Computer Lab 1b - Regularized classification

Machine Learning 7.5 credits

Mattias Villani, Department of Statistics, Stockholm University

INSTRUCTIONS:

- The sections named Intro do **not** have any problems for you. Those sections contain code used to set up the data and do some initial analysis, so just read and follow along by running each code chunk.
- Your problems are clearly marked out as Problem 1, Problem 2 etc. You should answer all problems by adding code chunks and text below each question.
- Your submission in Athena should contain two files:
  - This Rmd file with your answers.
  - A PDF version of this file (use the knit to PDF option above).
- You can also write math expression via LaTeX, using the dollar sign, for example $\acute{} \beta$.
- You can navigate the sections of this file clicking (Top Level) in the bottom of RStudio's code window.

Intro1 - Loading packages and data

Loading some packages first. Do `install.packages()` for each package the first time you use a new package.

```
suppressMessages(library(dplyr)) # Package for data transformations and tables
library("RColorBrewer") # for pretty colors
library(caret)  # Fitting ML models with CV and more
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(MLeval) # for plotting ROC curves and more
colors = brewer.pal(12, "Paired")[c(1,2,7,8,3,4,5,6,9,10)];
options(repr.plot.width = 12, repr.plot.height = 12, repr.plot.res = 100) # plot size
set.seed(12332)          # set the seed for reproducability
```

The aim of this part of Lab 1 is to build a classification model for predicting if an items get sold at an eBay auction before the auction ends. The dataset contains data from 1000 auctions of collector coins. Let's load the data and have a look.

```
eBayData = read.csv('https://github.com/mattiasvillani/MLcourse/raw/main/Data/eBayData.csv', sep = ',')
eBayData = eBayData[-1] # Remove a variable that we will not use.
eBayData['Sold'] = as.factor((eBayData['Sold']==1)) # Changing from 1->TRUE, 0->FALSE
levels(eBayData$Sold) <- c("notsold","sold")
head(eBayData)
```

```
##   PowerSeller VerifyID Sealed MinBlem MajBlem LargNeg LogBook MinBidShare Sold
## 1           0        0      0       0       0       0 -0.2237     -0.2088 sold
## 2           1        0      0       0       0       0  0.6073     -0.3478 sold
## 3           1        0      0       0       0       0  0.0332      0.4423 sold
## 4           0        0      0       1       0       0  0.3755      0.1441 sold
## 5           0        0      0       0       0       1  1.4347     -0.4104 sold
```

```
## 6           0        0      0       0       0       0 -0.9142      0.6318 sold
```

The response is the last column `Sold` and we will use all the other features except `nBids` to predict if a sale (`Sold=1`) took place. The features are all characteristics of the auction which would be available at the time the auction was announced.

- **PowerSeller** - does the seller sell often at eBay?
- **VerifyID** - is the sellers ID verified by eBay?
- **Sealed** - is the sold coin sealed in an unopened envelope?
- **MinBlem** - does the coin has a minor blemish (determined by a human from pictures and text description)
- **MajBlem** - does the coin has a major blemish (determined by a human from pictures and text description)
- **LargNeg** - does the seller have a large number of negative feedback from previous actions?
- **LogBook** - log of the coin's value according to book of collector coins.
- **MinBidShare** - (standardized) ratio of the seller's reservation price (lowest tolerated selling price) to the book value.

It is always useful to check if the dataset is imbalanced. There are clearly more sold items that unsold, but the data are not catastrophically imbalanced:

```
message(paste("Percentage of auctions where the object was sold:", 100*sum(eBayData['Sold']=="sold")/10
```

```
## Percentage of auctions where the object was sold: 86.3
```

Split the data using the `createDataPartition` function in `caret` to get 75% of the data for training and 25% for testing.
The default method in `caret` is stratified sampling, keeping the same fraction of positive and negative observations in both training and test datasets.

```
set.seed(123)
inTrain <- createDataPartition(
  y = eBayData$Sold,
  p = .75, # The percentage of data in the training set
  list = FALSE
)
training = eBayData[ inTrain,]
testing  = eBayData[-inTrain,]

message(paste("Percentage of training auctions where the object was sold:",
              100*sum(training['Sold']=="sold")/dim(training)[1]))
```

```
## Percentage of training auctions where the object was sold: 86.2849533954727
```

```
message(paste("Percentage of test auctions where the object was sold:",
              100*sum(testing['Sold']=="sold")/dim(testing)[1]))
```

```
## Percentage of test auctions where the object was sold: 86.3453815261044
```

Let's get started by with a logistic regression with the following steps:

- fit a logistic regression on the training data using maximum likelihood with the `glm` function from the basic `stats` package in R.
- predict the test data using the rule $\Pr(y = 1|\mathbf{x}) > 0.5 \Rightarrow y = 1$
- compute the confusion matrix (using `caret`), and the usual measurements of predictive performance for binary data.

```
glmFit = glm(Sold ~ ., family = binomial, data = training)
yProbs = predict(glmFit, newdata = testing, type = "response")
```

```r
threshold = 0.5 # Predict Sold if yProbs>threshold
yPreds = as.factor(yProbs>threshold)
levels(yPreds) <- c("notsold","sold")
confusionMatrix(yPreds, testing$Sold, positive = "sold")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction notsold sold
##    notsold      15    7
##    sold         19  208
##
##                Accuracy : 0.8956
##                  95% CI : (0.8508, 0.9306)
##     No Information Rate : 0.8635
##     P-Value [Acc > NIR] : 0.07955
##
##                   Kappa : 0.4799
##
##  Mcnemar's Test P-Value : 0.03098
##
##             Sensitivity : 0.9674
##             Specificity : 0.4412
##          Pos Pred Value : 0.9163
##          Neg Pred Value : 0.6818
##              Prevalence : 0.8635
##          Detection Rate : 0.8353
##    Detection Prevalence : 0.9116
##       Balanced Accuracy : 0.7043
##
##        'Positive' Class : sold
##
```

Problem 1

- 1a) Reconstruct the 2-by-2 confusion matrix for the test data without using a package, i.e. code it up from yPreds and testing$Sold yourself.
- 1b) Use the confusion matrix in 1a) to compute the accuracy, sensitivity and specificity and the classifier.
- 1c) Compute the ROC curve from the above fitted glm model and plot it. No packages allowed.

Intro2 - Using the caret package

Let's use the `caret` package to fit a **elastic net** classifier where the two hyperparameters α and λ are chosen by 10-fold cross-validation.

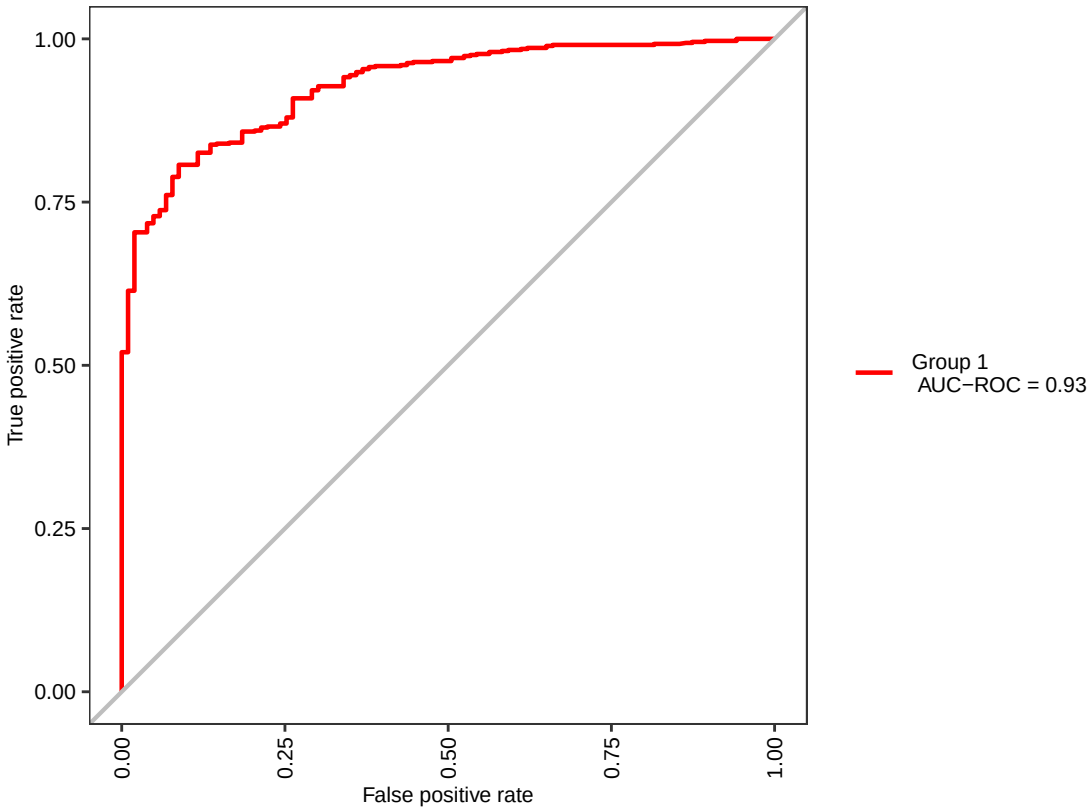
```r
cv10 <- trainControl(
  method = "cv",
  number = 10, # number of folds
  classProbs = TRUE,
  summaryFunction = twoClassSummary, # Standard summary for binary outcomes
  savePredictions = TRUE # Important, otherwise MLeval below will not work.
)
glmnetFit <- train(
```

```
  Sold ~ .,
  data = training,
  method = "glmnet",
  preProc = c("center", "scale"), # the covariates are centered and scaled
  tuneLength = 10, # The number of tuning parameter values to try
  trControl = cv10,
  metric = "ROC"
)
```

The `MLeval` package can be used to plot ROC curves from model fit with `caret`. By just supplying the fitted model object `glmnetFit` to the function we get the cross-validated results from the training data, i.e. this evaluation is not evaluating against the `testing` data. See below for how to evaluate on the testing data using the `MLeval` package.

```
glmnetEval = evalm(glmnetFit, plots='r', rlinethick=0.8, fsize=8) # plots='r' gives ROC
```

```
## ***MLeval: Machine Learning Model Evaluation***

## Input: caret train function object

## Not averaging probs.

## Group 1 type: cv

## Observations: 751

## Number of groups: 1

## Observations per group: 751

## Positive: sold

## Negative: notsold

## Group: Group 1

## Positive: 648

## Negative: 103

## ***Performance Metrics***

## Group 1 Optimal Informedness = 0.7197201246554

## Group 1 AUC-ROC = 0.93
```



The optimal model returned in the caret object `glmnetFit` can be directly used for prediction of the test data. The `MLeval` package can again be used for the evaluation, this time on the `testing` data.

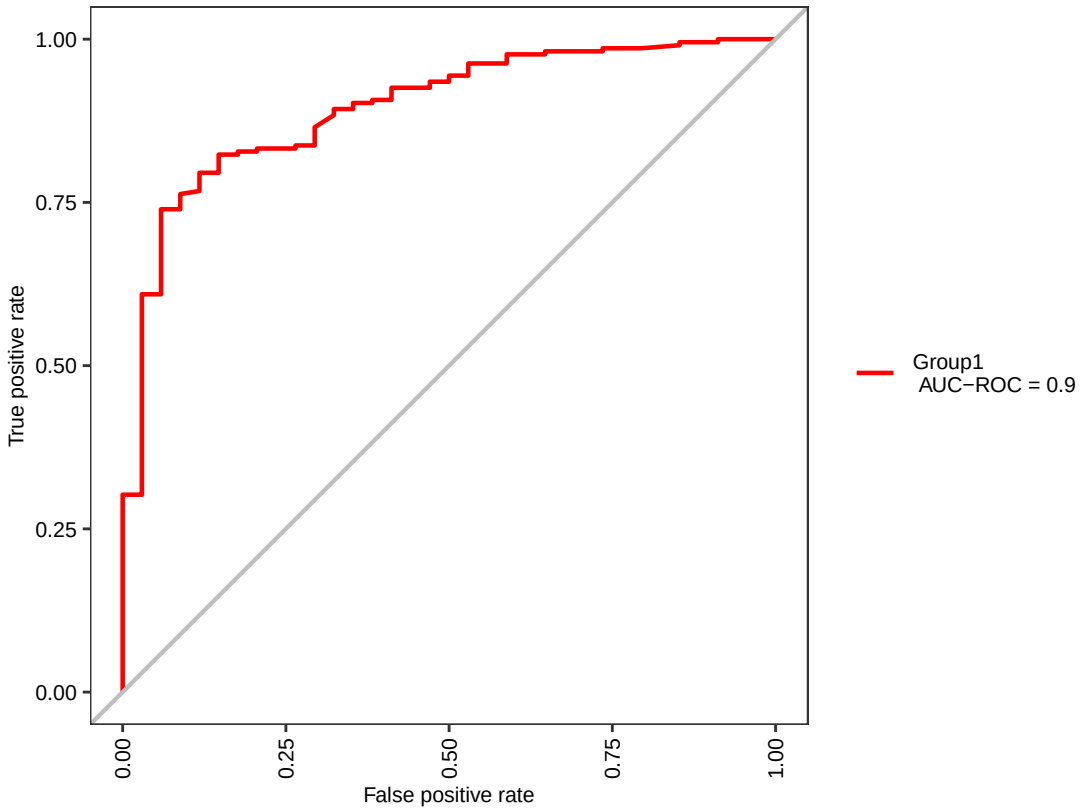
```
yPreds = predict(glmnetFit, newdata = testing)
yProbs = predict(glmnetFit, newdata = testing, type = "prob")
confusionMatrix(yPreds, testing$Sold, positive = "sold")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction notsold sold
##    notsold      14    5
##    sold         20  210
##
##                Accuracy : 0.8996
##                  95% CI : (0.8554, 0.934)
##     No Information Rate : 0.8635
##     P-Value [Acc > NIR] : 0.05403
##
##                   Kappa : 0.4771
##
##  Mcnemar's Test P-Value : 0.00511
##
##             Sensitivity : 0.9767
##             Specificity : 0.4118
##          Pos Pred Value : 0.9130
##          Neg Pred Value : 0.7368
##              Prevalence : 0.8635
##          Detection Rate : 0.8434
```

```
##    Detection Prevalence : 0.9237
##       Balanced Accuracy : 0.6943
##
##        'Positive' Class : sold
##
```

```
res = evalm(data.frame(yProbs, testing$Sold), plots='r', rlinethick=0.8, fsize=8)
```

```
## ***MLeval: Machine Learning Model Evaluation***
## Input: data frame of probabilities of observed labels
## Group does not exist, making column.
## Observations: 249
## Number of groups: 1
## Observations per group: 249
## Positive: sold
## Negative: notsold
## Group: Group1
## Positive: 215
## Negative: 34
## ***Performance Metrics***
## Group1 Optimal Informedness = 0.680711354309166
## Group1 AUC-ROC = 0.9
```



Prob-

lem 2

Fit a random forest using the `rf` package in `caret` to the training data with tuning parameters chosen by 10-fold cross-validation. Plot the ROC curve and compute AUC for the `testing` data using the package `MLeval`. Compute also the accuracy and recall on the `testing` data.

Problem 3 Same as Problem 2, but using the k-nearest neighbor classifier `knn` in caret, with k chosen by 10-fold cross-validation between $k = 1$ and $k = 10$. You need to use the `tuneGrid` option instead of the `tuneLength` option in `caret` for this.

Problem 4 Same as Problem 2, but using stochastic gradient boosting through the `gbm` package in `caret`. There are four tuning parameters here, but you can keep two of them fixed in the training: `n.trees = 100` and `n.minobsinnode = 10`. You need to use the `tuneGrid` option instead of the `tuneLength` option in `caret` for this. Just before the tuneGrid argument in the train function, you can add the argument `verbose = FALSE` to avoid unnecessary messages from being printed during the training process.

Problem 5 Same as Problem 2, but using any model from `caret` that you are curious about. Here is the list of models in `caret`.

Problem 6 Plot the ROC curves on the `testing` data from models fit under Problem 2-6 in a single plot. [Hint: the `evalm` function in the `MLeval` package can plot for more than one model.]