Excercise: binary classification

Fabio Morea

2022-02-10

Contents

. . .

1. Introduction

Description of the problem and a motivation for addressing it, why relevant.

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, ...)

The method should be presented to a group of people with a basic knowledge of R.

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).

2. Formal statement of the problem

Define a model such that ...

3. Indexes for measuring the solutions

- error rate

4. Data exploration and feature engineering

- most relevant features
- required to feasibility of proposed solution with the data available

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

```
#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( 25)
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"
```

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

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

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

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

```
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>3))  %>%
          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

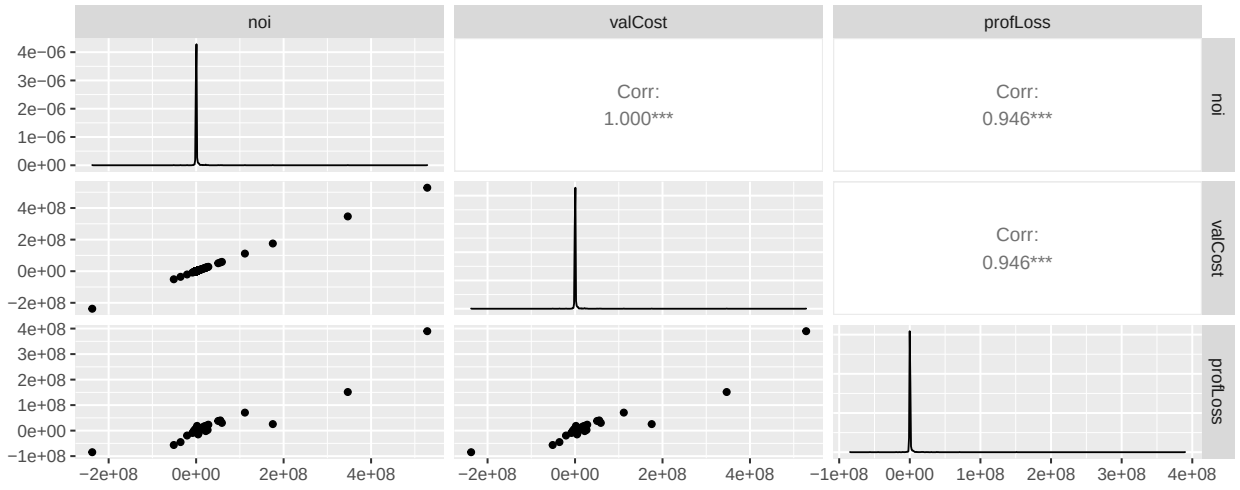
```
require(GGally)
```

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

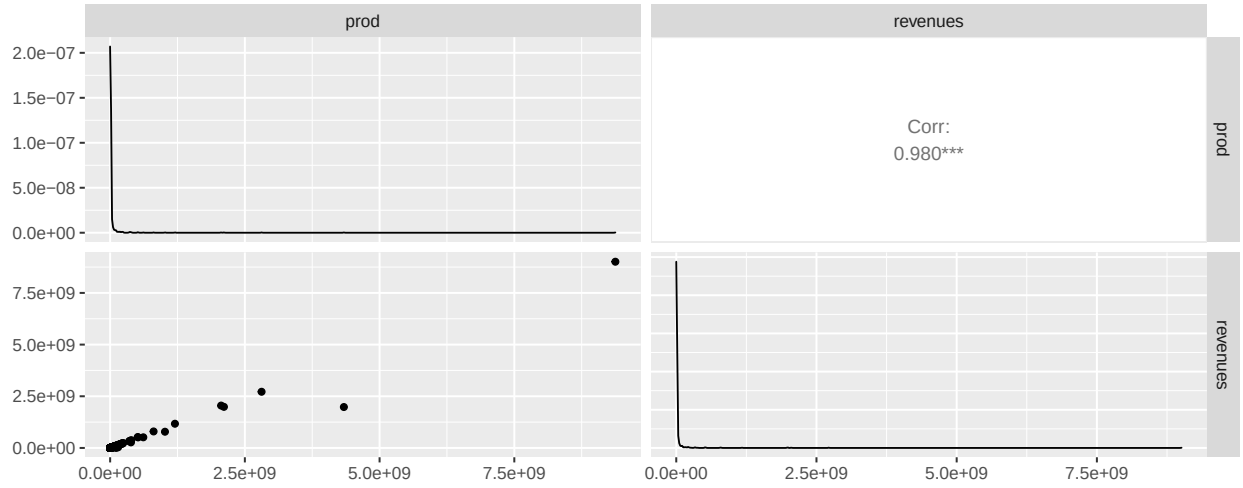
```
## Warning: il pacchetto 'GGally' è stato creato con R versione 4.1.2
```

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

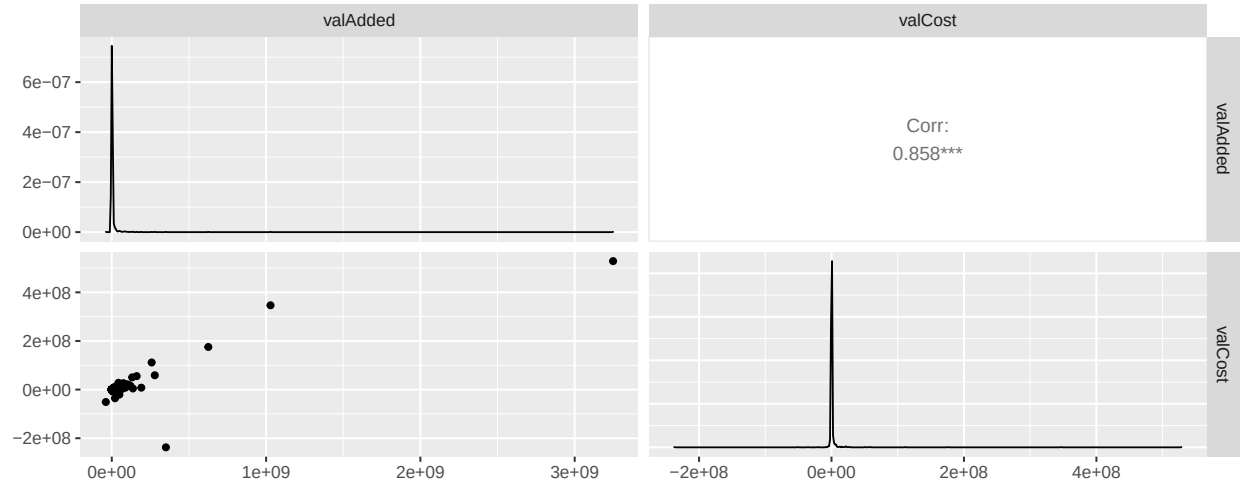
```
data %>%  select(noi, valCost, profLoss) %>% ggpairs()
```



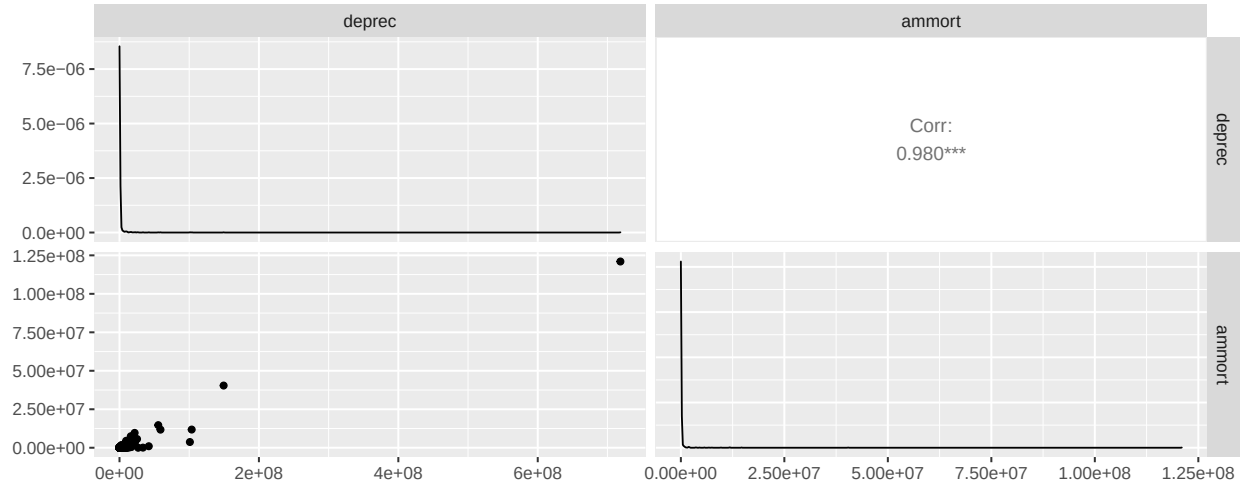

```
data %>%  select(prod, revenues) %>% ggpairs()
```



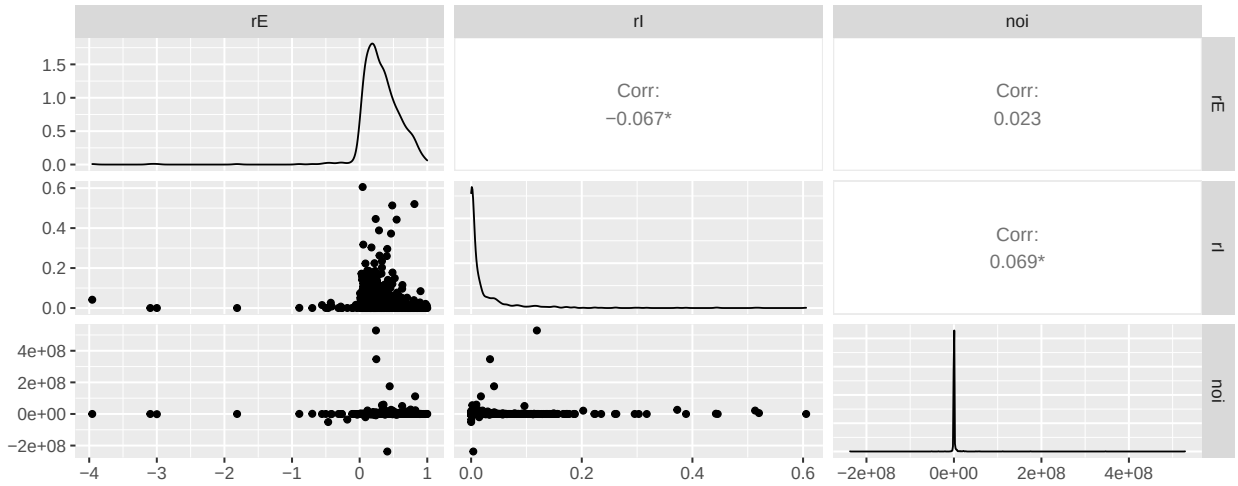

```
data %>%  select(valAdded, valCost) %>% ggpairs()
```




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



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



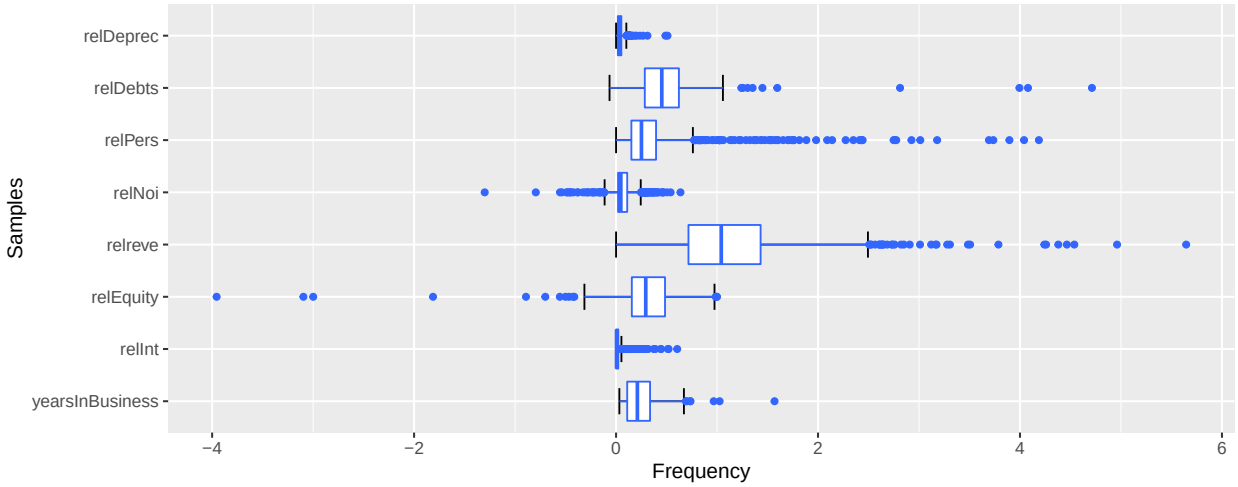
we check for the scale of each variable

```
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?

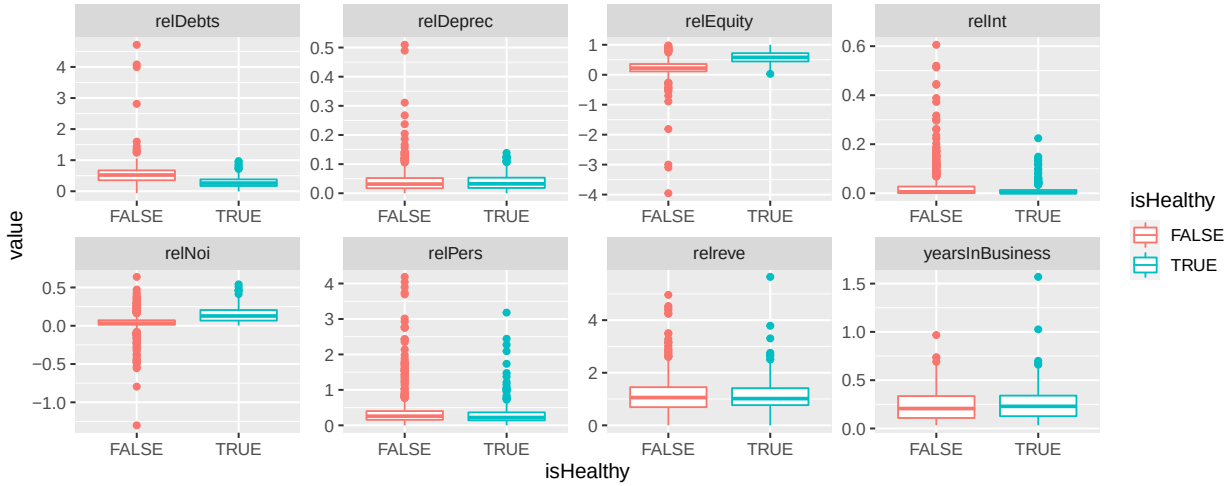
```
ggplot(stack(data), aes(x = ind, y = values)) +
 stat_boxplot(geom = "errorbar", width = 0.5) +
 labs(x="Samples", y="Frequency") +
 geom_boxplot(fill = "white", colour = "#3366FF") + coord_flip()
```

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



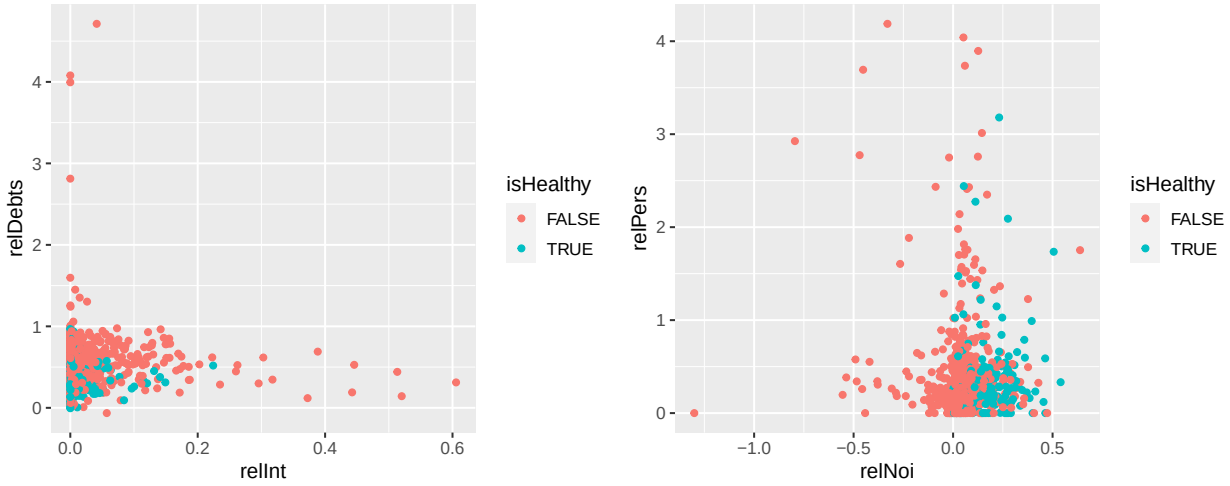
Questions: are the variables useful to predict the given label? are the distributions different acocrding to the label? Answet: bowplot by laber

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



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

```
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)
```



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

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

5. Proposed solution

5.1 trees (intuition)

two examples of trees with different values of the hyperparameter k

[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:

- 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.

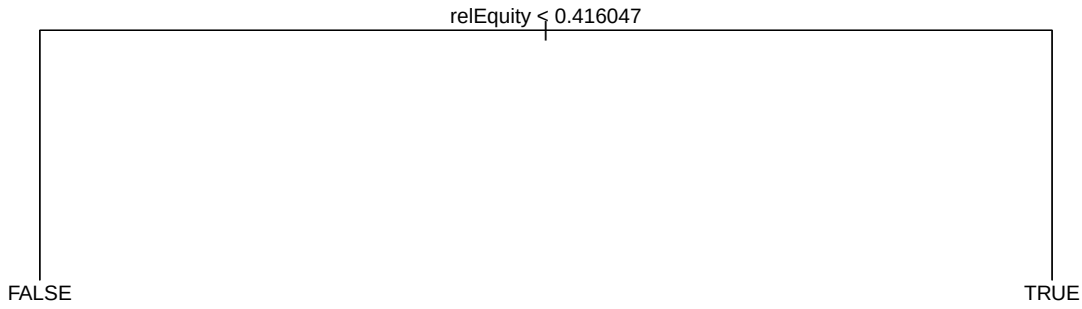
```
require(tree)
```

```
## 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 988 1118.0 FALSE ( 0.74696 0.25304 )
##   2) relEquity < 0.416047 665  349.9 FALSE ( 0.92632 0.07368 ) *
##   3) relEquity > 0.416047 323  428.3 TRUE ( 0.37771 0.62229 ) *
```

5.2 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.

```
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.

```
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:

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

```
## [1] 0.1313131
```

```
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:  0.127"
```

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

```
## [1] "Error rate on test data:      0.131"
```

```
# 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   137   17
##          TRUE      9   35
```

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] 0.8686869
```

and the error rate can be computed as:

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

```
## [1] 0.131
```

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,] %>%
    mutate(err = (categorical.y.name = predicted.y)) %>%
    filter(err == TRUE) %>%
    nrow()

  return(errors/length(predicted.y))

}

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

```
## [1] "learning a model for  isHealthy using parameter =  0.8"
## [1] 0.2373737
```

5.3 check overfitting

compare errors on learning and training dataset

TODO: use the computeErrorrate function in the code below

```
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] 0.2278481
```

```
err.rate.training
```

```
## [1] 0.2272727
```

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

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

```
library(rpart)
```

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

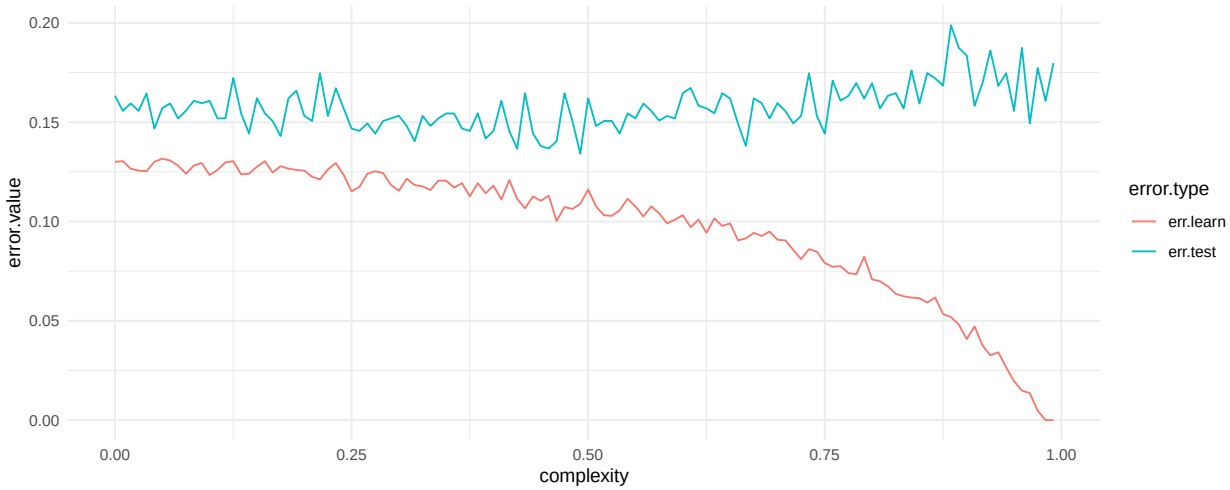
```
compute.Kfold.error <- function(data,  k_folds=10, minsplit) {
  data_f <- data %>%
    group_by(isHealthy) %>%
    sample_frac(1) %>%
    mutate(fold=rep(1:k_folds, length.out=n())) %>%
    ungroup

  fold.errors = tibble(f = NA, err.learn = NA, err.test = NA,  n = NA, minsplit = 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 <- rpart(  isHealthy~.,data.other.folds,method = "class", minsplit = mspl, minbucket = 1, cp =
    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) %>% roun

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


  }

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

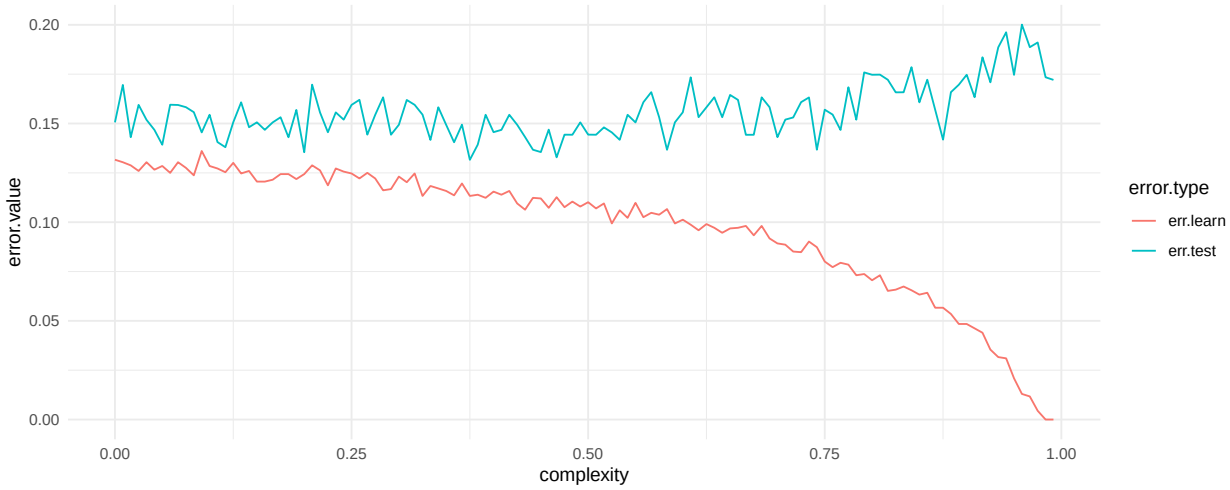
Here we start with nested CV cycles:

```
require(rpart)

#splot train and test data
fraction = .8
indexes.learning = sample(c(1:nrow(data)))[1:(nrow(data)*fraction)]
data.learn  <- data %>% slice(indexes.learning)
data.test <- data %>% slice(-indexes.learning)

minsplit_candidates = seq(120,1,-1)
m = length(minsplit_candidates)
maxms=max(minsplit_candidates)

k_folds_testing=5
k_folds_tuning=5

errors1 <- tibble(i=NA , err=NA, number.of.nodes=NA, minsplit=NA) %>% head(0)
plots   <- tibble()
```

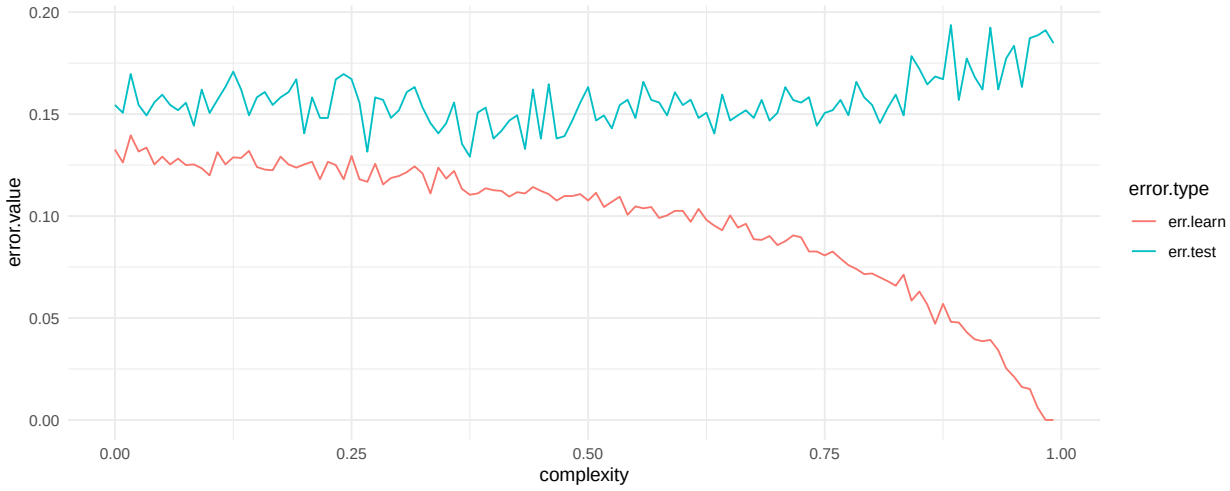
```r
for(i in 1:k_folds_testing){

  #initialize empty tibble for errors
  errors2 = tibble(err.learn = NA, err.test = NA,  n = NA, minsplit = NA) %>% head(0)

  # loop to calculate errors for all minsplit candidates
  for (mspl in minsplit_candidates){
    errors2 <- errors2 %>% bind_rows(compute.Kfold.error(data.learn,  k_folds=k_folds_tuning, minsplit=m
  }
 # choose the optimal value of mspl

 data.to.plot <- errors2 %>%
   mutate(tree.size = n)%>% select(-n) %>%
   gather(error.type, error.value,  c( err.test, err.learn)) %>%
   mutate(complexity = (maxms-minsplit)/maxms)%>%
   mutate(i=i)

 #save errors for plots at the end of the cycle
 ff =  rep(i, nrow(errors2))
 et <- errors2$err.test
 el <- errors2$err.learn
 ms <- errors2$minsplit
 plots <- plots %>% rbind(plots, tibble(ff, et, el, ms))

 print(ggplot( data.to.plot) + theme_minimal()+
  geom_line(aes(x=complexity, y = error.value, group = error.type, colour = error.type)))

 best = errors2 %>% arrange(err.test) %>% head(1)
 mspl.star = best$minsplit[1]
 print(mspl.star)

 #tune model with optimal value
 opt.model <- rpart(  isHealthy~.,data.learn, method = "class", minsplit = mspl.star, minbucket = 1, cp
 opt.number.of.nodes = nrow(model$frame)

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

 opt.err <- opt.errors %>% filter(err == TRUE) %>%  nrow()

 #save results
 errors1 <-  errors1 %>%
   add_row(i = i , err = opt.err, number.of.nodes = opt.number.of.nodes, minsplit = mspl.star)

}
```


0.20
0.15
0.10
0.05
0.00
error.value
0.00
0.25
0.50
0.75
1.00
complexity
error.type
err.learn
err.test

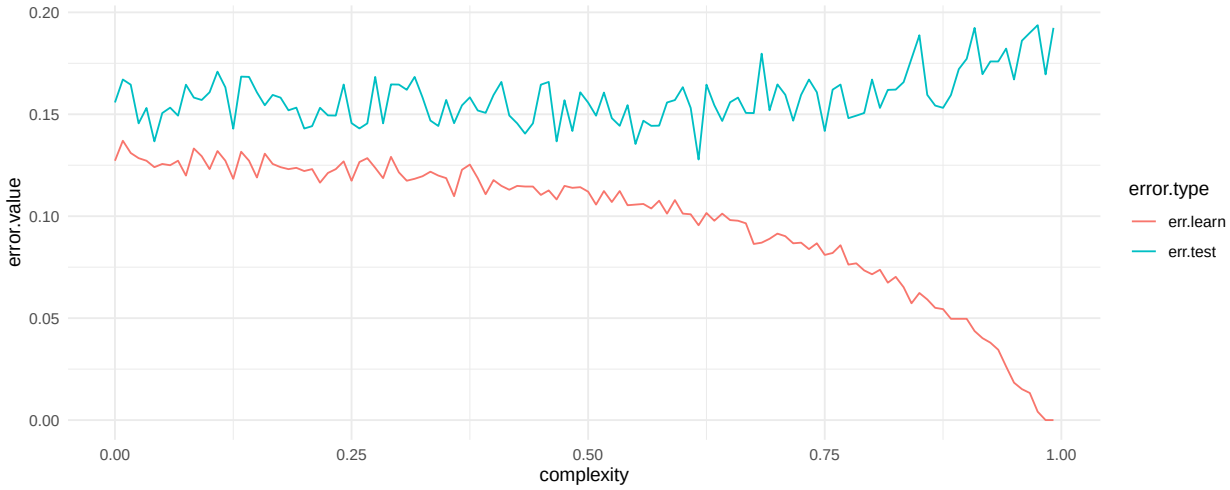
```
## [1] 61
```


0.20
0.15
0.10
0.05
0.00
error.value
0.00
0.25
0.50
0.75
1.00
complexity
error.type
err.learn
err.test

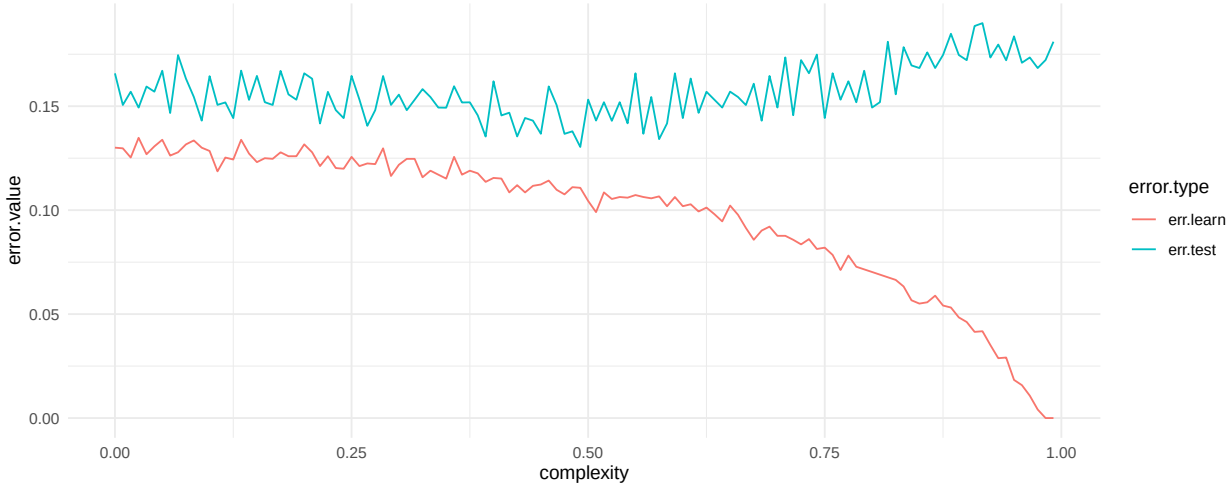
```
## [1] 75
```


0.20
0.15
0.10
0.05
0.00
error.value
0.00
0.25
0.50
0.75
1.00
complexity
error.type
err.learn
err.test

```
## [1] 75
```




```
## [1] 46
```




```
## [1] 61
```

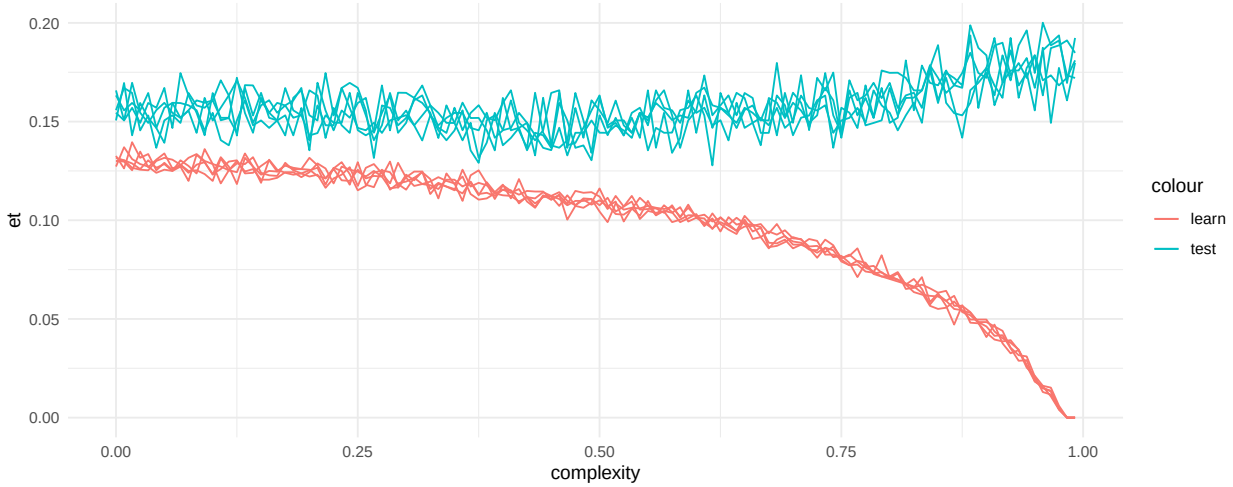
```
errors1
```

```
## # A tibble: 5 x 4
##       i   err number.of.nodes minsplit
##   <int> <int>           <int>    <dbl>
## 1     1    25              17       61
## 2     2    22              17       75
## 3     3    22              17       75
## 4     4    22              17       46
## 5     5    25              17       61
```

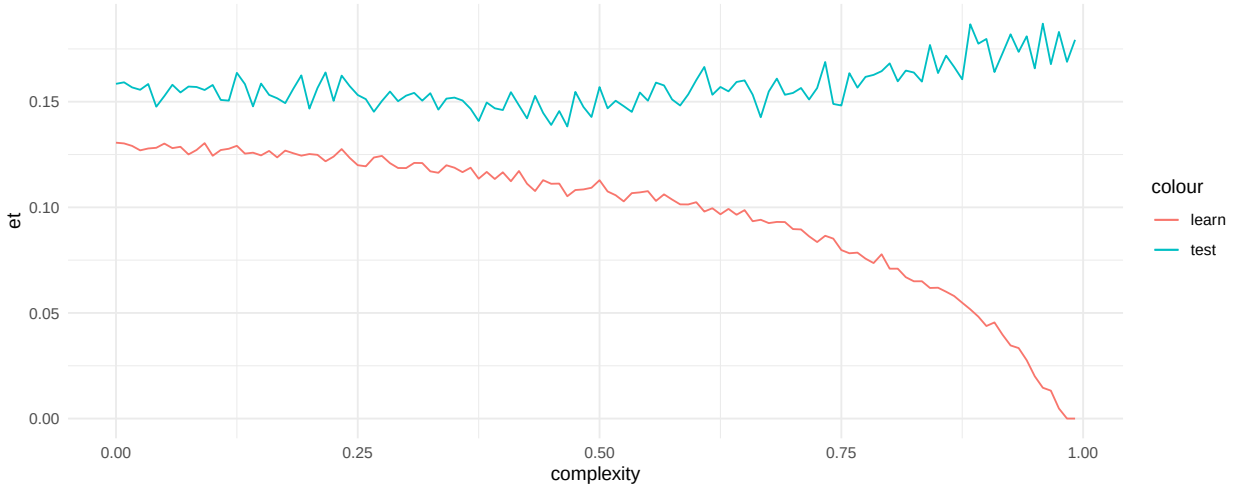
```
names(plots)
```

```
## [1] "ff" "et" "el" "ms"
```

```
plots %>%
  mutate(complexity = (maxms-ms)/maxms)%>%
  ggplot() + theme_minimal()+
  geom_line(aes(x=complexity, y = et, group = ff, colour = 'test')) +
  geom_line(aes(x=complexity, y = el, group = ff, colour = 'learn'))
```




```
plots %>%
  mutate(complexity = (maxms-ms)/maxms)%>%
  group_by(complexity) %>% summarise(el = mean(el), et=mean(et))%>%
  ggplot() + theme_minimal()+
  geom_line(aes(x=complexity, y = et,   colour = 'test')) +
  geom_line(aes(x=complexity, y = el,  colour = 'learn'))
```



```
print(mspl.star)
```

```
## [1] 61
```

show eact values of minsplit

optimized model

train the model with optimal minsplit. Not a single value, but a choiche for a rather flat zone. next figure: Complexity / Minsplit / Tree depth

Assess robustness against change in sector // applicability to various sectors

conclusions

- classification trees are ok

further investigations:

- other error indicators
- regression trees
- random forrest
- svm
- naive bayes
- knn