ACTL4305/5305 Actuarial Data Analytic Application

Week 3: Proposed Solutions

Credit Risk Modelling

Learning Objectives

- Understand how to build logistic regression models with shrinkage techniques, such as lasso, ridge, and elastic net, to predict clients' default rates.
- Understand how to find the optimal regularization parameter for shrinkage techniques.
- Use evaluation metrics to compare the predictive performance of different models in the test set.

1 Introduction

In this tutorial, we use the credit data of the credit card clients in Taiwan to predict if a client will default or not. The data set is the customers' default payments which include 30000 instances described over 24 attributes. This data set contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005. The risk involved in lending business or credit card can be the business loss by not approving the good client or financial loss by approving the client who is at bad risk.

Questions for Lab 3:

- Use the code below to start building a logistic regression model without any shrinkage techniques.

- Build logistic regression models with lasso, ridge, and elastic net penalties. *Hint:* Use 'cv.glmnet()' to find the optimal regularization parameter for these shrinkage techniques.

- Compare the predictive performance of all four models on the test set.

- Is your data balanced? If not, implement data balancing techniques and check whether the results improve with balancing enabled.

2 Models Details

2.1 Logistic Regression

Logistic regression can be considered a special case of linear regression models. A logistic regression model specifies that an appropriate function of the fitted probability of the event is a linear function of the observed values of the available explanatory variables. The major advantage of this approach is that it can produce a simple probabilistic formula of classification. The weaknesses are that logistic regression cannot properly deal with the problems of non-linear and interactive effects of explanatory variables. The predicted values under the logistic regression are based on the likelihood of each event using the logit function. The probability is given by:

$$\mathbb{P}(Y = 1|X = x) = \frac{1}{1 + e^{-\sum_j^p x_{ij}\beta_j}}. \tag{1}$$

We estimate β by minimizing the negative log-likelihood which, of course, is the same as maximizing the log-likelihood. The quantity one minimizes is thus:

$$\mathbb{L} = -\sum_{i=1}^{N} [y_i \log(\mathbb{P}(Y_i|X_i)) + (1 - y_i) \log(1 - \mathbb{P}(Y_i|X_i))], \tag{2}$$

where N is the number of observations.

2.2 Ridge Penalty

Ridge penalty can be used when all the features are relevant in predicting the default probability. Ridge shrinks all of the weighting coefficients towards zero via shrinkage parameter λ, however, none of them will be set exactly to zero. Ridge penalty approximates the coefficients by minimizing the quantity:

$$\widehat{\beta} = \underset{\beta}{\operatorname{argmin}}\ \mathbb{L} + \lambda \sum_{j=1}^{p} \beta_j^2, \tag{3}$$

where j is the number of the features (X).

2.3 Lasso Penalty

Lasso penalty has a tuning parameter λ that controls the degree of shrinkage applied to the model coefficient. If the penalty parameter λ is chosen correctly, the total generalization error will decrease. This penalty parameter is optimally determined via cross-validation. Lasso regression can force some weighting coefficients to be identically zero if the constraint λ is tight enough, which results in a simple and interpretable model. Lasso penalty approximates the coefficients by minimizing the quantity:

$$\widehat{\beta} = \underset{\beta}{\operatorname{argmin}}\ \mathbb{L} + \lambda \sum_{j=1}^{p} |\beta_j|. \tag{4}$$

2.4 Elastic Net Penalty

The elastic net penalty can be used to keep or drop the correlated features jointly. It combines the ℓ_1 and ℓ_2 properties of lasso and ridge penalties, respectively into:

$$\widehat{\beta} = \underset{\beta}{\operatorname{argmin}}\ \mathbb{L} + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2. \tag{5}$$

The elastic net produces a sparse model depending on the choices of λ_1 and λ_2.

3 Model Evaluation

We use the area under the receiver operating characteristic curve (AUC) to evaluate the models implemented in this case study. The receiver operating characteristic curve (ROC curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters namely true positive rate and false-positive rate. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. AUC measures the entire two-dimensional area underneath the entire ROC curve from (0,0) to (1,1). The higher the AUC, the better the model is at predicting non-defaulters as non-defaulters and defaulters as defaulters.

3.1 Loading the Required Packages

We load multiple packages required for this analysis. The package `readxl` is used to read the excel data with the extension `.xls`. The package `PRROC` is used for computing the Area Under the Curve (AUC), while the package `caret` contains functions for data splitting and fitting the classification and regression models with the richness of different parameter settings. The package `corrplot` helps in plotting the correlation between the numeric features. The package `tidyverse` allows us to do multiple data manipulation and finally `glmnet` permits us to fit the shrinkage models. `doParallel` is used to perform parallel computing but is optional.

```r
knitr::opts_chunk$set(echo = TRUE, cache = FALSE, warning = FALSE, message = FALSE)
library(readxl) #read excel files
library(tidyverse)
library(glmnet)
library(caret)
library(e1071)
library(PRROC) #roc curve
library(doParallel)
library(Rcpp)
library(MASS)
```

3.2 Import Data

```r
# read excel file
Raw_credit <- as.data.frame(read_excel("credit.xls", skip = 1))
credit<-Raw_credit
```

3.3 Data Preparations

We prepare the data ready for predicting the probability of default. We rename some of the features, handle the missing values, and change the features into the appropriate format as well as scaling the numeric variables. And finally, we split the data into training and testing data as well as ensuring the training data is balanced.

```r
#Renaming columns
colnames(credit)[7] <- "PAY_1"
colnames(credit)[25] <- "default"

#removing the customer index
credit <- credit[,-1]

#fixing errors in the data - similar to week 2
```

```r
credit$MARRIAGE[credit$MARRIAGE == 0] <- 3
credit$EDUCATION[credit$EDUCATION %in% c(0,5,6)] <- 4

#Changing categorical vars to factor - leaving the PAY variables as they cause issues as factors.
credit[,c(2:4,24)] <- lapply(credit[,c(2:4,24)], FUN=factor)
# factor: SEX, EDUCATION, MARRIAGE, default
```

3.4 Data Splitting

The data is split into training and test datasets for evaluating the model and its accuracy. The training set is used to build the model and the test dataset will validate the accuracy of the developed model. The dataset is divided in the ratio of 70% training set and 30% test dataset. We randomly split the data using the function `createDataPartition()` from the `caret` package or by using `sample()`.

```r
set.seed(1)# note that different seeds will lead to different sampling results.
#For example, you can set seed to be 2 and check the sampled index below.
index <- sample(1:length(credit$LIMIT_BAL), 0.7*length(credit$LIMIT_BAL)) # "replace = FALSE"
#index <- createDataPartition(credit$default, p = 0.7, list = FALSE) #alt method

mean(credit$default==1)

x_train <- credit[index, -24] # all independent variables in the training set
y_train <- credit[index, 24] # the dependent variable "default" in the training set
Data_train<-credit[index,] # X and Y

x_test <- credit[-index, -24] # all independent variables in the test set
y_test <- credit[-index, 24] # the dependent variables in the test set
Data_test<-credit[-index,] # X and Y
```

4 Modelling

4.1 Predicting Using the Shrinkage Predictive Methods

We train the shrinkage predictive methods namely Lasso (4), Ridge (3), and Elastic net (5) regressions on the training data sets via the cross-validation and then predict whether the client is credible or not credible. Take note of the transformations done to the data here as each model and function will have a particular way it likes the data so this is important to understand.

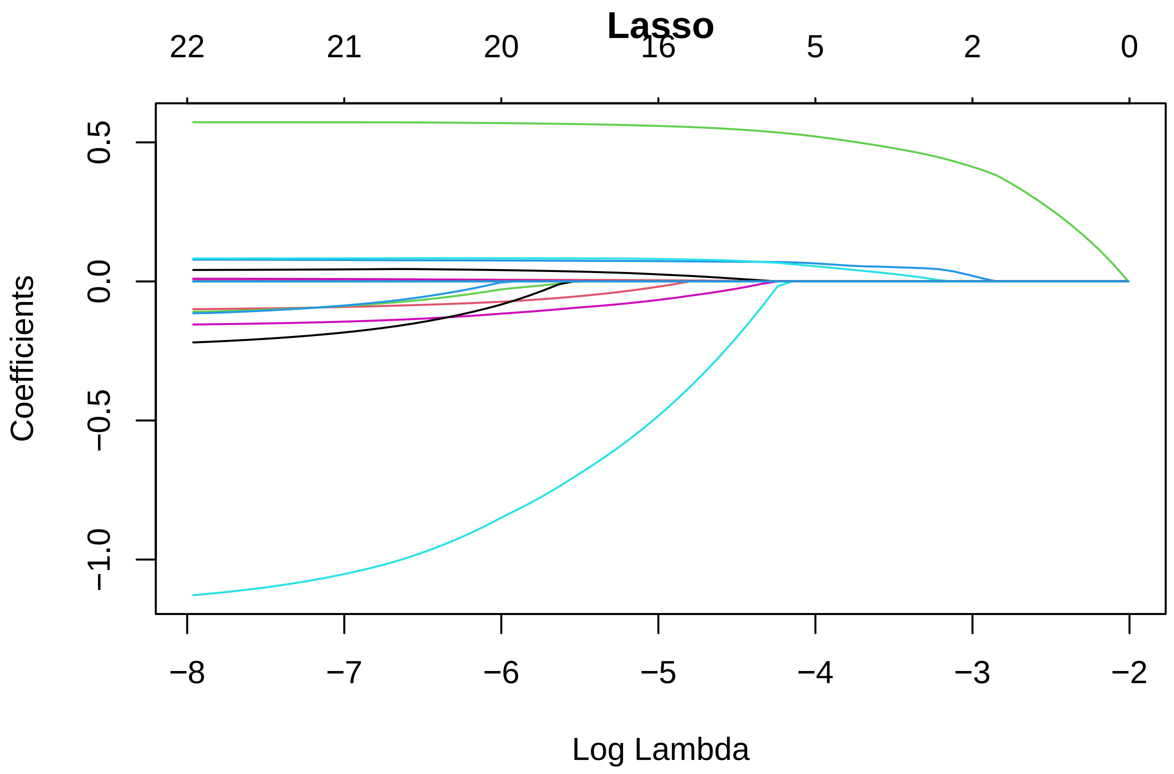
```r
# 1. We will use glmnet() to apply Lasso, Ridge, and elasticnet penalties,
#so we have to transform the current X and Y into matrix format.
#Note that in matrix format, we do not have factor variables,
#so the factor variables will be transform into dummy variables.
x_train_matrix <- model.matrix(~., x_train)
#model.matrix(): create a matrix and expand factors to a set of dummy variables
y_train_matrix <- as.matrix(y_train)
#Note the test set should NOT be used in the model building process.
x_test_matrix <- model.matrix(~., x_test)
y_test_matrix <- as.matrix(y_test)
```

We use 10-fold CV to tune find the optimal the regularization parameter (*lambda*).

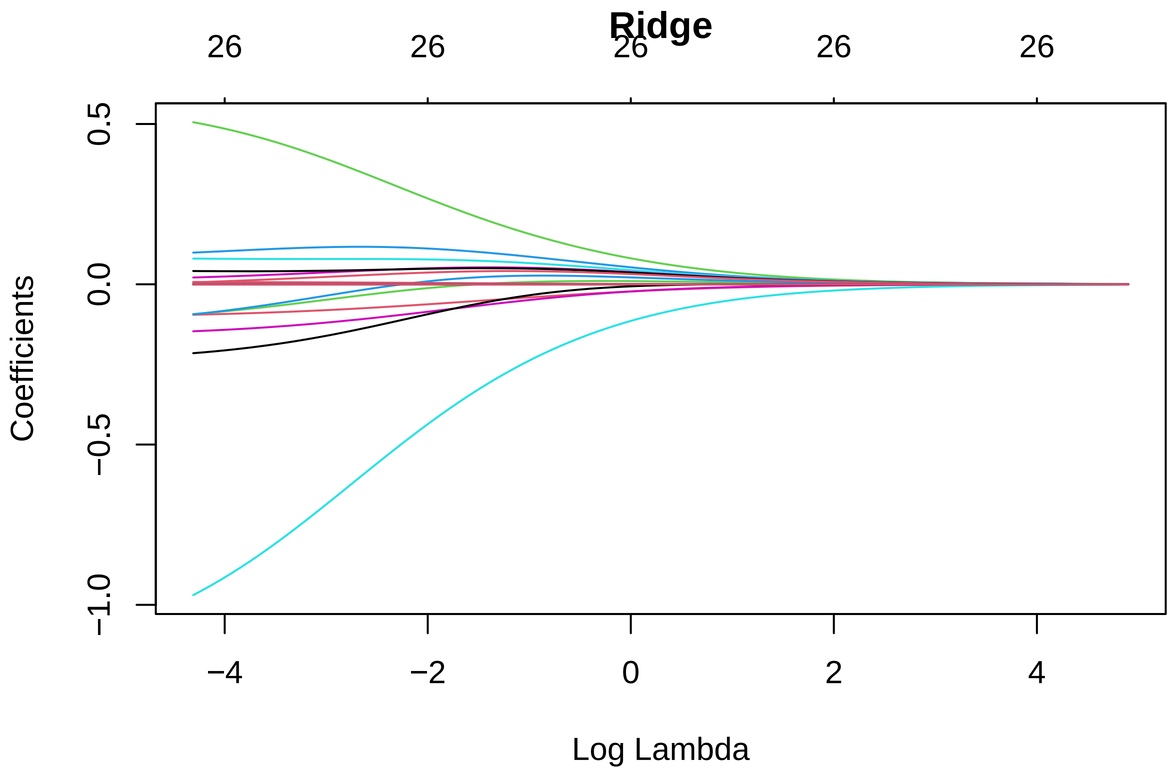
```
#2. Run cross-validation to find the optimal the regularization parameter(lambda).

#alpha=1 is the lasso penalty,
#alpha=0 the ridge penalty,
#alpha=0.5 the elasticnet penalty,
#Starting parallel computing
cl <- makeCluster(detectCores()-1) #this detects the cores of your computer

registerDoParallel(cl)# start the parallel computing mode to speed up the CV process.

#Lasso penalty
CV_lasso <- cv.glmnet(x_train_matrix, y_train_matrix, family="binomial", type.measure = "auc",
                      alpha = 1, nfolds = 10, parallel = T)
#Ridge penalty
CV_ridge <- cv.glmnet(x_train_matrix, y_train_matrix, family="binomial", type.measure = "auc",
                      alpha = 0, nfolds = 10, parallel = T)
#Elasticnet penalty
CV_EN <- cv.glmnet(x_train_matrix, y_train_matrix, family="binomial", type.measure = "auc",
                      alpha = 0.5, nfolds = 10, parallel = T)
#ending parallel computing - important, otherwise R can get funky.
stopCluster(cl)

#Notes
#family="binomial": this means we are doing a binary classification problem.
#type.measure = "auc": this defines the evaluation metric we use to determine-
#the optimal the regularization parameter lambda ("auc" is for two-class logistic regression only).
#alpha=1 is the lasso penalty,
#alpha=0 the ridge penalty,
#alpha=0.5 the elasticnet penalty,
#parallel = T; parallel computing to speed up the CV process.

# We also fit a logistic regression without using any penalty; i.e., treating lambda as 0.
#Note that when using glm(), we do not use matrix format variables.
#Instead, we need to define the formula for Y and Xs.
LogisticModel<-glm(default~., family = "binomial", data=Data_train)
```

When we use different penalties, the effect from the regularization parameter (λ) on the coefficients (β) is also different. Let's see how the regularization parameters (λ) of different penalties affect the model's coefficients.

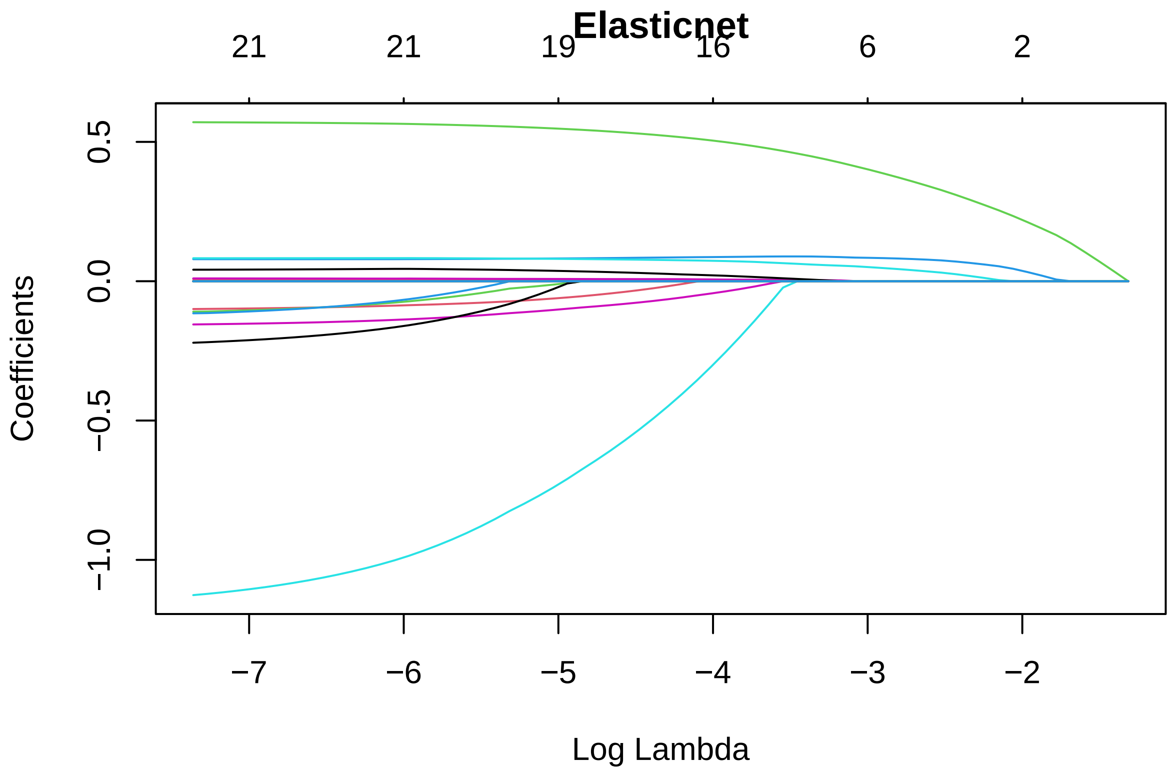
```
plot(CV_lasso$glmnet.fit, xvar = "lambda",main="Lasso")
```



```
plot(CV_ridge$glmnet.fit, xvar = "lambda",main="Ridge")
```



```
plot(CV_EN$glmnet.fit,    xvar = "lambda",main="Elasticnet")
```



4.2 Using the fitted models to make predictions in the test set

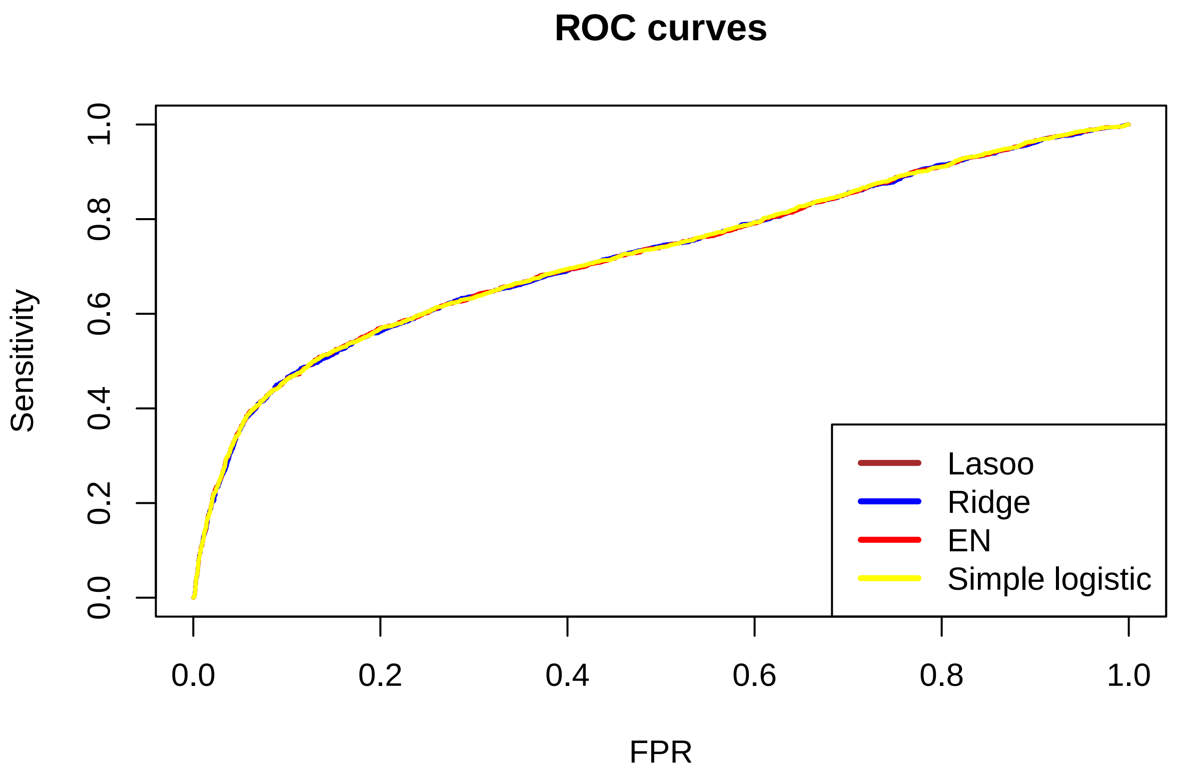
The following codes show how to use fitted models to predict default rates in the test set.

```
#make a prediction using the optimal lambda based on 10-fold CV: s=CV_lasso$lambda.min
#type = "response" important, this will give probabilities
prediction_lasso <- predict(CV_lasso, s=CV_lasso$lambda.min, newx=x_test_matrix, type="response")

prediction_ridge <- predict(CV_ridge, s=CV_ridge$lambda.min, newx=x_test_matrix, type="response")

prediction_EN <- predict(CV_EN, s=CV_EN$lambda.min, newx=x_test_matrix, type="response")

prediction_Logistic <- predict(LogisticModel, newdata=Data_test, type="response")
```

4.3 Models Comparison

You can plot the ROC curves of the different models in a same figure. We can see that the ROC curves of the four models are similar and close to each other, indicating the four models have the similar predictive performance measured by ROC.

```
ROC_lasso<-PRROC::roc.curve(scores.class0 = prediction_lasso,
                            weights.class0 = as.numeric(Data_test$default)-1,curve = T)

ROC_ridge<-PRROC::roc.curve(scores.class0 = prediction_ridge,
                                  weights.class0 = as.numeric(Data_test$default)-1,curve = T)

ROC_EN<-PRROC::roc.curve(scores.class0 = prediction_EN,
                              weights.class0 = as.numeric(Data_test$default)-1,curve = T)

ROC_Logistic<-PRROC::roc.curve(scores.class0 = prediction_Logistic,
                                    weights.class0 = as.numeric(Data_test$default)-1,curve = T)

plot(ROC_lasso,color = "brown", main="ROC curves", auc.main = F, lwd=2)
plot(ROC_ridge,color ="blue",add=T, lwd=2)
plot(ROC_EN,color ="red",add=T, lwd=2)
plot(ROC_Logistic,color ="yellow",add=T, lwd=2)
legend("bottomright", legend = c("Lasoo", "Ridge","EN", "Simple logistic"),
       lwd = 3, col = c("brown", "blue","red", "yellow"))
```



We can also check the four models' AUC, the area under the ROC curve. A larger AUC suggests a better model. In this case, the simple logistic regression has the highest AUC.

```
ROC_lasso$auc #Lasso
```

```
## [1] 0.7215665
```

```
ROC_ridge$auc #Ridge
```

```
## [1] 0.7209229
```

```
ROC_EN$auc #EN
```

```
## [1] 0.7215649
```

```
ROC_Logistic$auc #simple logistic
```

```
## [1] 0.722072
```

Besides ROC and AUC, there are many evaluation metrics we can use for a binary classification problem. To check these metrics, we can use the following code to get all metrics for a model. For example, the logistic regression with Lasso penalty has the following performance. Generally speaking, there is no single metric that works well for all cases and different models may perform well in one metric but bad in another. Thus, we should carefully choose the metrics according to the business contexts and requirements.

```
PredClass_lasso<- as.integer(prediction_lasso>0.5)
PredClass_ridge<- as.integer(prediction_ridge>0.5)
PredClass_EN<- as.integer(prediction_EN>0.5)
PredClass_Logistic<- as.integer(prediction_Logistic>0.5)

ConfuMatrix_lasso<- confusionMatrix(as.factor(PredClass_lasso), Data_test$default,positive="1")
ConfuMatrix_ridge<- confusionMatrix(as.factor(PredClass_ridge), Data_test$default,positive="1")
ConfuMatrix_EN<- confusionMatrix(as.factor(PredClass_EN), Data_test$default,positive="1")
ConfuMatrix_Logistic<- confusionMatrix(as.factor(PredClass_Logistic), Data_test$default,positive="1")

ConfuMatrix_lasso
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6838 1556
##          1  159  447
##
##                Accuracy : 0.8094
##                  95% CI : (0.8012, 0.8175)
##     No Information Rate : 0.7774
##     P-Value [Acc > NIR] : 5.984e-14
##
##                   Kappa : 0.2669
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.22317
##             Specificity : 0.97728
##          Pos Pred Value : 0.73762
##          Neg Pred Value : 0.81463
##              Prevalence : 0.22256
##          Detection Rate : 0.04967
##    Detection Prevalence : 0.06733
##       Balanced Accuracy : 0.60022
##
##        'Positive' Class : 1
##
```

Think about which metric is suitable for this credit card default study. A bank will suffer a great loss if their clients default. Hence, we are more interested in how well the models predict the defaulter (class 1 not class 0), who is the minority class. In this case, people usually check Sensitivity, i.e., how many true defaulters are classified to be defaulters. Go and check which model has the highest Sensitivity using the code in the chunk above.

my mse for lasso: 0.1454841

4.4 Balancing

It is important to check whether your data is balanced. If not, implementing data balancing techniques could be beneficial. Subsampling methods can be broadly categorized into:

1. **Down-sampling**: This involves randomly sampling a dataset so that all classes have the same frequency as the minority class. This can be achieved using the `downSample` function from the `caret` package.
2. **Up-sampling**: randomly sampling (with replacement) the minority class to match the size of the majority class. This can be done using the `upSample` function from the `caret` package.
3. **Hybrid methods**: Techniques like SMOTE (from the `SMOTE` package) and ROSE (from the `ROSE` package) both down-sample the majority class and synthesize new data points for the minority class.

However, DO NOT blindly apply data balancing - data balancing should be applied as part of the modelling rather than the data validation and you should check whether the results improve with balancing enabled. In this data set, defaulters take 22.12%, so the data is imbalanced. In this week, We use the upsampling method (`upSample` function from `caret` package) to make the number of defaulters equal to the number of non-defaulters, see their numbers before and after the upsampling.

```
table(Data_train$default)
```

```
##
##     0     1
## 16367  4633
```

```
set.seed(2023)
Balanced_train<-upSample(x=Data_train[,-24],
                         y=Data_train$default)
table(Balanced_train$Class)
```

```
##
##     0     1
## 16367 16367
```

```
Baclanced_x<-Balanced_train[,-24]
Baclanced_y<-Balanced_train[,24]

x_baclanced_matrix <- model.matrix(~., Baclanced_x)
y_baclanced_matrix <- as.matrix(Baclanced_y)

cl <- makeCluster(detectCores()-1) #this detects the cores of your computer
registerDoParallel(cl)# start the parallel computing mode to speed up the CV process.

#Lasso penalty
```

```r
CV_lasso_balanced <- cv.glmnet(x_baclanced_matrix, y_baclanced_matrix,
                                family="binomial", type.measure = "auc",
                     alpha = 1, nfolds = 10, parallel = T)
#Ridge penalty
CV_ridge_balanced<- cv.glmnet(x_baclanced_matrix, y_baclanced_matrix,
                               family="binomial", type.measure = "auc",
                     alpha = 0, nfolds = 10, parallel = T)
#Elasticnet penalty
CV_EN_balanced <- cv.glmnet(x_baclanced_matrix, y_baclanced_matrix,
                             family="binomial", type.measure = "auc",
                     alpha = 0.5, nfolds = 10, parallel = T)
#ending parallel computing - important, otherwise R can get funky.
stopCluster(cl)

LogisticModel_balanced<-glm(Class~., family = "binomial", data=Balanced_train)
```

Then, we use the balanced training data to fit four models again and used them to make predictions in the test set.

```r
#prediction using "balanced" models
prediction_lasso_balanced <- predict(CV_lasso_balanced,
                              s=CV_lasso_balanced$lambda.min, newx=x_test_matrix, type="response")

prediction_ridge_balanced <- predict(CV_ridge_balanced,
                             s=CV_ridge_balanced$lambda.min, newx=x_test_matrix, type="response")

prediction_EN_balanced <- predict(CV_EN_balanced, s=CV_EN_balanced$lambda.min,
                                  newx=x_test_matrix, type="response")

prediction_Logistic_balanced <- predict(LogisticModel_balanced, newdata=Data_test, type="response")

ROC_lasso_balanced<-PRROC::roc.curve(scores.class0 = prediction_lasso_balanced,
                                     weights.class0 = as.numeric(Data_test$default)-1,curve = T)

ROC_ridge_balanced<-PRROC::roc.curve(scores.class0 = prediction_ridge_balanced,
                                     weights.class0 = as.numeric(Data_test$default)-1,curve = T)

ROC_EN_balanced<-PRROC::roc.curve(scores.class0 = prediction_EN_balanced,
                                  weights.class0 = as.numeric(Data_test$default)-1,curve = T)

ROC_Logistic_balanced<-PRROC::roc.curve(scores.class0 = prediction_Logistic_balanced,
                                  weights.class0 = as.numeric(Data_test$default)-1,curve = T)
```

Now check four models' Sensitivity again and brainstorming! Is the simple logistic regression still the best model? Discuss your findings with your mates and tutors:)

```r
PredClass_lasso_balanced<- as.integer(prediction_lasso_balanced>0.5)
PredClass_ridge_balanced<- as.integer(prediction_ridge_balanced>0.5)
PredClass_EN_balanced<- as.integer(prediction_EN_balanced>0.5)
PredClass_Logistic_balanced<- as.integer(prediction_Logistic_balanced>0.5)

ConfuMatrix_lasso_balanced<- confusionMatrix(as.factor(PredClass_lasso_balanced),
                                             Data_test$default,positive="1")
ConfuMatrix_ridge_balanced<- confusionMatrix(as.factor(PredClass_ridge_balanced),
                                              Data_test$default,positive="1")
ConfuMatrix_EN_balanced<- confusionMatrix(as.factor(PredClass_EN_balanced),
                                           Data_test$default,positive="1")
ConfuMatrix_Logistic_balanced<- confusionMatrix(as.factor(PredClass_Logistic_balanced),
                                                 Data_test$default,positive="1")
```

5 Conlusion

Please explore this code and see how results change if you play with the inputs. In reality you will need to make a class prediction so you will need to think about how those probabilities are converted to classes. The important variable here is the threshold, we used 50% here but this can be changed to whatever you think is appropriate. In the case of default detection like this, false-negatives are worse than false-positives so you may wish the lower the threshold to maybe 25% for example (ROC curve can guide you in this).