

Statistical Programming Languages

Valeryia Mosinzova

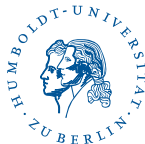
Michail Psarakis

Thomas Siskos

Ladislav von Bortkiewicz Chair of Statistics

Humboldt-Universität zu Berlin

<http://lvb.wiwi.hu-berlin.de>



Contents

Data Preparation

Evaluate Predictions

Cart

Logit

Random Forest

Gradient Descent



Data Preparation

```
1 library("dplyr")
```

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

```
1 data1 = filter(data, JAHR>= 1997 & JAHR<=2002)
2
3 #Extract the industry class of companies
4 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

```
1 Man = filter (data1, Ind.Klasse %in% as.character  
  (15:37))  
2 Man$Ind.Klasse = "Man"  
3  
4 WaR = filter (data1, Ind.Klasse %in% as.character  
  (50:52))  
5 WaR$Ind.Klasse = "WaR"  
6  
7 Con = filter (data1, Ind.Klasse == "45")  
8 Con$Ind.Klasse = "Con"  
9  
10 RE = filter (data1, Ind.Klasse %in% as.character  
  (70:74))  
11 RE$Ind.Klasse = "RE"
```



Data Preparation

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

```
1 rm(data, data1)
2
3 data = rbind(Man, WaR, Con, RE)
```

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

```
1 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

```
1 data_clean = data %>% filter(VAR6 != 0,  
2                             VAR16 != 0,  
3                             VAR1 != 0,  
4                             VAR2 != 0,  
5                             VAR12 != 0,  
6                             VAR12 + VAR13 != 0,  
7                             VAR6 - VAR5 - VAR1 -  
8                             VAR8 != 0,  
                             VAR19 != 0)
```

Show table with number of solvent/insolvent companies:

```
1 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:

```
1 test_data = data_clean %>%  
2   mutate(  
3     # NetIncome/TotalAssets  
4     x1 = VAR22/VAR6 ,  
5     # NetIncome/TotalSales  
6     x2 = VAR22/VAR16 ,  
7     # OperatingIncome/TotalAssets  
8     x3 = VAR21/VAR6 ,  
9     # OperatingIncome/TotalSales  
10    x4 = VAR21/VAR16 ,  
11    # EBIT/TotalAssets  
12    x5 = VAR20/VAR6 ,
```

...



Data Preparation

...

```
1      # (EBIT+AmortizationDepreciation)/TotalAssets
2      x6 = (VAR20+VAR18)/VAR6 ,
3      # EBIT/TotalSales
4      x7 = VAR20/VAR16 ,
5      # OwnFunds/TotalAssets
6      x8 = VAR9/VAR6 ,
7      # (OwnFunds-IntangibleAssets)/
8      # (TotalAssets-IntangibleAssets-Cash-
9      LandsAndBuildings)
10     x9 = (VAR9-VAR5)/(VAR6-VAR5-VAR1-VAR8) ,
11     # CurrentLiabilities/TotalAssets
12     x10 = VAR12/VAR6 ,
13     # (CurrentLiabilities-Cash)/TotalAssets
14     x11 = (VAR12-VAR1)/VAR6 ,
```



Data Preparation

...

```
1      # TotalLiabilities/TotalAssets
2      x12 = (VAR12+VAR13)/VAR6 ,
3      # Debt/TotalAssets
4      x13 = VAR14/VAR6 ,
5      # EBIT/InterestExpense
6      x14 = VAR20/VAR19 ,
7      # Cash/TotalAssets
8      x15 = VAR1/VAR6 ,
9      # Cash/CurrentLiabilities
10     x16 = VAR1/VAR12 ,
11     # QuickAssets(=Cur.Assets-Invent,)/
12     CurrentLiabilities
13     x17 = (VAR3-VAR2)/VAR12 ,
```

...



Data Preparation

...

```
1      # CurrentAssets/CurrentLiabilities
2      x18 = VAR3/VAR12 ,
3      # WorkingCapital (=Cur.Assets - Cur.Liab.) /
      TotalAssets
4      x19 = (VAR3 - VAR12)/VAR6 ,
5      # CurrentLiabilities/TotalLiabilities
6      x20 = VAR12/(VAR12+VAR13) ,
7      # TotalAssets/TotalSales
8      x21 = VAR6/VAR16 ,
9      # Inventories/TotalSales
10     x22 = VAR2/VAR16 ,
11     # AccountsReceivable/TotalSales
12     x23 = VAR7/VAR16 ,
```

...



Data Preparation

...

```
1      # AccountsPayable/TotalSales
2      x24 = VAR15/VAR16 ,
3      # Log(TotalAssets)
4      x25 = log(VAR6) ,
5      # IncreaseDecreaseInventories/Inventories
6      x26 = VAR23/VAR2 ,
7      # IncreaseDecreaseLiabilities/TotalLiabilities
8      x27 = VAR24/(VAR12+VAR13) ,
9      # IncreaseDecreaseCashFlow/Cash)
10     x28 = VAR25/VAR1)
```



Data Preparation

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

```
1 test_data_rel = select(test_data,  
2                        ID, T2, JAHR, x1:x28)  
3  
4 table(factor(test_data_rel$T2,
```

Result:

	Solvent	Insolvent
1		
2	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:

```
1 get_prediction = function(fitted_probs, threshold){  
2   predictions = ifelse(fitted_probs > threshold,  
3                       1, 0)  
4   return(predictions)  
5 }
```



Evaluate Predictions

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

```
1 evaluate_predictions = function(labels, predictions,
2   verbose = FALSE){
3   ct = table(factor(x = labels, levels = c(0, 1)),
4             factor(x = predictions, levels = c(0,1)
5               ))
6
7   if(any(dim(ct) != 2)) stop("Labels or Predictions
8     contain more than 2 classes or both consist of
9     only one class.")
10
11   TN = ct[1, 1]
12   TP = ct[2, 2]
13   FP = ct[1, 2]
14   FN = ct[2, 1]
```



Evaluate Predictions

Calculate Measures:

...

```
1  reports = list(  
2      sensitivity = TP / (TP + FN),  
3      specificity = TN / (FP + TN),  
4      precision = TP / (TP + FP),  
5      accuracy = (TP + TN) / sum(ct))  
6  
7  if(verbose) print(data.frame(reports), digits = 3)  
8  return(reports)  
9 }
```



Evaluate Predictions

```
1 evaluate_model = function(fitted_probs, labels){  
2   threshold_list = seq(from = 0, to = 1, by =  
3     0.0001)  
4   pred_list = lapply(threshold_list,  
5     get_prediction,  
6     fitted_probs = fitted_probs)  
7   report_list = lapply(pred_list,  
8     evaluate_predictions,  
9     labels = labels)  
10  reports = unlist(report_list)
```



Evaluate Predictions

Calculate Values for ROC:

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

Calculate AUC:

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

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



Evaluate Predictions

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

```
1  opt_ind = which.max(sensitivities + specificities  
2    - 1)  
3  opt_sensitivity = sensitivities[opt_ind]  
4  opt_specificity = specificities[opt_ind]  
5  opt_threshold = seq(from = 0, to = 1, by = 0.0001)  
6    [opt_ind]  
7  opt_predictions = get_prediction(fitted_probs =  
    fitted_probs,  
                                   threshold = opt_  
                                   threshold)  
8  ct = table(factor(x = labels, levels = c(0, 1)),
```



Evaluate Predictions

Plot ROC-Curve

```
1  plot(x = 1 - specificities ,
2      y = sensitivities ,
3      main = "ROC-Curve",
4      xlab = "False Positive Rate (1 - specificity)
5          ",
6      ylab = "True Positive Rate (sensitivity)",
7      xlim = c(0, 1),
8      ylim = c(0, 1),
9      asp = TRUE,
10     type = "l")
11  abline(c(0, 0), c(1,1), col = "grey")
```



Evaluate Predictions

Return the Values:

```
1  return(list(sensitivity = opt_sensitivity,  
2             specificity = opt_specificity,  
3             threshold = opt_threshold,  
4             predictions = opt_predictions,  
5             auc = auc,  
6             contingency_table = ct))  
7  }
```



Build the Cart Model

Create a subset to train and evaluate the model:

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



Cart

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

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



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

```
1 conf_mat_true_cart=0
```

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

```
1 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.

```
1  random_numb=round(runif(nrow(training_insolvent),  
    min = 1, max = nrow(training[training$X.T2  
    .==0,])))  
2  
3  training_solvent=training_solvent_full[random_numb  
    ,]  
4  training_complete=rbind(training_insolvent,  
    training_solvent)  
5  training_complete=training_complete[, -1]  
6  training_complete=training_complete[, -2]
```

...



Cart

```
1  # training the model:
2  names(training_complete)[1]="status"
3  modfit=train(status~.,method="rpart",data=training
   _complete)
4  # Validation/Test phase
5  pred.cart=predict(modfit,newdata=validierung_cart)
6  # Accuracy and misclassification
7  conf_mat_cart=table(pred.cart,validierung_cart$
   status)
8  # fill the matrix with true positive and true
   negative rates
9  conf_mat_true_cart=sum(conf_mat_true_cart,conf_mat
   _cart[1,1],conf_mat_cart[2,2],na.rm=T)
10 # store total sum of predicted outcomes
11 conf_mat_sum_cart=sum(conf_mat_sum_cart+sum(conf_
   mat_cart,na.rm=T),na.rm=T)}
```



Cart

Calculate accuracy rate

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



Logit

Create a subset train- and testsets:

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



Logit

Train the model:

```
1  names(training_complete)[1]="status"
2  glm_mod=train(as.factor(status)~.,data=training_
3  complete, method="glm", family="binomial")
4  glm_pred = predict(glm_mod, newdata=validierung_
5  logit)
6  conf_mat_logit=table(glm_pred, validierung_logit$
7  status)
8  conf_mat_true_logit=sum(conf_mat_true_logit,conf_
9  mat_logit[1,1],conf_mat_logit[2,2],na.rm=T)
10 conf_mat_sum_logit=sum(conf_mat_sum_logit+sum(conf
11 _mat_logit,na.rm=T),na.rm=T)
```



Random Forest: Training the model

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



Validation- and Test-Phase

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

Calculate accuracy rate

```
1  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:

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

where η describes the learning-rate.



Implementation in R - Gradient

Approximate Gradient by taking finite differences:

```
1  get_gradient = function(x, d = n_pars,  
2                          objective = obj,  
3                          epsilon = epsilon_step){  
4      init = matrix(data = x, nrow = d, ncol = d,  
5                    byrow = TRUE)  
6      steps = init + diag(x = epsilon, ncol = d, nrow  
7                    = d)  
8      f_steps = apply(steps, 1, objective)  
9      f_comp = apply(init, 1, objective)  
10     D = (f_steps - f_comp) / epsilon  
11     D_trimmed = ifelse(abs(D) <= 100, abs(D), 100) *  
12                   sign(D)  
13     return(D_trimmed)}
```



Choose Starting Point

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

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



Update Starting Point a

```
1  while(i <= max_iter & any(abs(gradient) >=
    precision)){
2      if(i %% report_freq == 0 & verbose) {
3          cat("\nStep:\t\t", i,
4              "\nx:\t\t", a,
5              "\ngradient:\t", gradient,
6              "\nlearn:\t", learn_rates[i],
7              "\n-----")
8
9          i = i + 1
10         a = a - learn_rates[i] * gradient
11         gradient = get_gradient(a)
12     }
```



Example

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

Continuously Differentiable
Objective Function:

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

