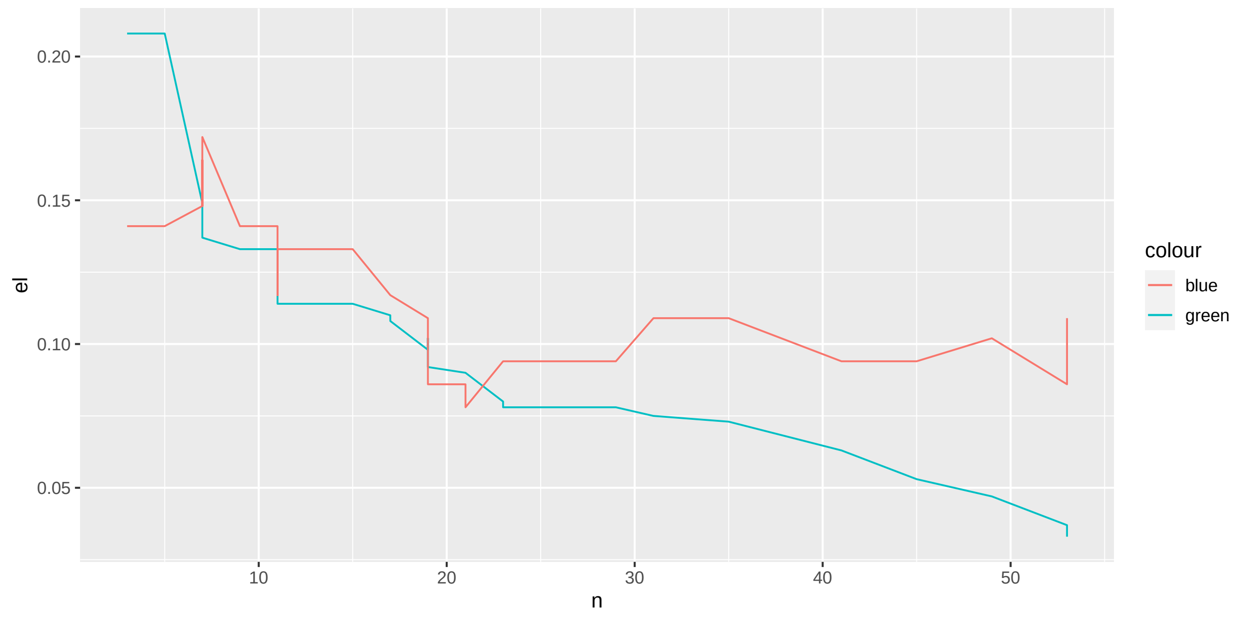
minsize = The smallest allowed node size: a weighted quantity. The default is 10.

mindev = The within-node deviance must be at least this times that of the root node for the node to be split.

```
computeErrorRate2 = function(categorical.y.name, data, learner, p.learn = 0.8, mindev = 2,...) {

  print(paste("learning a model for ", categorical.y.name, "using mindev = ", mindev))

  indexes.learning = sample(c(1:nrow(data)))[1:(nrow(data)*p.learn)]
  model = learner(formula(paste0(categorical.y.name,"~.")), data[indexes.learning,], ...)

  predicted.y.learning = predict(model, data[indexes.learning, ], type="class")
  errors.learning <- data[indexes.learning,] %>%
    select(categorical.y.name) %>%
    mutate(ph = predicted.y.learning) %>%
    mutate(err = (categorical.y.name = ph)) %>%
```

```r
    filter(err == TRUE) %>%
    nrow()/length(predicted.y.learning)

  predicted.y.test = predict(model, data[-indexes.learning, ], type="class")
  errors.test <- data[-indexes.learning,] %>%
    select(categorical.y.name) %>%
    mutate(ph = predicted.y.test) %>%
    mutate(err = (categorical.y.name = ph)) %>%
    filter(err == TRUE) %>%
    nrow()/length(predicted.y.test)

  return(c(errors.learning,errors.test))

}

# mindevs = seq(0, 1, .1)
# error1 = mindevs %>% map_dbl(function(r){computeErrorRate2("isHealthy", data, tree, 0.8, mindev = min
# error2 = mindevs %>% map_dbl(function(r){computeErrorRate2("isHealthy", data, tree, 0.8, mindev = min
#
# ddddd <- tibble(mindevs = mindevs, error1=error1, error2=error2)
#
# ddddd %>%  ggplot()+geom_line(aes(x=mindevs,y=error1, color = 'blue'))+geom_line(aes(x=mindevs,y=erro
```

```r
p.learn <- .8
indexes.learning = sample(c(1:nrow(data)))[1:(nrow(data)*p.learn)]

df <- tibble(i=0, n=0, el=0, et=0) %>% head(0)

# i = mincut The minimum number of observations to include in either child node. This is a weighted qua

for(i in seq(150,1,-1)){
  model <- tree(isHealthy~.,data, subset = indexes.learning, mincut = i , mindev=.005) #1,200,10

  predicted.learn = predict(model, data[indexes.learning,], type = "class")
  predicted.test  = predict(model, data[-indexes.learning,], type = "class")
  errors.learn <- tibble (L = predicted.learn, D = data[indexes.learning,]$isHealthy) %>%
                mutate(err = (L != D))
  errors.test <- tibble  (T = predicted.test, D = data[-indexes.learning,]$isHealthy) %>%
                mutate(err = (T != D))

  err.rate.learning <- errors.learn %>% filter(err == TRUE) %>% nrow()/length(predicted.learn)
  err.rate.test <-     errors.test  %>% filter(err == TRUE) %>% nrow()/length(predicted.test)
  number.of.nodes = nrow(model$frame)
  #print(paste(i, number.of.nodes,err.rate.learning, err.rate.test))
  df <- df %>% add_row(i = i , n=number.of.nodes,el=round(err.rate.learning,3), et = round(err.rate.tes
}

plot1 <- df %>% filter(i>1)%>%ggplot()+
  geom_line(aes(x = n, y=el, color='green'))+
  geom_line(aes(x = n, y=et, color='blue'))
plot1
```



```
# plot2 <- df %>% filter(i>1)%>%ggplot()+
#   geom_line(aes(x = i, y=el, color='green'))+
#   geom_line(aes(x = i, y=et, color='blue'))
# ggarrange(plot1,plot2)
#
# mins = df %>% group_by(et) %>% slice(which.min(et))
#
# print(paste("minimum error for i = ", mins$i[1], "tree composed of ", mins$n[1], "nodes, error on tes
```

k-fold cross validation for error estimate

How will themodel perform on "unseen data"? can the learner generalize beyond available data? The classification error is an appropriate indicator. But the error depends on a random choice of learn and test data. We need to have a more stable value, so we introduce k-fold cross validation on the whole dataset.

1. split learning data (X and y) in k equal slices (each of n k observations)
2. for each split (i.e., each i {1, . . . , k}) 2.1 learn on all but k-th slice 2.2 compute classification error on unseen k-th slice
3. average the k classification errors

stratified k-fold CV using dplyr,

```
p.learn <- .8
indexes.learning = sample(c(1:nrow(data)))[1:(nrow(data)*p.learn)]
test_data <- data[-indexes.learning,]
learn_data <- data[indexes.learning,]
k_folds = 10

learn_data <- learn_data %>%
  group_by(isHealthy) %>%
  sample_frac(1) %>%
  mutate(fold=rep(1:k_folds, length.out=n())) %>%
  ungroup

errors = tibble(f = 0, err = 0) %>% head(0)

mincut = 10
minsize = i*2

for(i in 1:k_folds){
  data.kth.fold <- learn_data %>% filter(fold==i)
  data.other.folds <- learn_data %>% filter(fold!=i)

  # learn on  data.other.folds
  model <- tree(isHealthy~.,data, subset = indexes.learning, mincut = i , minsize = i*2)
  # estimate learn error on data.other.folds
  predicted.learn = predict(model, data[indexes.learning,], type = "class")
  errors.learn <- tibble (L = predicted.learn, D = data[indexes.learning,]$isHealthy) %>%
              mutate(err = (L != D))
  # estimate test error on  data.kth.fold
  predicted.test  = predict(model, data[-indexes.learning,], type = "class")
  errors.test <- tibble  (T = predicted.test, D = data[-indexes.learning,]$isHealthy) %>%
              mutate(err = (T != D))

  err.rate.learning <- errors.learn %>% filter(err == TRUE) %>% nrow()/length(predicted.learn)
  err.rate.test <-     errors.test  %>% filter(err == TRUE) %>% nrow()/length(predicted.test)
  number.of.nodes = nrow(model$frame)
  #print(paste(i, number.of.nodes,err.rate.learning, err.rate.test))
  df <- df %>% add_row(i = i , n=number.of.nodes,el=round(err.rate.learning,3), et = round(err.rate.tes

  # compute classification error on unseen data.kth.fold
  class.err = 1
  errors <-  errors %>% add_row(f = i , err = round(class.err,3))


}

# average the k classification errors
mean(errors$err)
```

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

Pruning

Random Forrest

Conclusions

k-fold cross validation for error estimate

How will themodel perform on "unseen data"? can the learner generalize beyond available data? The classification error is an appropriate indicator. But the error depends on a random choice of learn and test data. We need to have a more stable value, so we introduce k-fold cross validation on the whole dataset.

1. split learning data (X and y) in k equal slices (each of n k observations)
2. for each split (i.e., each i in {1, . . . , k}) 2.1 learn on all but k-th slice 2.2 compute classification error on unseen k-th slice
3. average the k classification errors

mean error estimation using stratified k-fold CV

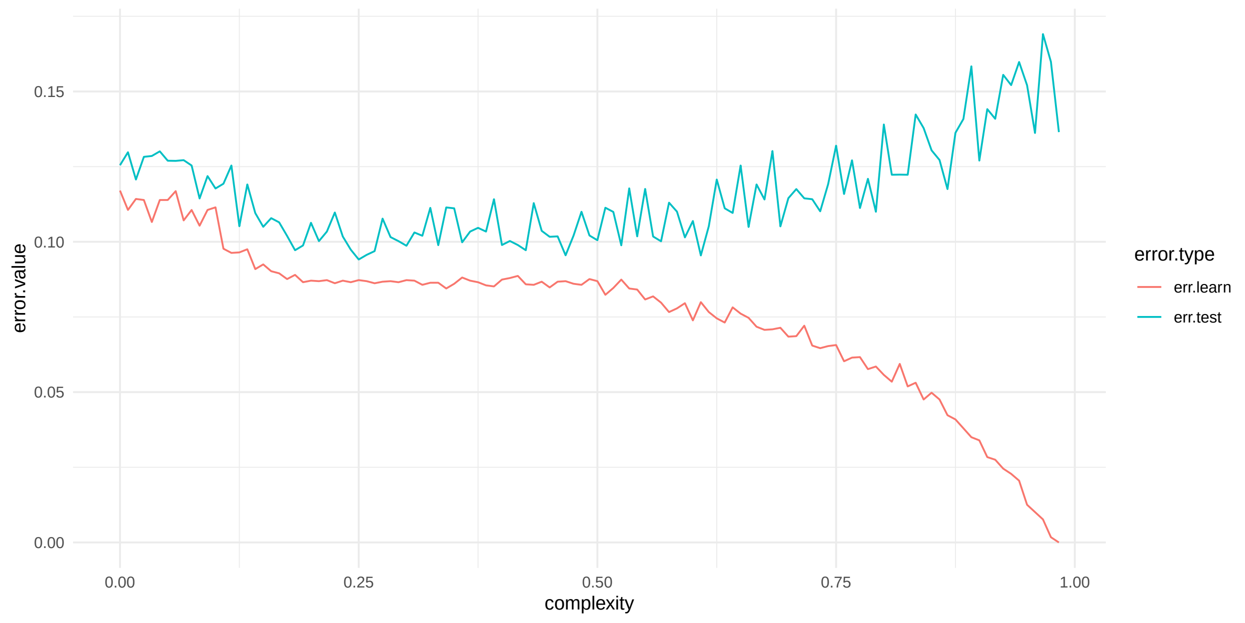
```r
library(rpart)
```

```
## Warning: il pacchetto 'rpart' è stato creato con R versione 4.1.2
```

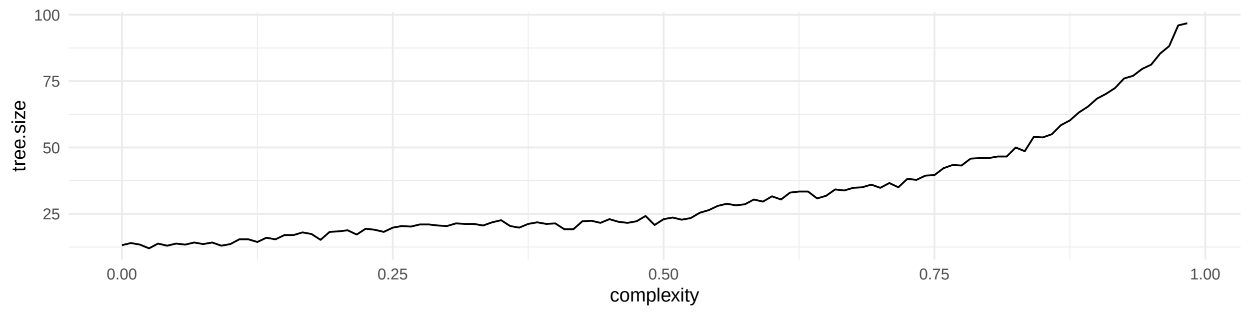
```r
compute.Kfold.error <- function(data,  k_folds=10, mincut) {
  data_f <- data %>%
    group_by(isHealthy) %>%
    sample_frac(1) %>%
    mutate(fold=rep(1:k_folds, length.out=n())) %>%
    ungroup

  fold.errors = tibble(f = NA, err.learn = NA, err.test = NA,  n = NA, mincut = NA) %>% head(0)


  for(i in 1:k_folds){

    data.kth.fold <- data_f %>% filter(fold==i)
    data.other.folds <- data_f %>% filter(fold!=i)

    # learn on  data.other.folds
    ##model <- tree(isHealthy~.,data.other.folds, mincut = mincut,mindev = mindev )

    model <- rpart(  isHealthy~.,data.other.folds,method = "class", minsplit = mincut, minbucket = 1, c
    number.of.nodes = nrow(model$frame)

    # estimate learn error on data.other.folds
    predicted.learn = predict(model, data.other.folds, type = "class")
    errors.learn <- tibble (L = predicted.learn, D = data.other.folds$isHealthy) %>%
                  mutate(err = (L != D))

    # estimate test error on  data.kth.fold
    predicted.test  = predict(model, data.kth.fold, type = "class")
    errors.test <- tibble  (T = predicted.test, D = data.kth.fold$isHealthy) %>%
                  mutate(err = (T != D))

    err.rate.learning <- errors.learn %>% filter(err == TRUE) %>% nrow()/length(predicted.learn) %>% rou
    err.rate.test <-     errors.test  %>% filter(err == TRUE) %>% nrow()/length(predicted.test) %>% rour

    #save results
    fold.errors <-  fold.errors %>%
      add_row(f = i , err.learn = err.rate.learning, err.test = err.rate.test,   n = number.of.nodes, mi


  }
  fold.errors <- fold.errors %>% group_by(mincut) %>%
  summarize(err.learn = mean(err.learn), err.test = mean(err.test), n = mean(n) )
  return(fold.errors)
}


k_folds=10
maxmc = 120
errors = tibble(err.learn = NA, err.test = NA,  n = NA, mincut = NA) %>% head(0)

for (mc in seq(maxmc,2,-1)){
 errors <- errors %>% bind_rows(compute.Kfold.error(data,  k_folds=k_folds, mincut=mc ))
}

print(errors)
```

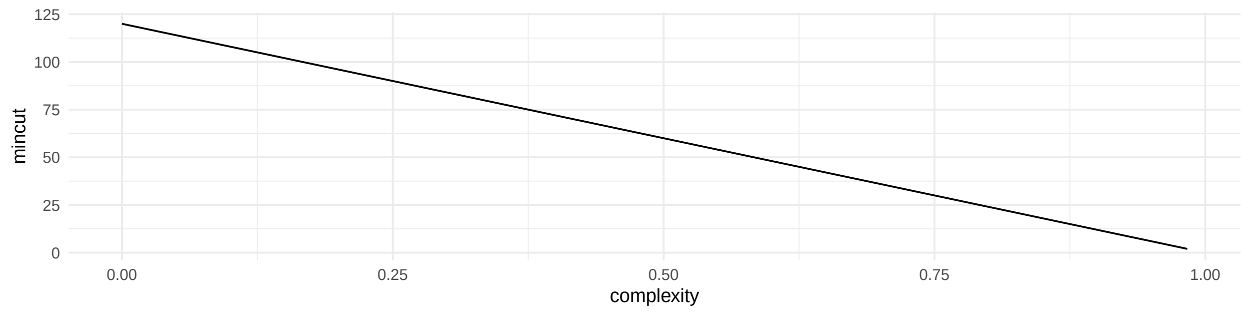
```
## # A tibble: 119 x 4
##    err.learn err.test     n mincut
##        <dbl>    <dbl> <dbl>  <dbl>
##  1     0.117    0.125  13.2    120
##  2     0.111    0.130  14      119
##  3     0.114    0.121  13.4    118
##  4     0.114    0.128  12      117
##  5     0.107    0.129  13.8    116
##  6     0.114    0.130  13      115
##  7     0.114    0.127  13.8    114
##  8     0.117    0.127  13.4    113
##  9     0.107    0.127  14.2    112
## 10     0.111    0.125  13.6    111
## # ... with 109 more rows
```

```
# # average the k classification errors
# mean.errors <- errors %>% group_by( mincut)%>%
#   summarize(mL = mean(err.learn), mT = mean(err.test), n = mean(n) )

#plot using a tidy data structure to represent data and legend:  gather (or pivot_longer) to bring all

data.to.plot <- errors %>%
  mutate(tree.size = n)%>% select(-n) %>%
  gather(error.type, error.value,  c( err.test, err.learn)) %>%
  mutate(complexity = (maxmc-mincut)/maxmc)

plot1 <- data.to.plot  %>% ggplot( ) + theme_minimal()+
  geom_line(aes(x=complexity, y = error.value, group = error.type, colour = error.type))

plot1
```



```
plot2 <- data.to.plot %>% ggplot( ) +  theme_minimal()+
  geom_line(aes(x=complexity, y =tree.size ))

plot3 <- data.to.plot %>% ggplot( ) +  theme_minimal()+
  geom_line(aes(x=complexity, y =mincut ))

ggarrange( plot2, plot3, ncol = 1, nrow = 2)
```





```r
min.test.error <- min(errors$err.test)
minima <- errors %>% filter(err.test == min.test.error) %>% arrange(mincut)
minima[1,] #there may be more rows
```

```
## # A tibble: 1 x 4
##   err.learn err.test     n mincut
##       <dbl>    <dbl> <dbl>  <dbl>
## 1    0.0873   0.0941  19.8     90
```

So we should tune the parameter according to minima[1,]

```r
library(caret)
```

```
## Warning: il pacchetto 'caret' è stato creato con R versione 4.1.2
```

```
## Caricamento del pacchetto richiesto: lattice
```

```
##
## Caricamento pacchetto: 'caret'
```

```
## Il seguente oggetto è mascherato da 'package:purrr':
##
##     lift
```

```r
data <- data %>% na.omit ()
train_control <- trainControl(method = "cv",
                              number = 10)
```

```r
# building the model and
# predicting the target variable
# as per the Naive Bayes classifier
model <- train(isHealthy~., data = data,
               trControl = train_control,
               method = "rpart")

print(model)
```

```
## CART
##
## 637 samples
##   8 predictor
##   2 classes: 'FALSE', 'TRUE'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 572, 574, 574, 574, 573, 572, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.02419355  0.8980052  0.6573799
##   0.14516129  0.8853797  0.6226522
##   0.19758065  0.8242361  0.2538895
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.02419355.
```

```r
library(rpart)

library(rpart.plot)
```

```
## Warning: il pacchetto 'rpart.plot' è stato creato con R versione 4.1.2
```

```r
library(RColorBrewer)
```

```
## Warning: il pacchetto 'RColorBrewer' è stato creato con R versione 4.1.1
```

```r
  model <- rpart(  isHealthy~.,
                   data=data,
                   method = "class",
                   minsplit = 2, minbucket = 1, cp = 0.022)


# plot mytree
```