Credit Scoring: A Comparison of Logistic Regression, Decision Tree Models and Support vector machine

Yangxue Han

Abstract

The purpose of this project is to use the characteristics from the behavior data, to build a successful model for making on-going decisions about open accounts. The results obtained indicate that support vector machine has higher accuracy rates and therefore outperforms the other proposed classification methods. The estimated models can be used for credit rating prediction and can serve as a useful tool in the investment decision process.

1. Introduction

Credit scoring is a statistical method that is used to predict the probability that a loan applicant or existing borrower will default or become delinquent (Mester, 1997). Credit scoring problem is general used in many banks, retailers and consumer loans. Risk evaluation is the central problem for concession of credit. The models called credit scoring are used for the evaluation of risk of credit by classification of the applicant (Gouvêa, 2007).

Credit scoring models can be divided into two types: (i) credit scoring in targeting and application, which deals with new applicants, and (ii) credit scoring in managing existing accounts called behavior scoring. The purpose of this project is to use the characteristics from the behavior data, to build a successful model for making on-going decisions about open accounts. To this purpose we have a database of recent concessions of credits done by analysts. For this problem, we are provided with a real data. The credit scoring problem in this case study is the decision of granting loans to the customer to buy an asset.

This project is organized as follows: Firstly, we give a short overview of credit scoring. Section 2 explains the methods used (it is particularly the logistic regression, decision trees, and support vector machine apply them on the data set). The data set for the analysis is described in the third section. Summary of analyze with the comparison is given in Section 4 and 5. Conclusion in the Section 6.

2. Methodology

2.1 Logistic regression

Develop the logistic regression algorithm to determine what class a new input should fall into. Consider the case when the observations may be classified into one of two mutually exclusive categories (1 or 0). In the models of logistic regression, the dependent variable is, in general a binary variable (nominal or ordinal) and the independent variables may be categorical or continuous.

The form of the model is

$$\log(\frac{p}{1-p}) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k$$

where p is the probability that Y=1 and X1, X2,.. .,Xk are the independent variables (predictors). β0, β1, β2, ... βk are known as the regression coefficients, which have to be estimated from the data.

Logistic regression thus forms a predictor variable (log (p/(1-p)) which is a linear combination of the explanatory variables. The values of this predictor variable are then transformed into probabilities by a logistic function

Logistic regression predicts the probability of an event taking place, which can range from 0 and 1. Logistic Regression is the technique most often used in the market for the development of credit scoring models (ROSA, 2000; OHTOSHI, 2003).

2.2 Decision trees (TREES)

Classification and regression trees—(CART) is a nonparametric method to analyze categorical dependent variables as a function of metric and/or categorical explanatory variables.

Decision trees have three main parts: a root node, leaf nodes and branches. The entire sample is called root node, the subsamples are called leaf nodes and the branches are arrows connecting nodes, showing the flow from question to answer. The basic idea is to split the sample into two subsamples each of which contains only cases from one response category and then repeatedly split each of the subsamples. Each node typically has two or more nodes extending from it. The splitting rules are done according to some statistical criteria. Usually the tree is split till it is overfitted and afterward the tree is pruned back to get a more robust model.

2.3 Support Vector Machine (SVM)

A Support Vector Machine (SVM) is a supervised machine learning algorithm that can be employed for both classification and regression purposes. The goal of a support vector machine is to find the optimal separating hyperplane which maximizes the margin of the training data. SVM's do well in classifying nonlinear separable groups.

They do not require large training datasets and training converges to a unique global solution. These features make SVM's suitable for applications such as credit card fraud detection.

If the data is far from linear and the datasets are inseparable. To allow for this kernels are used to non-linearly map the input data to a high-dimensional space. The new mapping is then linearly separable. If every data point is mapped into high-

dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \varphi(\mathbf{x})$, the inner product becomes: $K(\mathbf{x_i},\mathbf{x_j}) = \varphi(\mathbf{x_i})^T\varphi(\mathbf{x_j})$. A *kernel function* is some function that corresponds to an inner product in some expanded feature space

3. Data description and sampling procedure

The total number contains 8830 observations and 14 variables. Table 1 gives a list of variables in the data while identifying the variable type and their description. These variables are considered as potential predictors and are hence included in the model development.

After analyzing the data from the summary of the data, we can see that there are missing values in the original dataset. The missing values are filled using the process of multiple imputations which is done using MICE library in R. The data set is replicated 5 times and equal weights are assigned to them.

Next, the method for feature selection provided by the caret R package is called Recursive Feature Elimination or RFE. The feature extraction will help in reducing the number of features so that the model doesn't overfit the data leading to bad results.

For our analysis, we have randomly divided the data sample into two subsamples About two thirds of the whole data set (2979 observations) builds the first subsample, the TRAIN sample. It will be used for model estimation. The second subsample, TEST, with 1467 observation will be used to get some overall procedure to compare methods and to check the predictive power of the models used. This will be based on misclassification rates.

Table 1. Dependent and Independent Variables to be Modeled

Variable Name	**Type**	**Description**
Opinion	Categorical	Response Variable with 2 modalities (1 – positive, 2 – negative).
Number.of.employment.years	Continuous	Variables for the years the person worked.
House.type	Categorical	Variable for the type of house where the person is living; 1-rented, 2-ownerwithdeed, 3-private, 4-ignore, 5-parents, 6-others.
term.in.months	Continuous	Variable indicating whether the person registers for house or not; 1-No, 2-Yes.
Age	Continuous	Variable indicating the age of person.

Marital.status	Categorical	Variable for the marital status of the person; 1-single, 2-married, 3-widower, 4-separated, 5-dicorced.
Registers	Categorical	Variable indicating the installment terms in months.
Type.of.employment	Categorical	Variable indicating the type of employment the person have; 1-Permanent, 2-Temporaray, 3-Self, 4-Others.
Expenses	Continuous	Variable indicating the expenses of the person.
income	Continuous	Variable indicating the total income of the person.
total.wealth	Continuous	Variable indicating the total wealth a person have.
mortgage.left	Continuous	Variable indicating the mortgage left to pay by the person.
amount.of.money.solicitated	Continuous	Variable indicating the total wealth of the person taking into account the mortgage left to pay.
price.of.good.to.buy	Continuous	Variable indicating the price of good the person wants to buy.

4. Model Selection and Development

The most important thing in developing model is to select right modeling algorithm(s).

4.1 Logistics regression

1）To select the best classifier, several options are used. The four options for the classifiers which were used are stated below.

Building the Logistic Model and checking the model summary

Status = Seniority + Home + Time + Age + Marital + Records +
Job + Expenses + Income + Assets + Debt + Amount + Price +
Finrat

2）Significance Level of the Variables

Those variables which have at least one star in the coefficients table are significant. Positive coefficient means higher the value of that variable, an increased risk of default, and vice versa. The significant variables are Debt, Amount, finrat, seniority, assets, records and Job. Since some of the variables are not significant, we will rebuild the logistic regression with only the significant variables.

3）Use the 'step' function to perform model selection by applying a backward elimination method based on AIC (Akaike Info. Criterion)

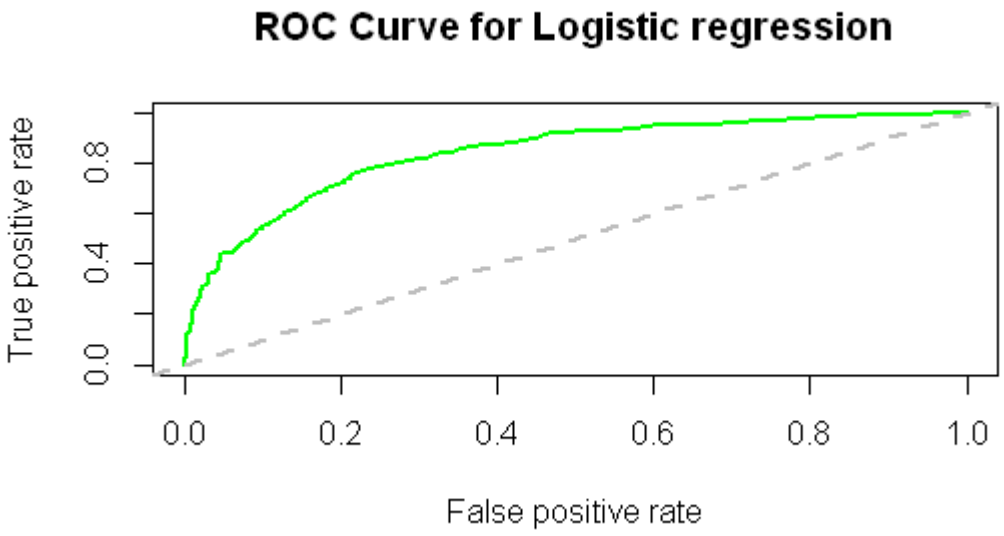
4）Revised Logistic Regression Model

Status ~ Debt + Amount + finrat + seniority + assets + records + Job

5） Measuring the accuracy of the simplified model to the Test dataset.

Accuracy%	Sensitivity%	Specificity%
80.85	83.78	70.31

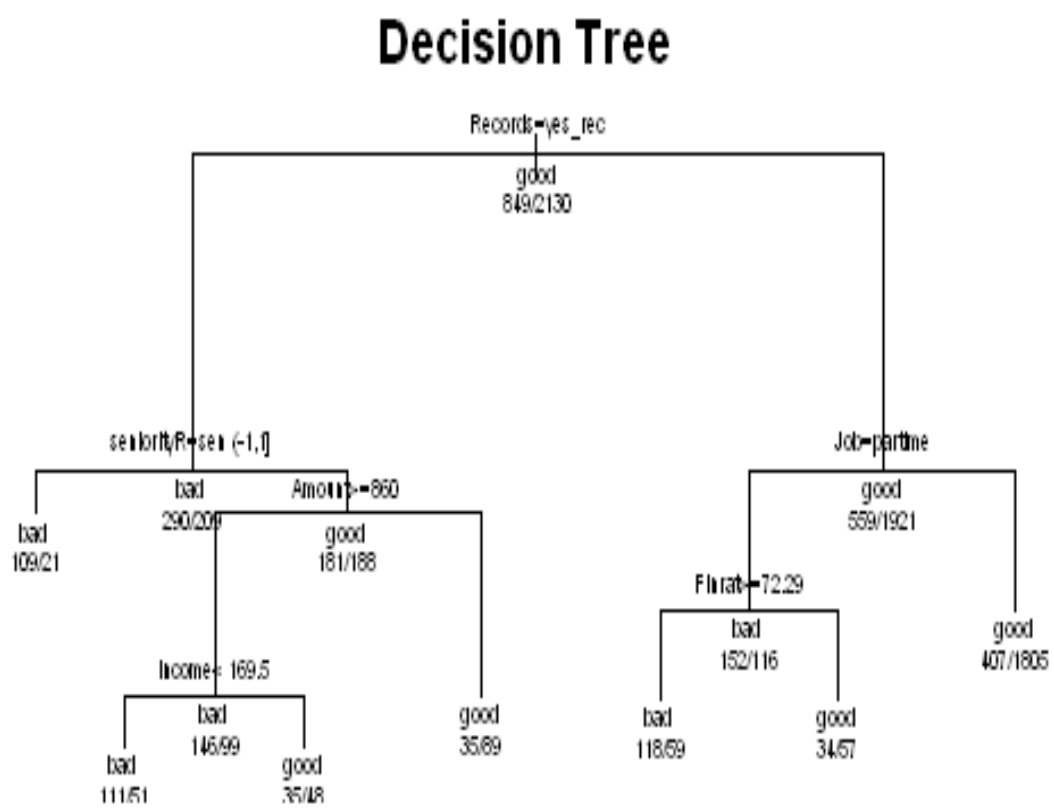
6）Plot the ROC curve by plotting the performance function which checks how our model is performing



4.2 Decision trees

1）Obtain a decision tree with the function 'rpart', using all the variables both continuous and categorical

2）Read a tree with a graphic



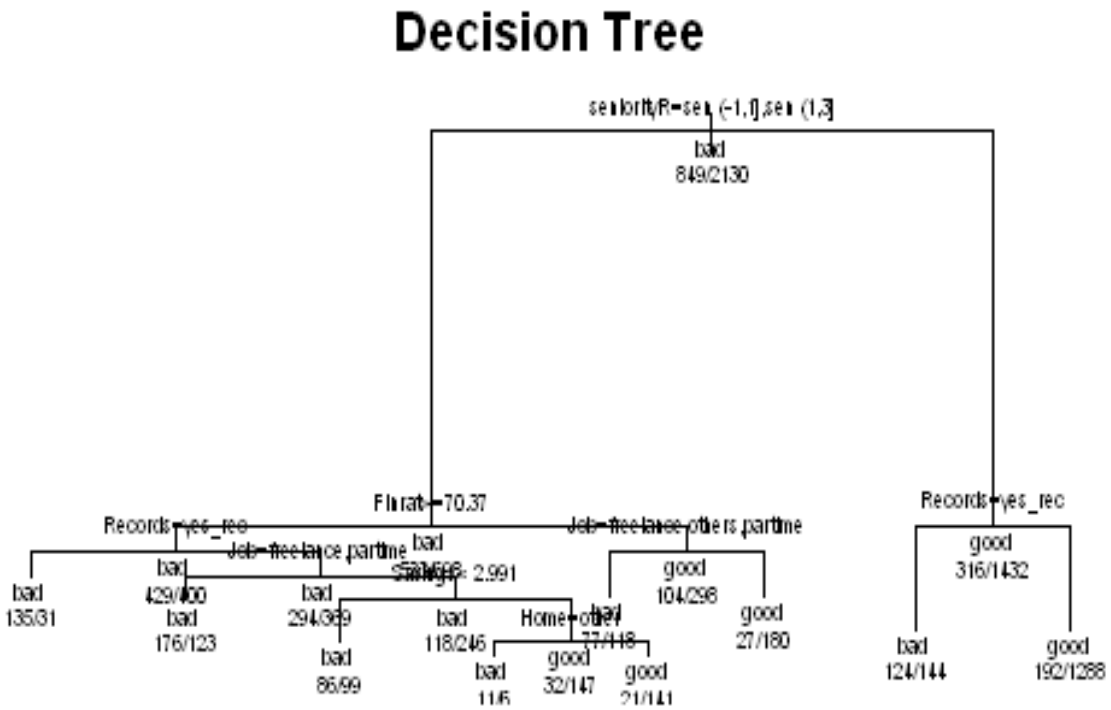
3）Prune the tree to ensure that it haven't over fitted the data.

In order to prune the data we need to find the appropriate complexity parameter. The ideal value of the complexity parameter in this case would then be the one that minimizes the X-error. Here we see that the value is 0.010000

```
Variables actually used in tree construction:
[1] Amount       Finrat       Income       Job          Records      seniorityR

Root node error: 849/2979 = 0.28499
n= 2979
          CP nsplit rel error    xerror       xstd
1 0.095406      0   1.00000 1.00000 0.029020
2 0.042403      1   0.90459 0.90459 0.028121
3 0.031802      2   0.86219 0.89282 0.028001
4 0.027091      4   0.79859 0.85748 0.027626
5 0.015312      5   0.77150 0.83157 0.027338
6 0.010000      6   0.75618 0.82097 0.027216
```

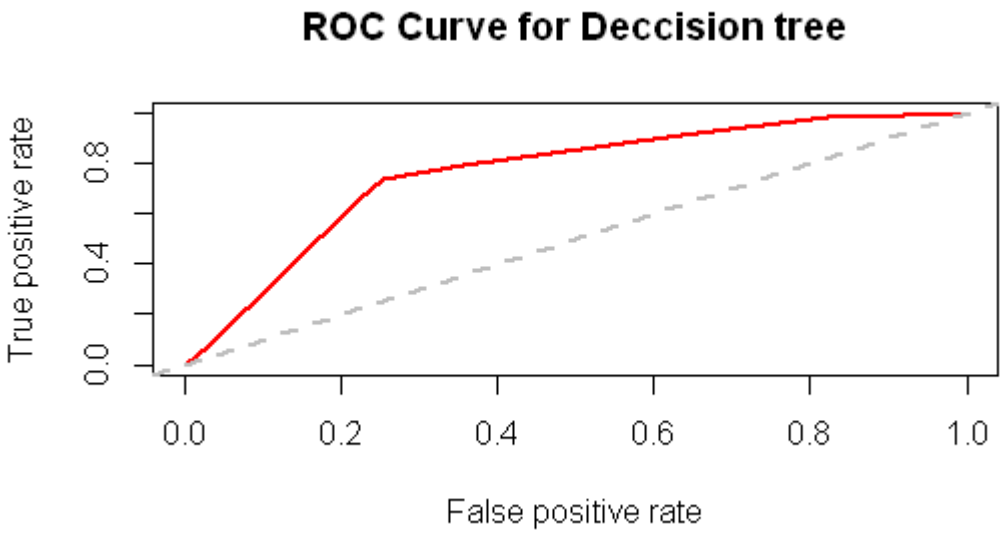
4）Now that we know that the 'best' tree has cp=0.01, we can plugin that information in the parameters. We'll apply it to the Test dataset to determine how effective it is.



5）Measuring the accuracy of the model

Accuracy	Sensitivity	Specificity
74.97	73.4	75.5

6）We will also plot the ROC curve for this tree.



4.3 support vector machines

1）This classifier uses support vector machines which is implemented in “ksvm” function of the R package. The most important parameter for the SVM is cost which is taken as 1 in this case.

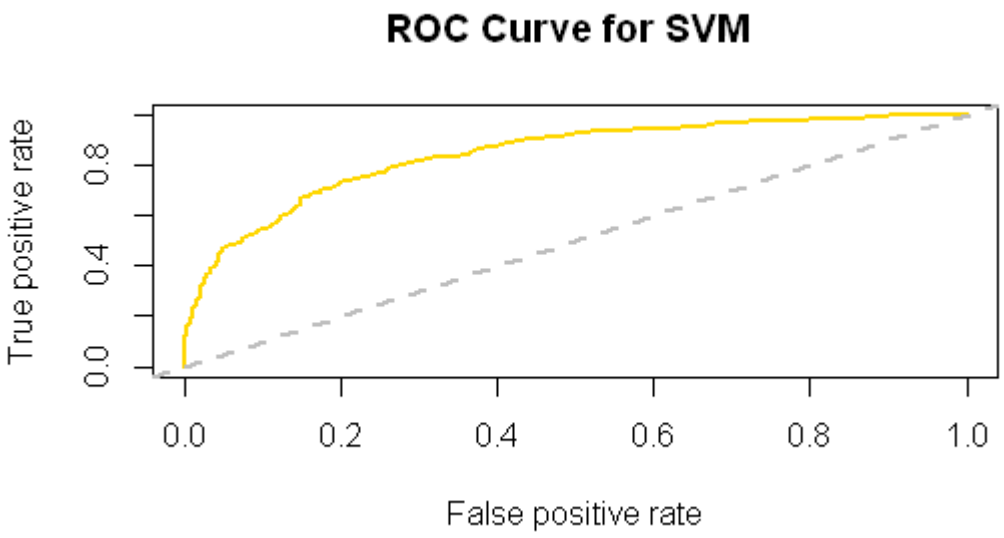
```
SVM-Type:  C-classification
SVM-Kernel:  radial
      cost:  1
     gamma:  0.01449275

Number of Support Vectors:  1619
```

2）Measuring the accuracy of the model

Accuracy	Sensitivity	Specificity
81.2	67.6	84.4

3）We can then plot the ROC curve by plotting the performance fuction which checks how our model is performing



5.Model Performance Comparison

To select best models out of set of experimental models we may compare the ROCs, Area Under the Receiver Operating Characteristic Curve (AUROC/AUC), KS, Gini etc.

1) An ROC curve is a commonly used way to visualize the performance of a binary classifier, meaning a classifier with two possible output classes. In a ROC curve

is the ratio of the true positive rate versus the false positive rate.

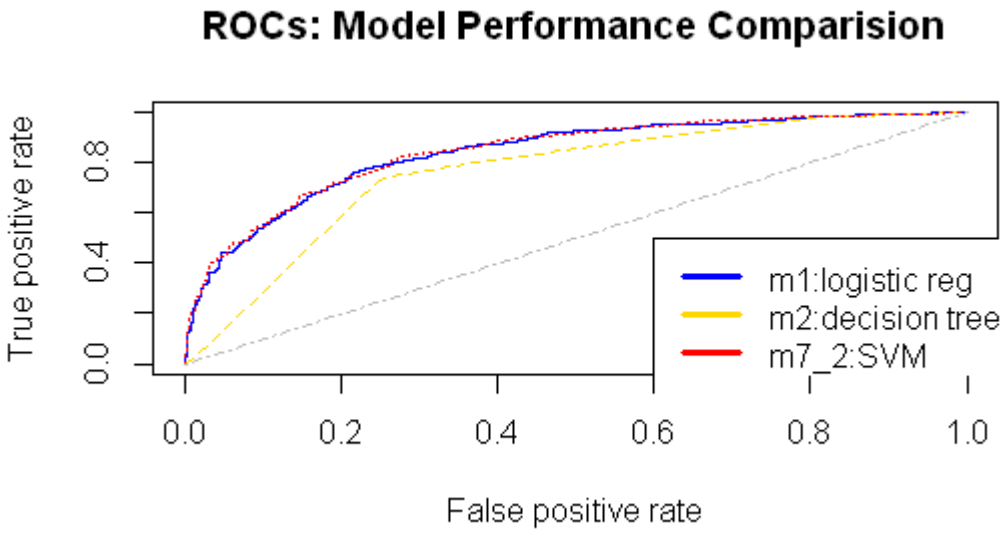
2) The ROC curve's performance is judged by the area under the curve formed by the ROC curve and the baseline. The more the area under the curve, better the model.

3) The Kolmogorov-Smirnov test (KS-test) tries to determine if two classes differ significantly. In our case we are trying to measure whether the model is able to classify "good" class from "bad" class very well. If they are completely separated (i.e. AUROCC=100%) then the value of KS will be 1 (100%) and if they are same then the value of KS will be 0 (0%). This means higher the value is better classification power.

4) Gini is another commonly used measure to evaluate goodness of fit for binary classification problems. Which has a straight relationship with AUROCC (Gini = 2*AUROCC - 1).

Model accuracy

Model	Accuracy	Sensitivity	Specificity
Logistic regression	80.85	83.78	70.31
Decision Tree	74.97	73.4	75.5
Support Vector Machine	81.2	67.6	84.4



Model Performance Comparison

Model	AUROC	KS	Gini
Support vector machine	84.51	54.68	69.02
Logistics regression	84.11	54.72	68.22
Decision Tree	76.45	48.66	52.9

6. Concluding Remarks

This paper discusses and illustrates the use of machine learning techniques in the construction and combination of credit scoring models.

According to the AUC: Area under the ROC curve and accuracy statics we can prefer Support vector machine model

2) According to the KS: we can prefer Logistics regression and Support vector machine

3) According to the Gini: we can prefer Support vector machine.

From the comparative study, we found that the Support vector machine performing better than other methods with given set-up. Depending on the problem definitions, data availability and given set-up one of the machine learning techniques would be best fit.

Appendix

```
setwd("G:/wangcai/New folder/CreditScoring-master")

# load package FactoMineR and ggplot2
require(FactoMineR)
require(ggplot2)

# read cleaned data set
dd = read.csv("G:/wangcai/New folder/CreditScoring-master/CleanCreditScoring.csv", header=TRUE, stringsAsFactors=TRUE)

#Apply logistic regression using all the variables
# ============================================================================

# let's keep 2/3 of the data for learning, and 1/3 for testing
n = nrow(dd)
learn = sample(1:n, size=round(0.67 * n))
nlearn = length(learn)
ntest = n - nlearn
train<-dd[learn,]
test<-dd[-learn,]

# "whole enchilada" logistic regression model
gl1 = glm(Status ~ ., data=dd[learn,], family=binomial)

# use the 'summary' function to obtain more details on the logistic model
# (pay attention to the signicance of the coefficients)
summary(gl1)

# let's use the 'anova' function to get a sequential analysis of
# variance of the model fit (ie importance of variables in the model)
anova(gl1)

# we can also use the 'step' function to perform model selection by
# applying a backward elimination method based on AIC (Akaike Info. Criterion)
step(gl1)
# ============================================================================
# Apply logistic regression after removing some variables
# ============================================================================

# new model
glf <- glm(formula = Status ~ Age + Income + Debt + Amount + Finrat +
```

```
                    seniorityR + expensesR + assetsR + priceR + savingsR + Home + Marital + Records +
                    Job, family = binomial, data = dd[learn, ])
# check summary
summary(glf)
# check coefficients
exp(glf$coefficients)
# re-expressed fitted values
glf$fitted.values = 1 - glf$fitted.values

# create vector for predictions
glfpred = rep(NA, length(glf$fitted.values))
glfpred[glf$fitted.values < 0.5] = 0
glfpred[glf$fitted.values >= 0.5] = 1

# how is the prediction? (confusion matrix)
table(dd$Status[learn], glfpred)
# error rate
error_rate.learn = 100*sum(diag(table(dd$Status[learn], glfpred))) / nlearn
error_rate.learn

# let's use the test data to get predictions
glft = predict(glf, newdata=dd[-learn,])
pt = 1 / (1 + exp(-glft))
pt = 1 - pt
# vector of predicted values
glftpred = rep(NA, length(pt))
glftpred[pt < 0.5] = 0
glftpred[pt >= 0.5] = 1

# confusion matrix
table(dd$Status[-learn], glftpred)
error_rate.test = 100*sum(diag(table(dd$Status[-learn], glftpred))) / ntest
error_rate.test
# ================================================================================
# Concentration curve
# ================================================================================

ac_tot = 100*(1:ntest) / ntest

pt.ord = order(pt, decreasing=T)

Status_test = dd$Status[-learn]
npos = table(Status_test)[2]
```

```
ac_pos_test = 100*cumsum(Status_test[pt.ord] == "good") / npos


plot(ac_tot, ac_pos_test, type="l", lwd=2,
     main="Concentration Curve")
lines(ac_tot, ac_tot, col="gray70", lwd=2)
# ================================================================================
# ROC Curve
# ================================================================================


nneg = ntest - npos


ac_neg_test = 100*cumsum(Status_test[pt.ord]=="bad") / nneg


plot(ac_neg_test, ac_pos_test, type="l", lwd=2, main="ROC Curve", col="red")
lines(ac_neg_test, ac_neg_test, col="grey70", lwd=2)


#load library#####################################################
library(ROCR)
#score test data set
test$score<-predict(glf,test,type="response")
pred<-prediction(test$score,test$Status)
perf <- performance(pred,"tpr","fpr")
#roc
plot(perf,col="green",lwd=2,main="ROC Curve for Logistic regression")
abline(a=0,b=1,lwd=2,lty=2,col="gray")


#KS, Gini & AUC m1
m1_KS <- round(max(attr(perf,'y.values')[[1]]-attr(perf,'x.values')[[1]])*100, 2)
m1_AUROC <- round(performance(pred, measure = "auc")@y.values[[1]]*100, 2)
m1_Gini <- (2*m1_AUROC - 100)
cat("AUROC: ",m1_AUROC,"\tKS: ", m1_KS, "\tGini:", m1_Gini, "\n")
require(ggplot2)
require(rpart)
dd = read.csv("G:/wangcai/New folder/CreditScoring-master/CleanCreditScoring.csv", header=TRUE, stringsAsFactors=TRUE)


# Let's
ct = rpart(Status ~ ., data=dd)
n = nrow(dd)
learn = sample(1:n, size=round(0.67 * n))
nlearn = length(learn)
ntest = n - nlearn
train<-dd[learn,]
test<-dd[-learn,]
```

```
# it's much easier to read a tree with a graphic
plot(ct, margin=0.05, compress=TRUE, main="Decision Tree")
text(ct, use.n=TRUE, pretty=1, all=TRUE, cex=0.7)

printcp(ct)
ct.cp<-ct
ct1 = rpart(Status ~ ., data = dd[learn,])
ct1
# check results of the complexity parameter table
ct1$cptable
# we can use the function 'plotcp' to see the results
# the 'best' tree is the one with the lowest xerror
plotcp(ct1, las=2, cex.axis=0.8)

plot(ct1, margin=0.05, compress=TRUE, main="Decision Tree")
text(ct1, use.n=TRUE, pretty=1, all=TRUE, cex=0.5)
summary(ct1)

printcp(ct1)
ct.cp<-ct



# what is the minimum XERROR?
min(ct1$cptable[,4])
min.xe = which(ct1$cptable[,4] == min(ct1$cptable[,4]))
# the optimal tree corresponds to a cp=0.003
ct1$cptable[min.xe,]

# Now that we know that the 'best' tree has cp=0.03
# we can plugin that information in the parameters
ct2 = rpart(Status ~ .,
                    data = dd[learn,],
                    parms = list(prior=c(0.50, 0.50), split='gini'),
                    control = rpart.control(cp=0.01, xval=0))
# plot
par(mar = c(1,1,2,0.5))
plot(ct2, margin=0.05, compress=TRUE, main="Decision Tree")
text(ct2, use.n=TRUE, pretty=1, all=TRUE, cex=0.5)
summary(ct2)
printcp(ct2)

# calculate error rate in the learning sample
# (this will give a matrix)
ct2.learn = predict(ct2, data=ddtot[learn,])
```

```
# create a vector with predicted status
ct2.learnp = rep("", nlearn)
ct2.learnp[ct2.learn[,1] < 0.5] = "pred_neg"
ct2.learnp[ct2.learn[,1] >= 0.5] = "pred_pos"
# let's make a table
status_learn = table(dd$Status[learn], ct2.learnp)
# classification error
100 * sum(diag(status_learn)) / nlearn

# let's make a table
status_test = table(dd$Status[-learn], ct2.testp)

table(dd$Status[-learn], ct2.testp)
# classification error
error_rate.test=100 * sum(diag(status_test)) / ntest

error_rate.test

# score test data
test$dtscore <- predict(ct2,type='prob',test)
m2_pred <- prediction(test$dtscore[,2],test$Status)
m2_perf <- performance(m2_pred,"tpr","fpr")

# MOdel performance plot ROC Curve
plot(m2_perf, col="red",lwd=2, main="ROC Curve for Deccision tree")
abline(a=0,b=1,lwd=2,lty=2,col="gray")

#KS, Gini & AUC m1
dtAUROC <- round(performance(m2_pred, measure = "auc")@y.values[[1]]*100, 2)
dtKS <- round(max(attr(m2_perf,'y.values')[[1]]-attr(m2_perf,'x.values')[[1]])*100, 2)
dtGini <- (2*dtAUROC - 100)
cat("AUROC: ",dtAUROC,"\tKS: ", dtKS, "\tGini:", dtGini, "\n")
# Basic Model

svmmodel<-ksvm(Status~., data=dd[learn,], kernel="vanilladot")
svmmodelpred <- predict(svmmodel, dd[-learn,], type="response")
head(svmmodelpred)
# Model accuracy:
table(svmmodelpred, dd[-learn,]$Status)
#agreement
svmmodelaccuracy<- (svmmodelpred == dd[-learn,]$Status)
# Compute at the prediction scores
svmmodelscore <- predict(svmmodel,dd[-learn,], type="decision")
svmmodelpred <- prediction(svmmodelscore, dd[-learn,]$Status)
```

```
# Plot ROC curve
m7_1_perf <- performance(svmmodelpred, measure = "tpr", x.measure = "fpr")
plot(m7_1_perf, col="gold", lwd=2, main="ROC Curve for SVM ")
abline(a=0,b=1,lwd=2,lty=2,col="gray")




# Plot precision/recall curve
m7_1_perf_precision <- performance(svmmodelpred, measure = "prec", x.measure = "rec")
plot(m7_1_perf_precision, main="m7_1 SVM:Plot precision/recall curve")

# Plot accuracy as function of threshold
m7_1_perf_acc <- performance(svmmodelpred, measure = "acc")
plot(m7_1_perf_acc, main="m7_1 SVM:Plot accuracy as function of threshold")

# Model Performance

#KS & AUC m7_1
m7_1_KS <- round(max(attr(m7_1_perf,'y.values')[[1]]-attr(m7_1_perf,'x.values')[[1]])*100, 2)
m7_1_AUROC <- round(performance(svmmodelpred, measure = "auc")@y.values[[1]]*100, 2)
m7_1_Gini <- (2*m7_1_AUROC - 100)
cat("AUROC: ",m7_1_AUROC,"\tKS: ", m7_1_KS, "\tGini:", m7_1_Gini, "\n")
```

Appendix

```
setwd("G:/wangcai/New folder/CreditScoring-master")

# load package FactoMineR and ggplot2
require(FactoMineR)
require(ggplot2)

# read cleaned data set
dd = read.csv("G:/wangcai/New folder/CreditScoring-master/CleanCreditScoring.csv",
header=TRUE, stringsAsFactors=TRUE)

#Apply logistic regression using all the variables
#
================================================================================
=====

# let's keep 2/3 of the data for learning, and 1/3 for testing
n = nrow(dd)
learn = sample(1:n, size=round(0.67 * n))
nlearn = length(learn)
ntest = n - nlearn
train<-dd[learn,]
test<-dd[-learn,]

# "whole enchilada" logistic regression model
gl1 = glm(Status ~ ., data=dd[learn,], family=binomial)

# use the 'summary' function to obtain more details on the logistic model
# (pay attention to the signicance of the coefficients)
summary(gl1)
```

```
      # let's use the 'anova' function to get a sequential analysis of
      # variance of the model fit (ie importance of variables in the model)
      anova(gl1)

      # we can also use the 'step' function to perform model selection by
      # applying a backward elimination method based on AIC (Akaike Info. Criterion)
      step(gl1)
      #
================================================================================
=====
      # Apply logistic regression after removing some variables
      #
================================================================================
=====

      # new model
      glf <- glm(formula = Status ~ Age + Income + Debt + Amount + Finrat +
                           seniorityR + expensesR + assetsR + priceR + savingsR + Home + Marital +
Records +
                           Job, family = binomial, data = dd[learn, ])
      # check summary
      summary(glf)
      # check coefficients
      exp(glf$coefficients)
      # re-expressed fitted values
      glf$fitted.values = 1 - glf$fitted.values

      # create vector for predictions
      glfpred = rep(NA, length(glf$fitted.values))
      glfpred[glf$fitted.values < 0.5] = 0
      glfpred[glf$fitted.values >= 0.5] = 1

      # how is the prediction? (confusion matrix)
      table(dd$Status[learn], glfpred)
      # error rate
      error_rate.learn = 100*sum(diag(table(dd$Status[learn], glfpred))) / nlearn
      error_rate.learn

      # let's use the test data to get predictions
      glft = predict(glf, newdata=dd[-learn,])
      pt = 1 / (1 + exp(-glft))
      pt = 1 - pt
      # vector of predicted values
```

```
	glftpred = rep(NA, length(pt))
	glftpred[pt < 0.5] = 0
	glftpred[pt >= 0.5] = 1

	# confusion matrix
	table(dd$Status[-learn], glftpred)
	error_rate.test = 100*sum(diag(table(dd$Status[-learn], glftpred))) / ntest
	error_rate.test
	#
================================================================================
=====
	# Concentration curve
	#
================================================================================
=====

	ac_tot = 100*(1:ntest) / ntest

	pt.ord = order(pt, decreasing=T)

	Status_test = dd$Status[-learn]
	npos = table(Status_test)[2]

	ac_pos_test = 100*cumsum(Status_test[pt.ord] == "good") / npos

	plot(ac_tot, ac_pos_test, type="l", lwd=2,
		main="Concentration Curve")
	lines(ac_tot, ac_tot, col="gray70", lwd=2)
	#
================================================================================
=====
	# ROC Curve
	#
================================================================================
=====

	nneg = ntest - npos

	ac_neg_test = 100*cumsum(Status_test[pt.ord]=="bad") / nneg

	plot(ac_neg_test, ac_pos_test, type="l", lwd=2, main="ROC Curve", col="red")
	lines(ac_neg_test, ac_neg_test, col="grey70", lwd=2)

	#load library#######################################################
```

```
library(ROCR)
#score test data set
test$score<-predict(glf,test,type="response")
pred<-prediction(test$score,test$Status)
perf <- performance(pred,"tpr","fpr")
#roc
plot(perf,col="green",lwd=2,main="ROC Curve for Logistic regression")
abline(a=0,b=1,lwd=2,lty=2,col="gray")

#KS, Gini & AUC m1
m1_KS <- round(max(attr(perf,'y.values')[[1]]-attr(perf,'x.values')[[1]])*100, 2)
m1_AUROC <- round(performance(pred, measure = "auc")@y.values[[1]]*100, 2)
m1_Gini <- (2*m1_AUROC - 100)
cat("AUROC: ",m1_AUROC,"\tKS: ", m1_KS, "\tGini:", m1_Gini, "\n")
require(ggplot2)
require(rpart)
dd = read.csv("G:/wangcai/New folder/CreditScoring-master/CleanCreditScoring.csv",
header=TRUE, stringsAsFactors=TRUE)

# Let's
ct = rpart(Status ~ ., data=dd)
n = nrow(dd)
learn = sample(1:n, size=round(0.67 * n))
nlearn = length(learn)
ntest = n - nlearn
train<-dd[learn,]
test<-dd[-learn,]

# it's much easier to read a tree with a graphic
plot(ct, margin=0.05, compress=TRUE, main="Decision Tree")
text(ct, use.n=TRUE, pretty=1, all=TRUE, cex=0.7)

printcp(ct)
ct.cp<-ct
ct1 = rpart(Status ~ ., data = dd[learn,])
ct1
# check results of the complexity parameter table
ct1$cptable
# we can use the function 'plotcp' to see the results
# the 'best' tree is the one with the lowest xerror
plotcp(ct1, las=2, cex.axis=0.8)

plot(ct1, margin=0.05, compress=TRUE, main="Decision Tree")
text(ct1, use.n=TRUE, pretty=1, all=TRUE, cex=0.5)
```

```
summary(ct1)

printcp(ct1)
ct.cp<-ct


# what is the minimum XERROR?
min(ct1$cptable[,4])
min.xe = which(ct1$cptable[,4] == min(ct1$cptable[,4]))
# the optimal tree corresponds to a cp=0.003
ct1$cptable[min.xe,]

# Now that we know that the 'best' tree has cp=0.03
# we can plugin that information in the parameters
ct2 = rpart(Status ~ .,
                    data = dd[learn,],
                    parms = list(prior=c(0.50, 0.50), split='gini'),
                    control = rpart.control(cp=0.01, xval=0))
# plot
par(mar = c(1,1,2,0.5))
plot(ct2, margin=0.05, compress=TRUE, main="Decision Tree")
text(ct2, use.n=TRUE, pretty=1, all=TRUE, cex=0.5)
summary(ct2)
printcp(ct2)

# calculate error rate in the learning sample
# (this will give a matrix)
ct2.learn = predict(ct2, data=ddtot[learn,])
# create a vector with predicted status
ct2.learnp = rep("", nlearn)
ct2.learnp[ct2.learn[,1] < 0.5] = "pred_neg"
ct2.learnp[ct2.learn[,1] >= 0.5] = "pred_pos"
# let's make a table
status_learn = table(dd$Status[learn], ct2.learnp)
# classification error
100 * sum(diag(status_learn)) / nlearn

# let's make a table
status_test = table(dd$Status[-learn], ct2.testp)

table(dd$Status[-learn], ct2.testp)
# classification error
error_rate.test=100 * sum(diag(status_test)) / ntest
```

```
error_rate.test

# score test data
test$dtscore <- predict(ct2,type='prob',test)
m2_pred <- prediction(test$dtscore[,2],test$Status)
m2_perf <- performance(m2_pred,"tpr","fpr")

# MOdel performance plot ROC Curve
plot(m2_perf, col="red",lwd=2, main="ROC Curve for Deccision tree")
abline(a=0,b=1,lwd=2,lty=2,col="gray")

#KS, Gini & AUC m1
dtAUROC <- round(performance(m2_pred, measure = "auc")@y.values[[1]]*100, 2)
dtKS <- round(max(attr(m2_perf,'y.values')[[1]]-attr(m2_perf,'x.values')[[1]])*100, 2)
dtGini <- (2*dtAUROC - 100)
cat("AUROC: ",dtAUROC,"\tKS: ", dtKS, "\tGini:", dtGini, "\n")
# Basic Model

svmmodel<-ksvm(Status~., data=dd[learn,], kernel="vanilladot")
svmmodelpred <- predict(svmmodel, dd[-learn,], type="response")
head(svmmodelpred)
# Model accuracy:
table(svmmodelpred, dd[-learn,]$Status)
#agreement
svmmodelaccuracy<- (svmmodelpred == dd[-learn,]$Status)
# Compute at the prediction scores
svmmodelscore <- predict(svmmodel,dd[-learn,], type="decision")
svmmodelpred <- prediction(svmmodelscore, dd[-learn,]$Status)

# Plot ROC curve
m7_1_perf <- performance(svmmodelpred, measure = "tpr", x.measure = "fpr")
plot(m7_1_perf, col="gold", lwd=2, main="ROC Curve for SVM ")
abline(a=0,b=1,lwd=2,lty=2,col="gray")




# Plot precision/recall curve
m7_1_perf_precision <- performance(svmmodelpred, measure = "prec", x.measure =
"rec")
plot(m7_1_perf_precision, main="m7_1 SVM:Plot precision/recall curve")

# Plot accuracy as function of threshold
m7_1_perf_acc <- performance(svmmodelpred, measure = "acc")
```

```
plot(m7_1_perf_acc, main="m7_1 SVM:Plot accuracy as function of threshold")

# Model Performance

#KS & AUC m7_1
m7_1_KS <- round(max(attr(m7_1_perf,'y.values')[[1]]-attr(m7_1_perf,'x.values')[[1]])*100, 2)
m7_1_AUROC <- round(performance(svmmodelpred, measure = "auc")@y.values[[1]]*100, 2)
m7_1_Gini <- (2*m7_1_AUROC - 100)
cat("AUROC: ",m7_1_AUROC,"\tKS: ", m7_1_KS, "\tGini:", m7_1_Gini, "\n")
```