Excercise: binary classification

Fabio Morea

2022-02-08

# 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

#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, totAssets, totIntang,accounts,totEquity,debts,prod,revenues,personnel,valCost, ammort, profLoss, valAdded, deprec, noi)  
  
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

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, profLoss, valAdded, deprec, noi)  
#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="free")

#(totAssets, totIntang,accounts,totEquity,debts,prod,revenues,personnel,valCost, ammort, profLoss, valAdded, deprec, noi)  
  
data <- data %>% select(totAssets,noi,personnel, debts,deprec,valCost,totIntang, revenues,yearsInBusiness,isHealthy) %>%  
 mutate(relInt = totIntang/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,-noi,-personnel,-debts, -deprec,-revenues,-valCost,-totIntang)

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

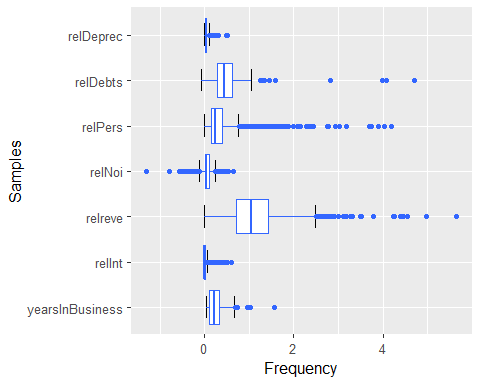
Question: a general overview of the selected variables.  
Design: pairs plot = matrix of scatterplots that lets you understand the pairwise relationship between different variables in a dataset

# require(GGally)  
# data %>% ggpairs()  
  
#plot(data)

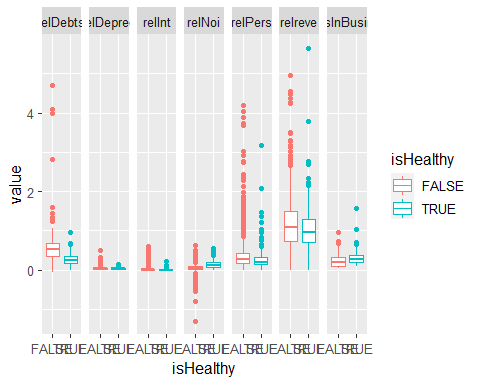
Questions: are the variable of the same order of magnitude?

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

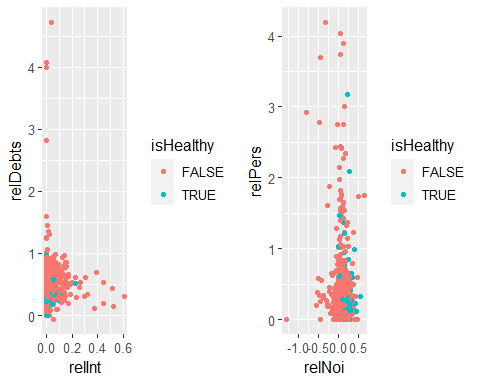
 Questions: are the variables useful to predict the given label? are the distributions different acocrding to the label? Answet: bowplot by laber

data %>% pivot\_longer(cols=!isHealthy) %>% ggplot(aes(x=isHealthy, y=value, color = isHealthy)) + geom\_boxplot() + facet\_grid(.~name, 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.

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\_boxplot() + facet\_grid(.~name, scales="free")   
# data %>% pivot\_longer(cols=!isHealthy) %>% ggplot(aes(x=isHealthy, y=value, color=isHealthy)) + geom\_violin() + facet\_grid(.~name, scales="free")

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.

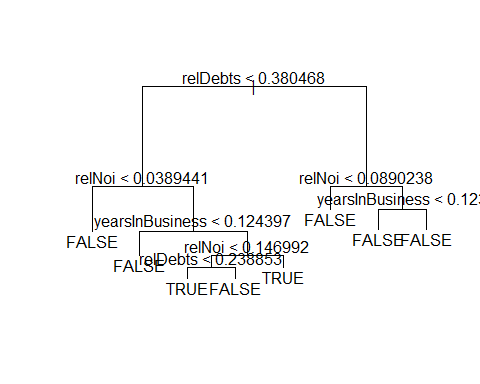
require(tree)

## 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, mincut = 50 , minsize = 100)   
plot(model)  
text(model)



print(model)

## node), split, n, deviance, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 988 995.40 FALSE ( 0.79757 0.20243 )   
## 2) relDebts < 0.380468 400 542.20 FALSE ( 0.58750 0.41250 )   
## 4) relNoi < 0.0389441 143 108.20 FALSE ( 0.87413 0.12587 ) \*  
## 5) relNoi > 0.0389441 257 350.90 TRUE ( 0.42802 0.57198 )   
## 10) yearsInBusiness < 0.124397 53 48.29 FALSE ( 0.83019 0.16981 ) \*  
## 11) yearsInBusiness > 0.124397 204 256.80 TRUE ( 0.32353 0.67647 )   
## 22) relNoi < 0.146992 140 190.60 TRUE ( 0.42143 0.57857 )   
## 44) relDebts < 0.238853 71 78.16 TRUE ( 0.23944 0.76056 ) \*  
## 45) relDebts > 0.238853 69 92.37 FALSE ( 0.60870 0.39130 ) \*  
## 23) relNoi > 0.146992 64 44.19 TRUE ( 0.10938 0.89062 ) \*  
## 3) relDebts > 0.380468 588 265.40 FALSE ( 0.94048 0.05952 )   
## 6) relNoi < 0.0890238 444 87.99 FALSE ( 0.97973 0.02027 ) \*  
## 7) relNoi > 0.0890238 144 136.00 FALSE ( 0.81944 0.18056 )   
## 14) yearsInBusiness < 0.123 68 0.00 FALSE ( 1.00000 0.00000 ) \*  
## 15) yearsInBusiness > 0.123 76 97.65 FALSE ( 0.65789 0.34211 ) \*

## 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 , 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 , 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 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 value, it is not using it

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

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

## [1] 0.1414141

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: 0.129"

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

## [1] "Error rate on test data: 0.141"

# classification.error <- length(which(predicted.test!=data$isHealthy[-indexes.learning]))/length(predicted.test)  
# 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 146 15  
## TRUE 13 24

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] 0.8585859

and the error rate can be computed as:

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

## [1] 0.141

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] 0.2070707

# require("rpart")  
# require("rpart.plot")  
# print(computeErrorRate("isHealthy", data, rpart))

## 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] 0.1797468

err.rate.training

## [1] 0.2323232

### 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, minsize = 2 )[1]})  
# error2 = ratio %>% map\_dbl(function(r){computeErrorRate2("isHealthy", data, tree, r, mindev = 0, minsize = 2 )[2]})  
# 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 = 'red'))+ylim(0,1)

### 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)) %>%   
 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 = mindevs, minsize = 1 )[1]})  
# error2 = mindevs %>% map\_dbl(function(r){computeErrorRate2("isHealthy", data, tree, 0.8, mindev = mindevs, minsize = 1 )[2]})  
#   
# 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=error2, color = 'red'))+ylim(0,1)

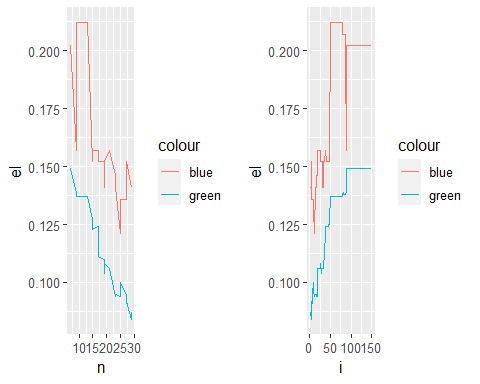
p.learn <- .8  
indexes.learning = sample(c(1:nrow(data)))[1:(nrow(data)\*p.learn)]  
  
df <- tibble(i=0, n=0, el=0, et=0) %>% head(0)  
  
# i = mincut The minimum number of observations to include in either child node. This is a weighted quantity; the observational weights are used to compute the ‘number’. The default is 5. we start from a very high value i.e. short trees and develop the tree up to mincut = 1  
  
for(i in seq(150,1,-1)){  
 model <- tree(isHealthy~.,data, subset = indexes.learning, mincut = i , minsize = i\*2) #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.test,3))  
}  
  
plot1 <- df %>% filter(i>1)%>%ggplot()+  
 geom\_line(aes(x = n, y=el, color='green', label = "learning data"))+   
 geom\_line(aes(x = n, y=et, color='blue', label = "test data"))

## Warning: Ignoring unknown aesthetics: label  
  
## Warning: Ignoring unknown aesthetics: label

plot2 <- df %>% filter(i>1)%>%ggplot()+  
 geom\_line(aes(x = i, y=el, color='green', label = "learning data"))+   
 geom\_line(aes(x = i, y=et, color='blue', label = "test data"))

## Warning: Ignoring unknown aesthetics: label  
  
## Warning: Ignoring unknown aesthetics: label

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 test data = ", mins$et[1]))

## [1] "minimum error for i = 12 tree composed of 25 nodes, error on test data = 0.121"

## 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.test,3))  
   
 # 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.

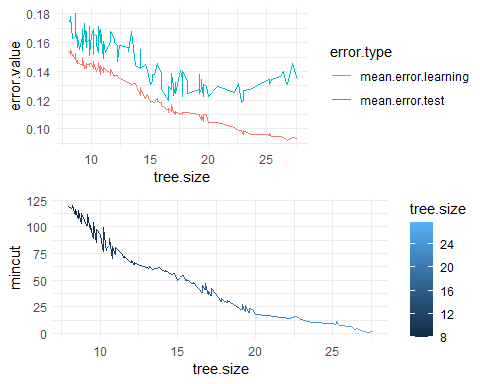
1. split learning data (X and y) in k equal slices (each of n k observations)
2. for each split (i.e., each i ∈ {1, . . . , k} ) 2.1 learn on all but k-th slice 2.2 compute classification error on unseen k-th slice
3. average the k classification errors

## mean error estimation using stratified k-fold CV

compute.Kfold.error <- function(data, errors, k\_folds=10, mincut=10,minsize=20) {  
 data\_f <- data %>%   
 group\_by(isHealthy) %>%  
 sample\_frac(1) %>%  
 mutate(fold=rep(1:k\_folds, length.out=n())) %>%  
 ungroup  
   
   
 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 , minsize = minsize)   
 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) %>% round(3)  
 err.rate.test <- errors.test %>% filter(err == TRUE) %>% nrow()/length(predicted.test) %>% round(3)  
   
 #save results  
 errors <- errors %>%   
 add\_row(f = i , err.learn = err.rate.learning, err.test = err.rate.test, n = number.of.nodes, minsize = minsize, mincut = mincut)  
  
 }  
 return(errors)  
}  
  
  
   
errors = tibble(f = NA, err.learn = NA, err.test = NA, n = NA, mincut = NA, minsize = NA) %>% head(0)  
  
k\_folds=10  
for (mc in seq(120,1,-1)){  
 errors <- compute.Kfold.error(data, errors, k\_folds=k\_folds, mincut=mc, minsize=mc\*2)  
}  
  
# average the k classification errors  
mean.errors <- errors %>% group\_by( mincut, minsize)%>%   
 summarize(mL = mean(err.learn), mT = mean(err.test), n = mean(n) )

## `summarise()` has grouped output by 'mincut'. You can override using the `.groups` argument.

#plot using a tidy data structure to represent data and legend: gather (or pivot\_longer) to bring all the data into one column and then use group and colour mappings in the aesthetic.  
  
data.to.plot <- mean.errors %>%   
 mutate(mean.error.learning = mL) %>%  
 mutate(mean.error.test = mT) %>%  
 mutate(tree.size = n)%>%   
 gather(error.type, error.value, c( mean.error.test, mean.error.learning)) %>%  
 select(-mT, -mL)  
  
   
   
plot1 <- data.to.plot %>% ggplot( ) + theme\_minimal()+  
 geom\_line(aes(x=tree.size, y = error.value, group = error.type, colour = error.type))  
  
plot2 <- data.to.plot %>% ggplot( ) + theme\_minimal()+  
 geom\_line(aes(x=tree.size, y = mincut, group = error.type, colour = tree.size))  
  
ggarrange(plot1, plot2, ncol = 1, nrow = 2)



# data.to.plot1 <- mean.errors %>%   
# mutate(mean.error.learning = mL) %>%  
# mutate(mean.error.test = mT) %>%  
# mutate(tree.size = n)%>%   
# gather(error.type, error.value, c( mean.error.test, mean.error.learning, tree.size)) %>%  
# select(-mT, -mL) %>%  
# mutate(complexity = max(errors$mincut)-mincut)   
#   
#   
# data.to.plot1 %>% ggplot( ) + theme\_minimal()+  
# geom\_line(aes(x=complexity, y = error.value, group = error.type, colour = error.type))+  
# facet\_wrap(~error.type, scales = "free\_y", ncol = 1)

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

## # A tibble: 1 x 5  
## # Groups: mincut [1]  
## mincut minsize mL mT n  
## <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 15 30 0.0991 0.118 22.8

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