Statistical Programming Languages

Valeryia Mosinzova
Michail Psarakis
Thomas Siskos

Ladislaus von Bortkiewicz Chair of Statistics
Humboldt–Universität zu Berlin
http://lvb.wiwi.hu-berlin.de

Contents

Data Preparation

```r
library("dplyr")
```

Due to missing insolvencies in 1996 and missing data from 2003 onwards we choose data from 1997-2002

```r
data1 = filter(data, JAHR>= 1997 & JAHR<=2002)

#Extract the industry class of companies
data1$Ind.Klasse = substring(data1$VAR26, 1, 2)
```

Data Preparation

As we are only interested in companies with high percentage in the industry composition we choose only companies belonging to the following sectors (according to German Classification of Economic Activities Standards (WZ 1993)):

1. Manufacturing (Man)
2. Wholesale and Retail (WaR)
3. Construction (Con)
4. Real Estate (RE)

Data Preparation

```
Man = filter (data1, Ind.Klasse %in% as.character(15:37))
Man$Ind.Klasse = "Man"

WaR = filter (data1, Ind.Klasse %in% as.character(50:52))
WaR$Ind.Klasse = "WaR"

Con = filter (data1, Ind.Klasse == "45")
Con$Ind.Klasse = "Con"

RE = filter (data1, Ind.Klasse %in% as.character(70:74))
RE$Ind.Klasse = "RE"
```

Data Preparation

Remove data and data1 and bind the above datasets to get one dataset containing only companies of interest:

```
rm(data, data1)

data = rbind(Man, WaR, Con, RE)
```

Furthermore we choose only companies whose total assets (`VAR26`) are in the range $10^5 - 10^8$.

```
data = data[data$VAR6 >= 10^5 & data$VAR6<= 10^8,]
```

Data Preparation

Eliminate observations with 0 value for the following variables used as denominators in calculation of financial ratios to be used in classification:

- total assets (`VAR6`)
- total sales(`VAR16`)
- cash (`VAR1`)
- inventories (`VAR2`)
- current liabilities (`VAR12`)
- total liabilities (`VAR12 + VAR13`)
- *total assets* − *intangible assets* − *cash* − *lands and buildings* (`VAR6 - VAR5 - VAR1 - VAR8`)
- interest expenses (`VAR19`)

Data Preparation

```
data_clean = data %>% filter(VAR6 != 0,
                             VAR16 != 0,
                             VAR1 != 0,
                             VAR2 != 0,
                             VAR12 != 0,
                             VAR12 + VAR13 != 0,
                             VAR6 - VAR5 - VAR1 -
                               VAR8 != 0,
                             VAR19 != 0)
```

Show table with number of solvent/insolvent companies:

```
table(data_clean$JAHR, factor(data_clean$T2,
```

Data Preparation: Results

Year	Solvent	Insolvent
1997	1084	126
1998	1175	114
1999	1277	147
2000	1592	135
2001	1920	132
2002	2543	129

Data Preparation

Add columns with financial ratios:

```
test_data = data_clean %>%
  mutate(
    # NetIncome/TotalAssets
    x1 = VAR22/VAR6,
    # NetIncome/TotalSales
    x2 = VAR22/VAR16,
    # OperatingIncome/TotalAssets
    x3 = VAR21/VAR6,
    # OperatingIncome/TotalSales
    x4 = VAR21/VAR16,
    # EBIT/TotalAssets
    x5 = VAR20/VAR6,
```

...

Data Preparation

...

```
# (EBIT+AmortizationDepreciation)/TotalAssets
x6 = (VAR20+VAR18)/VAR6,
# EBIT/TotalSales
x7 = VAR20/VAR16,
# OwnFunds/TotalAssets
x8 = VAR9/VAR6,
# (OwnFunds-IntangibleAssets)/
# (TotalAssets-IntangibleAssets-Cash-LandsAndBuildings)
x9 = (VAR9-VAR5)/(VAR6-VAR5-VAR1-VAR8),
# CurrentLiabilities/TotalAssets
x10 = VAR12/VAR6,
# (CurrentLiabilities-Cash)/TotalAssets
x11 = (VAR12-VAR1)/VAR6,
```

Data Preparation

...

```
# TotalLiabilities/TotalAssets
x12 = (VAR12+VAR13)/VAR6,
# Debt/TotalAssets
x13 = VAR14/VAR6,
# EBIT/InterestExpense
x14 = VAR20/VAR19,
# Cash/TotalAssets
x15 = VAR1/VAR6,
# Cash/CurrentLiabilities
x16 = VAR1/VAR12,
# QuickAssets(=Cur.Assets-Invent,)/CurrentLiabilities
x17 = (VAR3-VAR2)/VAR12,
```

...

Data Preparation

...

```
# CurrentAssets/CurrentLiabilities
x18 = VAR3/VAR12,
# WorkingCapital(=Cur.Assets-Cur.Liab.)/TotalAssets
x19 = (VAR3-VAR12)/VAR6,
# CurrentLiabilities/TotalLiabilities
x20 = VAR12/(VAR12+VAR13),
# TotalAssets/TotalSales
x21 = VAR6/VAR16,
# Inventories/TotalSales
x22 = VAR2/VAR16,
# AccountsReceivable/TotalSales
x23 = VAR7/VAR16,
```

...

Data Preparation

...

```
    # AccountsPayable/TotalSales
    x24 = VAR15/VAR16,
    # Log(TotalAssets)
    x25 = log(VAR6),
    # IncreaseDecreaseInventories/Inventories
    x26 = VAR23/VAR2,
    # IncreaseDecreaseLiabilities/TotalLiabilities
    x27 = VAR24/(VAR12+VAR13),
    # IncreaseDecreaseCashFlow/Cash)
    x28 = VAR25/VAR1)
```

Data Preparation

Prepare dataframe containing relative variables: `ID`, `JAHR` and the financial ratios will be used for classification.

```
test_data_rel = select(test_data,
                       ID, T2, JAHR, x1:x28)

table(factor(test_data_rel$T2,
```

Result:

```
  Solvent Insolvent
     9591       783
```

Which is almost the same result as in Zhang, Hardle (2010)

Evaluate Predictions

Intended Outcome: Function that takes labels and predictions as inputs and returns the following Measures, Plots and Tables:

- Confusion Matrix
  - true positives
  - true negatives
  - false positives
  - false negatives
- ROC + AUC
- Accuracy
- Precision
- Sensitivity

Evaluate Predictions

Get predictions from fitted probabilities:

```
get_prediction = function(fitted_probs, threshold){
  predictions = ifelse(fitted_probs > threshold,
                       1, 0)
  return(predictions)
}
```

Evaluate Predictions

Get TP, TN, FP, FN via a contingency table Set factor labels to avoid pitfalls if all predictions are the same.

```
evaluate_predictions = function(labels, predictions, verbose = FALSE){
  ct = table(factor(x = labels, levels = c(0, 1)),
             factor(x = predictions, levels = c(0,1)))

  if(any(dim(ct) != 2)) stop("Labels or Predictions contain more than 2 classes or both consist of only one class.")

  TN = ct[1, 1]
  TP = ct[2, 2]
  FP = ct[1, 2]
  FN = ct[2, 1]
```

Evaluate Predictions

Calculate Measures:

...

```
  reports = list(
    sensitivity = TP / (TP + FN),
    specificity = TN / (FP + TN),
    precision = TP / (TP + FP),
    accuracy = (TP + TN) / sum(ct))

  if(verbose) print(data.frame(reports), digits = 3)
  return(reports)
}
```

Evaluate Predictions

```r
evaluate_model = function(fitted_probs, labels){
  threshold_list = seq(from = 0, to = 1, by = 0.0001)
  pred_list = lapply(threshold_list,
                     get_prediction,
                     fitted_probs = fitted_probs)
  report_list = lapply(pred_list,
                       evaluate_predictions,
                       labels = labels)

  reports = unlist(report_list)
```

Evaluate Predictions

Calculate Values for ROC:

```
# Calculate ROC:
sensitivities = reports[names(reports) == "sensitivity"]
specificities = reports[names(reports) == "specificity"]
```

Calculate AUC:

Approximate area by by two sets of rectangles. One set systematically overestimates the area, the other systematically underestimates the area. Therefore take the average of both.

```
get_auc = function(x, y){
  abs(sum(diff(x) * (head(y, -1) + tail(y, -1)))/2)}
auc = get_auc(1-specificities, sensitivities)
```

Evaluate Predictions

Choose optimal threshold, specificity, sensitivity, predictions and confusion matrix:

```
opt_ind = which.max(sensitivities + specificities - 1)
opt_sensitivity = sensitivities[opt_ind]
opt_specificity = specificities[opt_ind]
opt_threshold = seq(from = 0, to = 1, by = 0.0001)[opt_ind]
opt_predictions = get_prediction(fitted_probs = fitted_probs,
                                 threshold = opt_threshold)
ct = table(factor(x = labels, levels = c(0, 1)),
```

Evaluate Predictions

Plot ROC-Curve

```
plot(x = 1 - specificities,
     y = sensitivities,
     main = "ROC-Curve",
     xlab = "False Positive Rate (1 - specificity)",
     ylab = "True Positive Rate (sensitivity)",
     xlim = c(0, 1),
     ylim = c(0, 1),
     asp = TRUE,
     type = "l")
abline(c(0, 0), c(1,1), col = "grey")
```

Evaluate Predictions

Return the Values:

```
  return(list(sensitivity = opt_sensitivity,
              specificity = opt_specificity,
              threshold = opt_threshold,
              predictions = opt_predictions,
              auc = auc,
              contingency_table = ct))
}
```

Logit

Create a subset train- and testsets:

```
training=subset(test_data_ratio,test_data_ratio$X.JAHR.<2000)
validierung=subset(test_data_ratio,test_data_ratio$X.JAHR.>=2000)
```

Logit

Set up variables:

```
validierung_logit=validierung
names(validierung_logit)[2]="status"
validierung_logit=validierung_logit[,-1]
validierung_logit=validierung_logit[,-2]

#Bootstrapping:
training_insolvent=training[training$X.T2.==1,]
training_solvent_full=training[training$X.T2.==0,]

conf_mat_true_logit=0
conf_mat_sum_logit=0
```

Logit

Train the model:

```
for (i in 1:30){
  randum_numb=round(runif(nrow(training_insolvent),
    min = 1, max = nrow(training[training$X.T2
    .==0,])))
  training_solvent=training_solvent_full[randum_numb
    ,]
  training_complete=rbind(training_insolvent,
    training_solvent)
  training_complete=rbind(training_insolvent,
    training_solvent)
  training_complete=training_complete[,-1]
  training_complete=training_complete[,-2]
```

Logit

Train the model:

```
## training the model:
names(training_complete)[1]="status"
glm_mod=train(as.factor(status)~.,data=training_complete, method="glm", family="binomial")
glm_pred = predict(glm_mod, newdata=validierung_logit)
conf_mat_logit=table(glm_pred, validierung_logit$status)
conf_mat_true_logit=sum(conf_mat_true_logit,conf_mat_logit[1,1],conf_mat_logit[2,2],na.rm=T)
conf_mat_sum_logit=sum(conf_mat_sum_logit+sum(conf_mat_logit,na.rm=T),na.rm=T)
```

Logit

Accuracy:

```
AR_logit=conf_mat_true_logit/conf_mat_sum_logit
#accuracy rate:~71 %
```

Build the Cart Model

Create a subset to train and evaluate the model:

```
training=subset(test_data_ratio,test_data_ratio$X.JAHR.<2000)
validierung=subset(test_data_ratio,test_data_ratio$X.JAHR.>=2000)
```

Cart

Since insolvent firms are underrepresented we need to oversample them in to balance the dataset.

```
#create a subset from training dataset, which only
  includes insolvent firms
training_insolvent=training[training$X.T2.==1,]
#create a subset, which only includes solvent firms
training_solvent_full=training[training$X.T2.==0,]
```

Create a matrix for performance evaluation of binary classification, which will be filled with true positive/negative rates:

```
conf_mat_true_cart=0
```

Create a matrix which will store the sum of all prediction outcomes:

```
conf_mat_sum_cart=0
```

Cart

For each bootstrap sample use all insolvent firms in the training set and randomly sample the same number of solvent firms from the training set.

```
randum_numb=round(runif(nrow(training_insolvent), min = 1, max = nrow(training[training$X.T2.==0,])))

training_solvent=training_solvent_full[randum_numb,]
training_complete=rbind(training_insolvent, training_solvent)
training_complete=training_complete[,-1]
training_complete=training_complete[,-2]
```

...

Cart

```
  # training the model:
  names(training_complete)[1]="status"
  modfit=train(status~.,method="rpart",data=training_complete)
# Validation/Test phase
  pred.cart=predict(modfit,newdata=validierung_cart)
# Accuracy and misclassication
  conf_mat_cart=table(pred.cart,validierung_cart$status)
# fill the matrix with true positive and true negative rates
  conf_mat_true_cart=sum(conf_mat_true_cart,conf_mat_cart[1,1],conf_mat_cart[2,2],na.rm=T)
# store total sum of predicted outcomes
  conf_mat_sum_cart=sum(conf_mat_sum_cart+sum(conf_mat_cart,na.rm=T),na.rm=T)}
```

Cart

Calculate accuracy rate

```
AR_cart=conf_mat_true_cart/conf_mat_sum_cart
#AR_cart: ~68%
```

Random Forest: Training the model

```
names(training_complete)[1]="status"
mod_forest_I=randomForest(as.factor(status)~.,
                          data=training_complete,
                          importance=T,
                          ntree = 2000,
                          maxnodes= 100,
                          norm.votes = F)
```

Validation- and Test-Phase

```
pred_rf=as.data.frame(predict(mod_forest_I, validierung_rf))
conf_mat_rf=table((as.numeric(unlist(pred_rf))-1), validierung_rf$status)
conf_mat_true_rf=sum(conf_mat_true_rf,conf_mat_rf[1,1],conf_mat_rf[2,2],na.rm=T)
conf_mat_sum_rf=sum(conf_mat_sum_rf+sum(conf_mat_rf,na.rm=T),na.rm=T)
```

Calculate accuracy rate

```
conf_mat_true_rf=sum(conf_mat_true_rf,conf_mat_rf[1,1],conf_mat_rf[2,2],na.rm=T)
```

Theory

At starting point a a (possibly multivariate) function $f(x)$ decays fastest if one follows the negative gradient of f evaluated at a, thus we can update a by:

$$a_{n+1} = a + \eta \cdot \nabla f(a) \tag{1}$$

Implementation in R - Gradient

Approximate Gradient by taking finite differences:

```
get_gradient = function(x, d = n_pars,
                        objective = obj,
                        epsilon = epsilon_step){
  init = matrix(data = x, nrow = d, ncol = d, byrow = TRUE)
  steps = init + diag(x = epsilon, ncol = d, nrow = d)

  f_steps = apply(steps, 1, objective)
  f_comp =  apply(init, 1, objective)

  D = (f_steps - f_comp) / epsilon
  # Trim D to limit the gradient:
  D_trimmed = ifelse(abs(D) <= 100, abs(D), 100) * sign(D)
```

Choose Starting Point

Try different Points and chose the one that provides the lowest value:

```
a = matrix(data = runif(1000 * n_pars,
                        min = -100,
                        max = 100),
           ncol = n_pars)
f_a = apply(a, 1, obj)
a = a[which.min(f_a), ]
gradient = get_gradient(a)
```

Update Starting Point *a*

```
while(i <= max_iter & any(abs(gradient) >=
  precision)){
   if(i %% report_freq == 0 & verbose) {
      cat("\nStep:\t\t", i,
            "\nx:\t\t", a,
            "\ngradient:\t", gradient,
            "\nlearn:\t", learn_rates[i],
            "\n------------------------------------")}

   i = i + 1
   a = a - learn_rates[i] * gradient
   gradient = get_gradient(a)
}
```

Example

Visualize the path of a one-dimensional Gradient-Descent:

Continuously Differentiable Objective Function:

$$f(x) = \frac{x^4}{10000} + 2 \qquad (2)$$