# 2. German Credit Data

Hair Parra

2024-01-15

## German Credit Data

- The German credit data set (Lichman, 2013) is another data set that we will use a lot.
- The data and description can be found here: UCI Machine Learning Repository
- This data set classifies 1000 people described by a set of 20 attributes as good or bad credit risks.
- The target variable, V21, is binary and is recorded to 0-1 (1-2 in the original data); 0=good risk and 1=bad risk.
- We work with a first version of the data set that includes 9 numeric covariates, 11 factor covariates, and the target variable.
- The goal of this example is to show you how to apply a **logistic regression Lasso model**

### Data Preprocessing

```r
# Prepare the German Credit data set

# Load path library
library("here")
```

```
## here() starts at C:/Users/jairp/OneDrive/Desktop_remote/HEC Montreal/3. Winter 2024/Advanced Statistical Lea
```

```r
# Set the path to the data file
gercred = read.table(here("code_data_W2024", "german.data"))

# Recode the target and two binary covariates to 0-1
gercred$V21 = as.numeric(gercred$V21 == 2)
gercred$V19 = as.numeric(gercred$V19 == "A192")
gercred$V20 = as.numeric(gercred$V20 == "A201")

# Convert factor variables to proper factors
factor_vars = c("V1", "V3", "V4", "V6", "V7", "V9", "V10", "V12", "V14", "V15", "V17", "V18")
gercred[factor_vars] <- lapply(gercred[factor_vars], factor)

# Get the names of the factor variables
fac_vars = vapply(gercred, is.factor, logical(1))
namfac = names(fac_vars)[fac_vars]

# Names of the numeric variables
num_vars = vapply(gercred, is.numeric, logical(1))
namnum = names(num_vars)[num_vars]

# Display a summary of the German Credit data set
summary(gercred)
```

```
##    V1           V2          V3           V4            V5           V6
##  A11:274   Min.   : 4.0   A30: 40   A43    :280   Min.   :  250   A61:603
```

```
##  A12:269   1st Qu.:12.0   A31: 49   A40    :234   1st Qu.: 1366   A62:103
##  A13: 63   Median :18.0   A32:530   A42    :181   Median : 2320   A63: 63
##  A14:394   Mean   :20.9   A33: 88   A41    :103   Mean   : 3271   A64: 48
##            3rd Qu.:24.0   A34:293   A49    : 97   3rd Qu.: 3972   A65:183
##            Max.   :72.0             A46    : 50   Max.   :18424
##                                     (Other): 55
##    V7            V8           V9        V10          V11          V12
##  A71: 62   Min.   :1.000   A91: 50   A101:907   Min.   :1.000   A121:282
##  A72:172   1st Qu.:2.000   A92:310   A102: 41   1st Qu.:2.000   A122:232
##  A73:339   Median :3.000   A93:548   A103: 52   Median :3.000   A123:332
##  A74:174   Mean   :2.973   A94: 92              Mean   :2.845   A124:154
##  A75:253   3rd Qu.:4.000                        3rd Qu.:4.000
##            Max.   :4.000                        Max.   :4.000
##
##       V13          V14         V15           V16           V17      V18
##  Min.   :19.00   A141:139   A151:179   Min.   :1.000   A171: 22   1:845
##  1st Qu.:27.00   A142: 47   A152:713   1st Qu.:1.000   A172:200   2:155
##  Median :33.00   A143:814   A153:108   Median :1.000   A173:630
##  Mean   :35.55                         Mean   :1.407   A174:148
##  3rd Qu.:42.00                         3rd Qu.:2.000
##  Max.   :75.00                         Max.   :4.000
##
##       V19             V20             V21
##  Min.   :0.000   Min.   :0.000   Min.   :0.0
##  1st Qu.:0.000   1st Qu.:1.000   1st Qu.:0.0
##  Median :0.000   Median :1.000   Median :0.0
##  Mean   :0.404   Mean   :0.963   Mean   :0.3
##  3rd Qu.:1.000   3rd Qu.:1.000   3rd Qu.:1.0
##  Max.   :1.000   Max.   :1.000   Max.   :1.0
##
```

### Version with dummies

```r
# load required libraries
library(fastDummies)
```

```
## Thank you for using fastDummies!
```

```
## To acknowledge our work, please cite the package:
```

```
## Kaplan, J. & Schlegel, B. (2023). fastDummies: Fast Creation of Dummy (Binary) Columns and Rows from Categor
```

```r
# Create dummy variables for the factors
gercreddum=dummy_cols(gercred, remove_first_dummy=TRUE, remove_selected_columns=TRUE)

# Now all variables are numeric.
# There are 48 covariates and 1 binary target "V21".
# summary(gercreddum)
```

### Train-test split

For the example, we create a training data set of size 600 and a test set of new data of size 400.

```r
# Splitting the data into a training (ntrain=600) and a test (ntest=400) set

# Set the seed for reproducibility
```

```r
set.seed(364565)

# Define the number of training and test samples
ntrain = 600
ntest = nrow(gercred) - ntrain

# Randomly select indices for the training set without replacement
indtrain = sample(1:nrow(gercred), ntrain, replace = FALSE)

# Create dummy variables for gercred data without the target variable (V21)
xdum = gercreddum # rename
xdum$V21 = NULL # target variable
xdum = as.matrix(xdum) # convert to matrix format

# Split the gercred data and dummy variables into training and test sets
gercredtrain = gercred[indtrain,]
gercredtest = gercred[-indtrain,]
gercreddumtrain = gercreddum[indtrain,]
gercreddumtest = gercreddum[-indtrain,]
gerxdumtrain = xdum[indtrain,]
gerxdumtest = xdum[-indtrain,]
```

## Logistic Regression Lasso Model

**Lasso:**

$$
\begin{aligned}
\hat{\beta} &= \arg\min_{\beta} \left\{ \sum_{i=1}^{n} (y_i - (\beta_0 + \beta^T x_i))^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\} \\
&= \arg\min_{\beta} ||Y - X\beta||_2^2 + \lambda ||\beta||_1
\end{aligned}
$$

**Elastic Net (likelihood-based)**

$$
\hat{\beta} = \arg\min_{\beta} \left\{ \frac{1}{n} \sum_{i=1}^{n} w_i \ell(y_i, (\beta_0 + \beta^T x_i))^2 + \lambda \left[ (1-\alpha) \frac{1}{2} \sum_{j=1}^{p} \beta_j^2 + \alpha \sum_{j=1}^{p} |\beta_j| \right] \right\}
$$

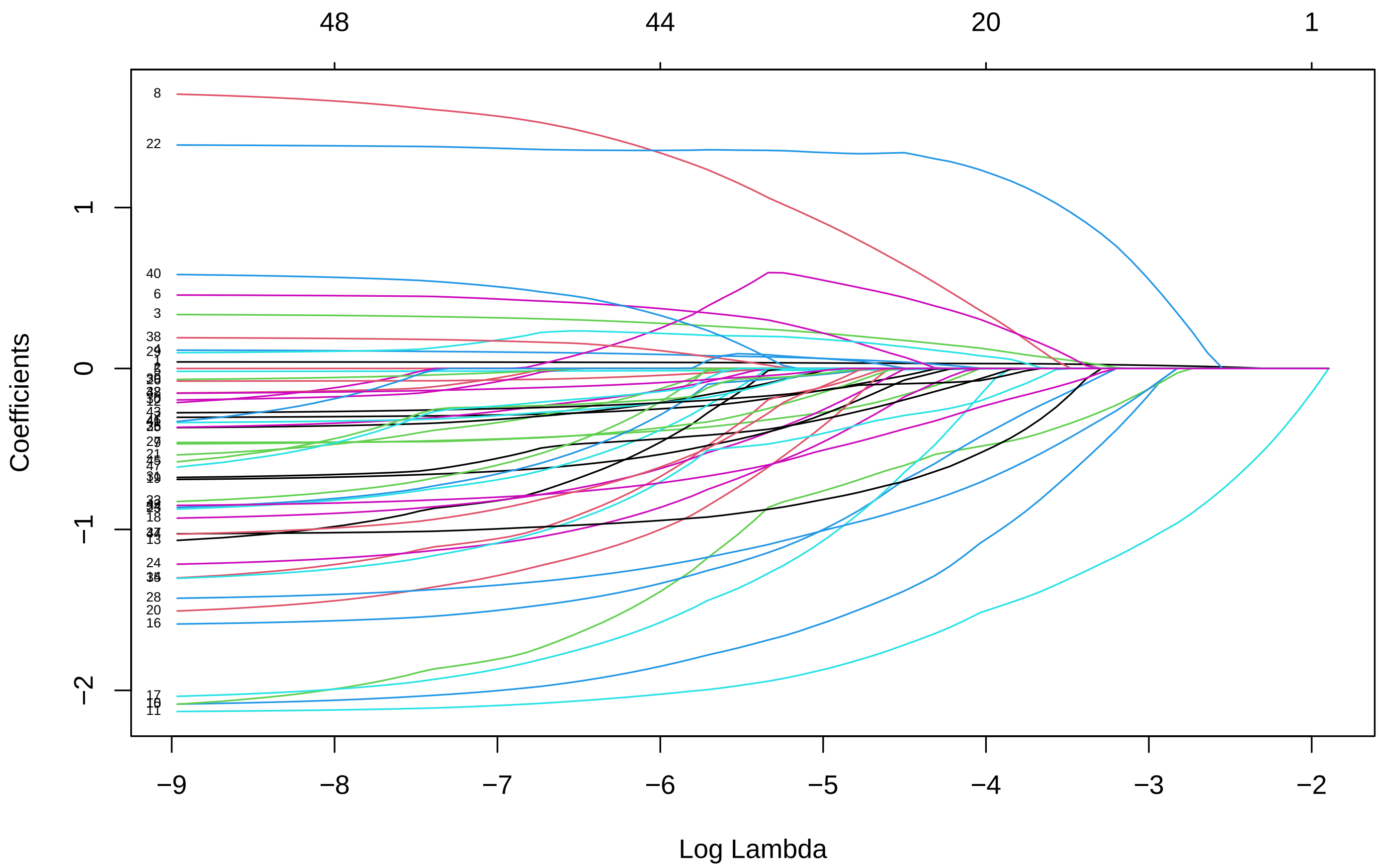
**Logistic Regression Elasticnet:**

Using the equation above:

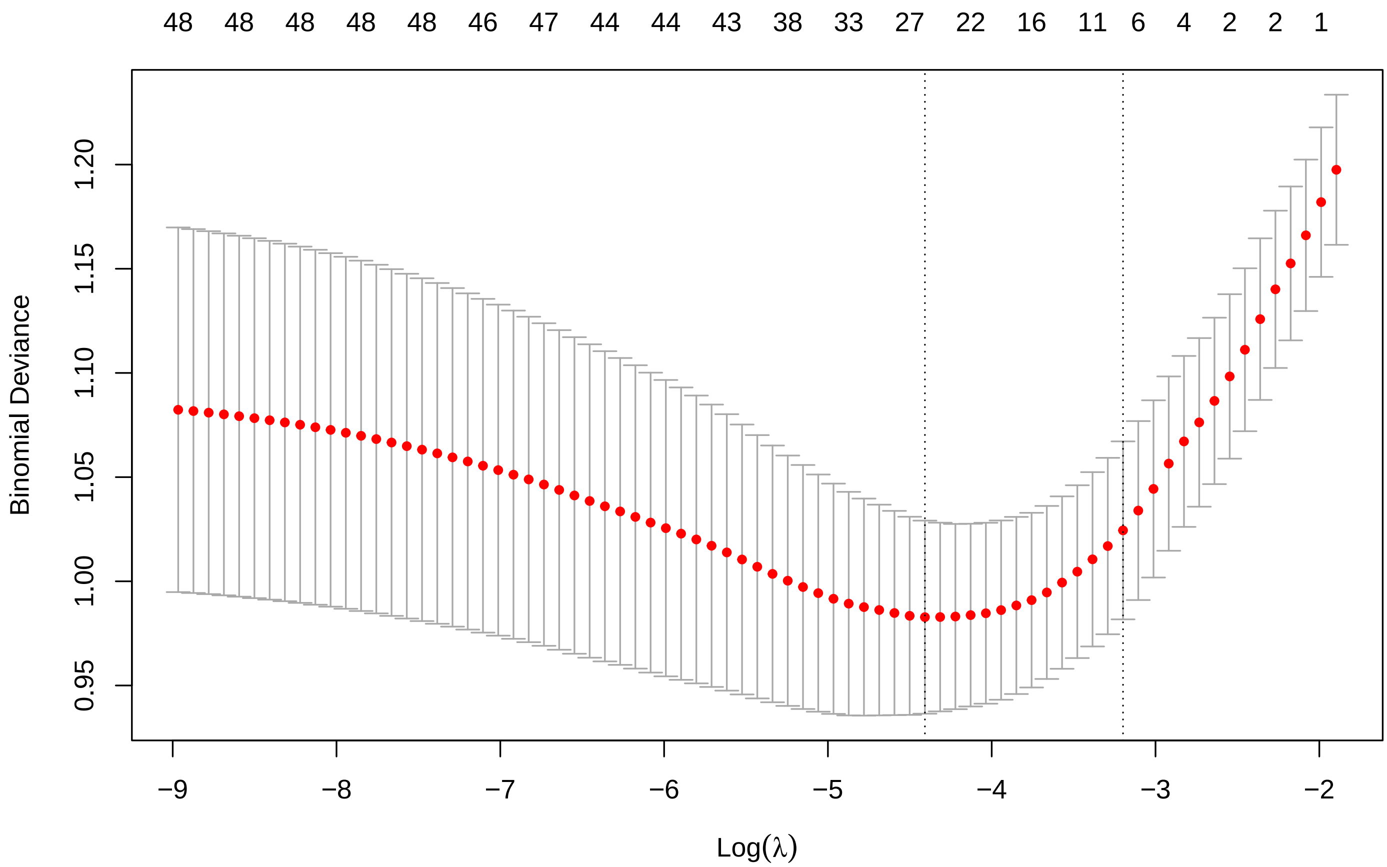
$$
\hat{\beta} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left[ y_i \log\left(1 + \exp(-\beta_0 - \beta^T x_i)\right) + (1 - y_i) \log\left( \frac{\exp(\beta_0 + \beta^T x_i)}{1 + \exp(\beta_0 + \beta^T x_i)} \right) \right] + \lambda \left[ (1-\alpha) \frac{1}{2} \sum_{j=1}^{p} \beta_j^2 + \alpha \sum_{j=1}^{p} |\beta_j| \right] \right\}
$$

- $\alpha = 0$: **Ridge**
- $\alpha = 1$: **Lasso**

```r
# Logistic regression with the lasso

# Set the seed for reproducibility
set.seed(162738)

# Load the necessary library
library(glmnet)

# Plot the lasso path with varying lambda values
plot(glmnet(gerxdumtrain,
            gercredtrain$V21,
            family="binomial", # binomial for logreg
            alpha=1),
     xvar = "lambda",
     label = TRUE)
```




```r
# Perform cross-validation to select the optimal lambda
cvgerlasso = cv.glmnet(gerxdumtrain, gercredtrain$V21, family="binomial", alpha=1)

# Plot the cross-validation results
plot(cvgerlasso)
```




```
# Get the coefficients for the optimal lambda
coeflassoger = predict(cvgerlasso, new=gerxdumtest, s="lambda.min", type="coefficients")

# Display the coefficients and count non-zero coefficients
coeflassoger
```

```
## 49 x 1 sparse Matrix of class "dgCMatrix"
##                lambda.min
## (Intercept) -1.454739e+00
## V2           3.182095e-02
## V5           4.126319e-05
## V8           1.650686e-01
## V11          3.332994e-02
## V13         -6.391242e-03
## V16          3.849864e-02
## V19         -2.446448e-02
## V20          5.901246e-01
## V1_A12      -1.356402e-01
## V1_A13      -1.335883e+00
## V1_A14      -1.683169e+00
## V3_A31       4.151325e-01
## V3_A32       .
## V3_A33       .
## V3_A34      -5.694585e-01
## V4_A41      -6.424560e-01
## V4_A410     -5.624105e-01
## V4_A42       .
## V4_A43      -4.904551e-02
## V4_A44       .
```

```
## V4_A45         .
## V4_A46         1.322736e+00
## V4_A48         .
## V4_A49        -1.339864e-01
## V6_A62         .
## V6_A63         .
## V6_A64         .
## V6_A65        -8.433750e-01
## V7_A72         1.246955e-01
## V7_A73         .
## V7_A74        -1.695024e-01
## V7_A75         .
## V9_A92         .
## V9_A93         .
## V9_A94        -2.781214e-01
## V10_A102       .
## V10_A103      -6.721503e-01
## V12_A122       .
## V12_A123       .
## V12_A124       .
## V14_A142       .
## V14_A143      -3.515876e-01
## V15_A152      -9.237637e-02
## V15_A153       .
## V17_A172       .
## V17_A173       .
## V17_A174       .
## V18_2          .
```

```r
length(coeflassoger[coeflassoger[,1] != 0 ,])
```

```
## [1] 26
```

We see that variable selection was peformed.

```r
# Make predictions on the test set using the selected lambda
predlassoger = predict(cvgerlasso, new=gerxdumtest, s="lambda.min", type="response")

# Display the first 10 predicted values
predlassoger[1:10]
```

```
##  [1] 0.14378795 0.14396077 0.06977632 0.37842897 0.35945215 0.50497267
##  [7] 0.46602492 0.43738247 0.31481853 0.01000166
```

- The lasso keeps 26 covariates (plus the intercept) out of the 48.
- The **predictions** are the **probabilities** of being a bad risk.
- Need a **threshold**.

We will use a function to estimate the threshold $c$ which maximizes the **gain matrix**

$$G = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix} = \begin{pmatrix} TP & FN \\ FP & TN \end{pmatrix}$$

$$\begin{aligned} \max \ & \mathbb{P}(\hat{y}=1, y=1) \times g_{11} + \mathbb{P}(\hat{y}=1, y=0) \times g_{12} \\ & + \mathbb{P}(\hat{y}=0, y=1) \times g_{21} + \mathbb{P}(\hat{y}=0, y=0) \times g_{22} \end{aligned}$$

### Cross-validated probabilities

First, we create a function to obtain cross-validated probabilities from `glmnet`. - The best $\lambda$ is chosen by CV in each fold. - The output can be used to find the best threshold afterwards.

```r
# Function to get cross-validated estimated probabilities from glmnet
# The best lambda is chosen by CV in each fold
# The output can be used to find the best threshold afterwards

predcvglmnet = function(xtrain, ytrain, k = 10, alpha = 1)
{
  # xtrain = matrix of predictors
  # ytrain = vector of target (0-1)
  # k = number of folds in CV
  # alpha = alpha parameter in glmnet

  # Load the necessary library
  library(glmnet)

  # Set a seed for reproducibility
  set.seed(375869)

  # Get the number of observations in the training data
  n = nrow(xtrain)

  # Initialize the vector to store predicted probabilities
  pred = rep(0, n)

  # Create a random permutation of indices
  per = sample(n, replace = FALSE)

  # Initialize indices for the current fold
  tl = 1

  # Perform k-fold cross-validation
  for (i in 1:k)
  {
    # Determine the upper index for the current fold
    tu = min(floor(tl + n / k - 1), n)

    # Adjust for the last fold
    if (i == k)
    {
      tu = n
    }

    # Get the current indices for the fold
    cind = per[tl:tu]

    # Fit a glmnet model with cross-validation on the current fold
    fit = cv.glmnet(xtrain[-cind, ], ytrain[-cind], family = "binomial", alpha = alpha)

    # Predict the probabilities for the current fold using lambda.min
    pred[cind] = predict(fit, new = xtrain[cind, ], s = "lambda.min", type = "response")

    # Update the starting index for the next fold
    tl = tu + 1
  }

  # Return the predicted probabilities
```

```r
  pred
}
```

Note that here there are two levels of cross validation:

1. The **outer CV** is used to compute **estimated probabilities**.
2. The **innter CV** estimates the tunning parmaeter for a given fold of the outer CV.

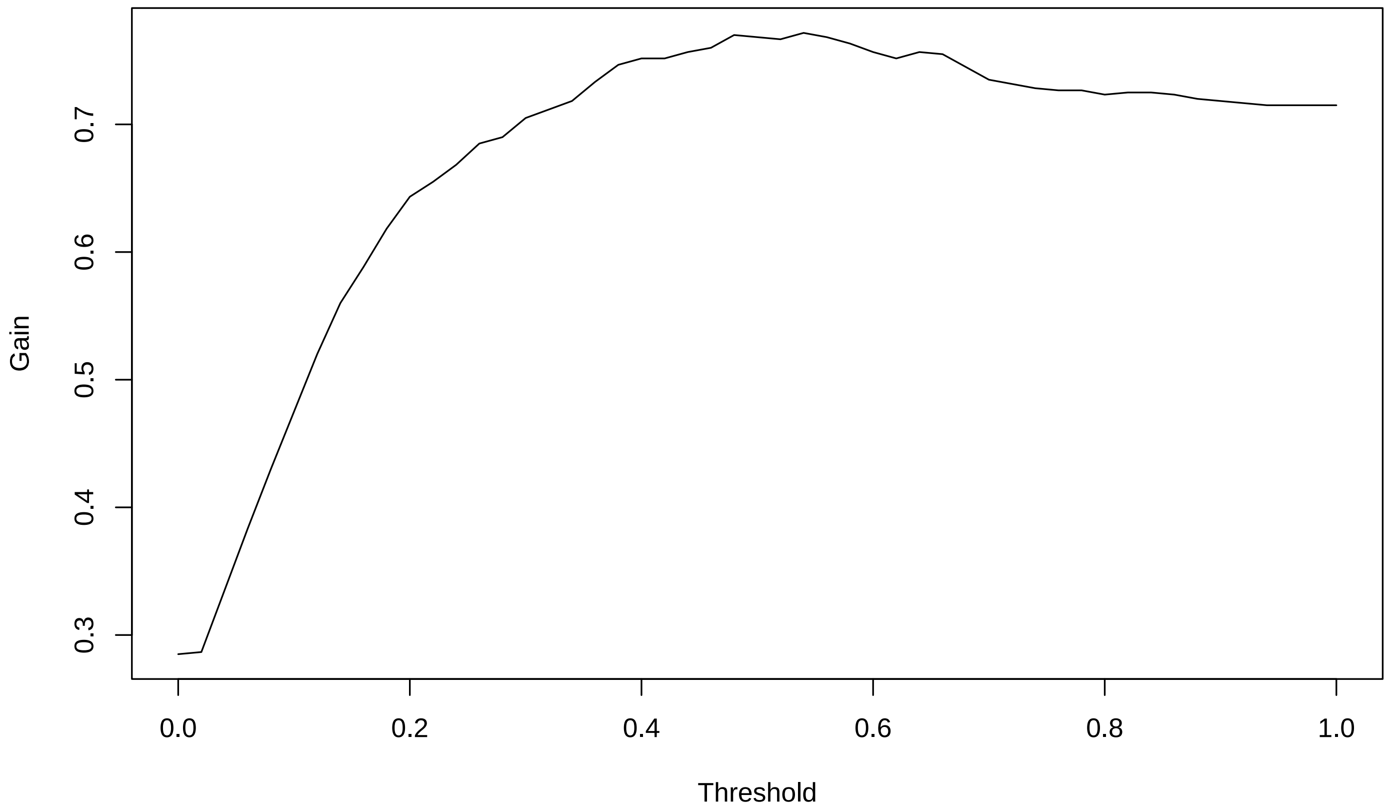
**Best Binary Classifier Threshold**

```r
# Function to find the best threshold to use for a
# binary classifier with respect to a gain matrix

bestcutp = function(predcv, y, gainmat = diag(2), cutp = seq(0, 1, .02), plotit = FALSE)
{
  # predcv = vector of predicted probabilities (e.g., obtained out-of-sample by CV)
  # y = vector of target labels (0 or 1)
  # gainmat = gain matrix (2x2) (we want to maximize the gain)
  #   (1,1) = gain if pred=0 and true=0
  #   (1,2) = gain if pred=0 and true=1
  #   (2,1) = gain if pred=1 and true=0
  #   (2,2) = gain if pred=1 and true=1
  # cutp = vector of thresholds to try
  # plotit = whether to plot the results

  # Initialize variables
  nc = length(cutp)   # Number of thresholds to evaluate
  gain = rep(0, nc)   # Vector to store calculated gains

  # Loop through each threshold value
  for (i in 1:nc)
  {
    pred = as.numeric(predcv > cutp[i])  # Predicted binary outcomes using the threshold
    gain[i] = mean(gainmat[1, 1] * (pred == 0) * (y == 0) +  # Calculate gain for this threshold
                   gainmat[1, 2] * (pred == 0) * (y == 1) +
                   gainmat[2, 1] * (pred == 1) * (y == 0) +
                   gainmat[2, 2] * (pred == 1) * (y == 1))
  }

  # Optionally plot the gains over different thresholds
  if (plotit)
  {
    plot(cutp, gain, type = "l", xlab = "Threshold", ylab = "Gain")
  }

  # Create a list containing results
  out = list(NULL, NULL)
  out[[1]] = cbind(cutp, gain)               # Matrix of thresholds and associated gains
  out[[2]] = out[[1]][which.max(gain),]     # Threshold with the maximum gain and its associated mean gain
  out
}
```

**Finding the optimal cutoff using the gain matrix**

```r
# Computing CV estimated probabilities with glmnet
set.seed(16274)
```

```r
pred = predcvglmnet(gerxdumtrain, gercredtrain$V21, k = 10, alpha = 1)

# Estimating the best threshold with the identity gain matrix
# This step is intended to find the threshold that maximizes gain
res = bestcutp(pred, gercredtrain$V21, gainmat = diag(2),
               cutp = seq(0, 1, .02), plotit = TRUE)
```




```r
# Display the threshold with the associated mean gain
res[[2]]
```

```
##      cutp      gain
## 0.5400000 0.7716667
```

**COmputing the good classification rate**

```r
# using the best threshold to obtain the predictions
predlassoger01=as.numeric(predlassoger>res[[2]][1])

# good classification rate on the test set
mean(gercredtest$V21==predlassoger01)
```

```
## [1] 0.6975
```

```r
# a naive rule would get a good classification rate of
max(mean(gercredtest$V21),1-mean(gercredtest$V21))
```

```
## [1] 0.6775
```

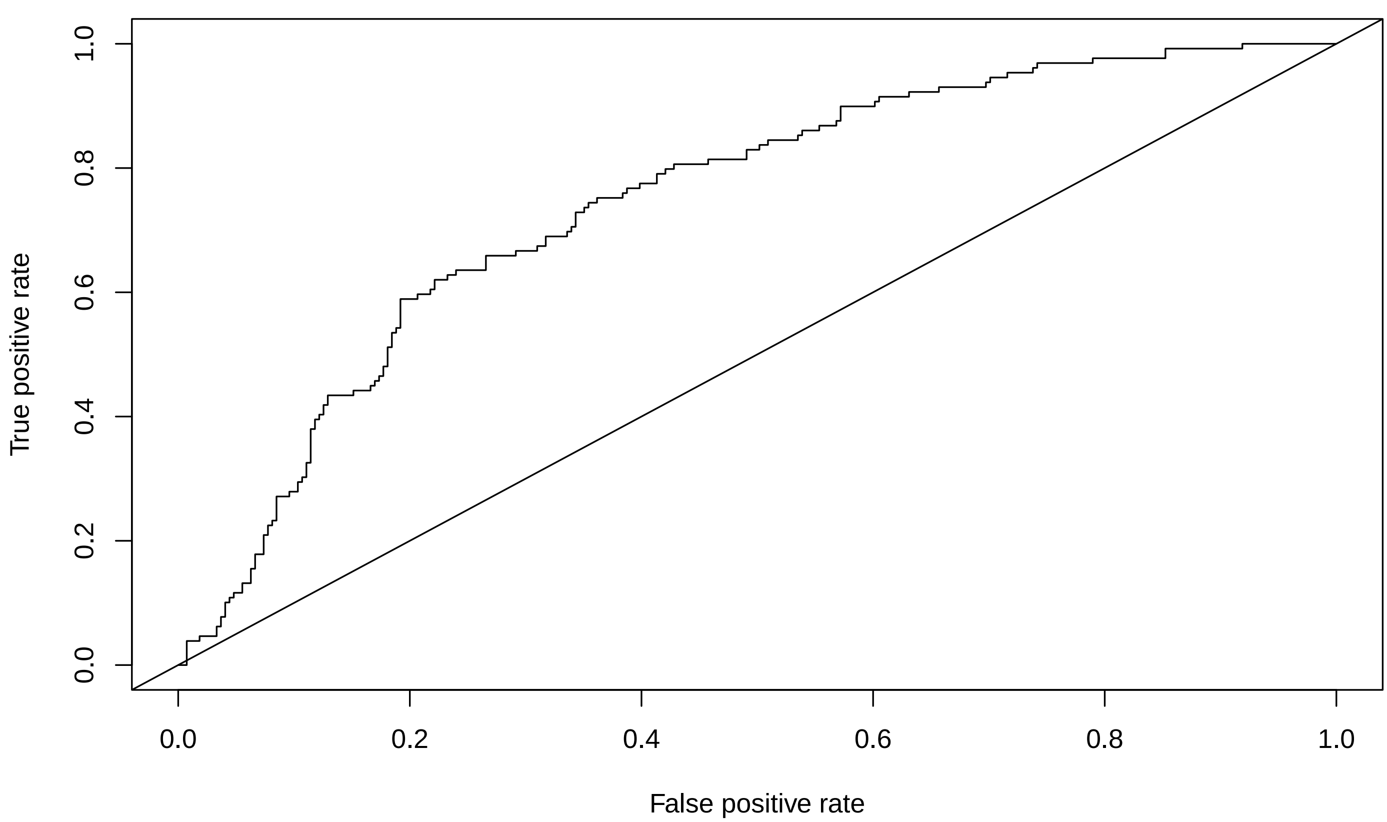
We obtain a true good classification rate of 0.6975 on the test set, and a **naive rule**, assigning everyone to the 0 class would get a good classification rate of 0.677.

Hence, the lasso logistic regression performs only a little better compared to the naive rule.

## AUC and ROC Curves

- The **ROC curve** is a plot of the **true positive rate** (TPR) against the **false positive rate** (FPR) for the different possible thresholds.
- The **AUC** is the area under the ROC curve.

```r
# Load the ROCR library for ROC curve analysis
library(ROCR)

# Create a prediction object using predicted probabilities and true values
predrocr = prediction(predlassoger, gercredtest$V21)

# Calculate the ROC curve
roc = performance(predrocr, "tpr", "fpr")

# Plot the ROC curve
plot(roc)

# Add a diagonal reference line for a random classifier
abline(a = 0, b = 1)
```



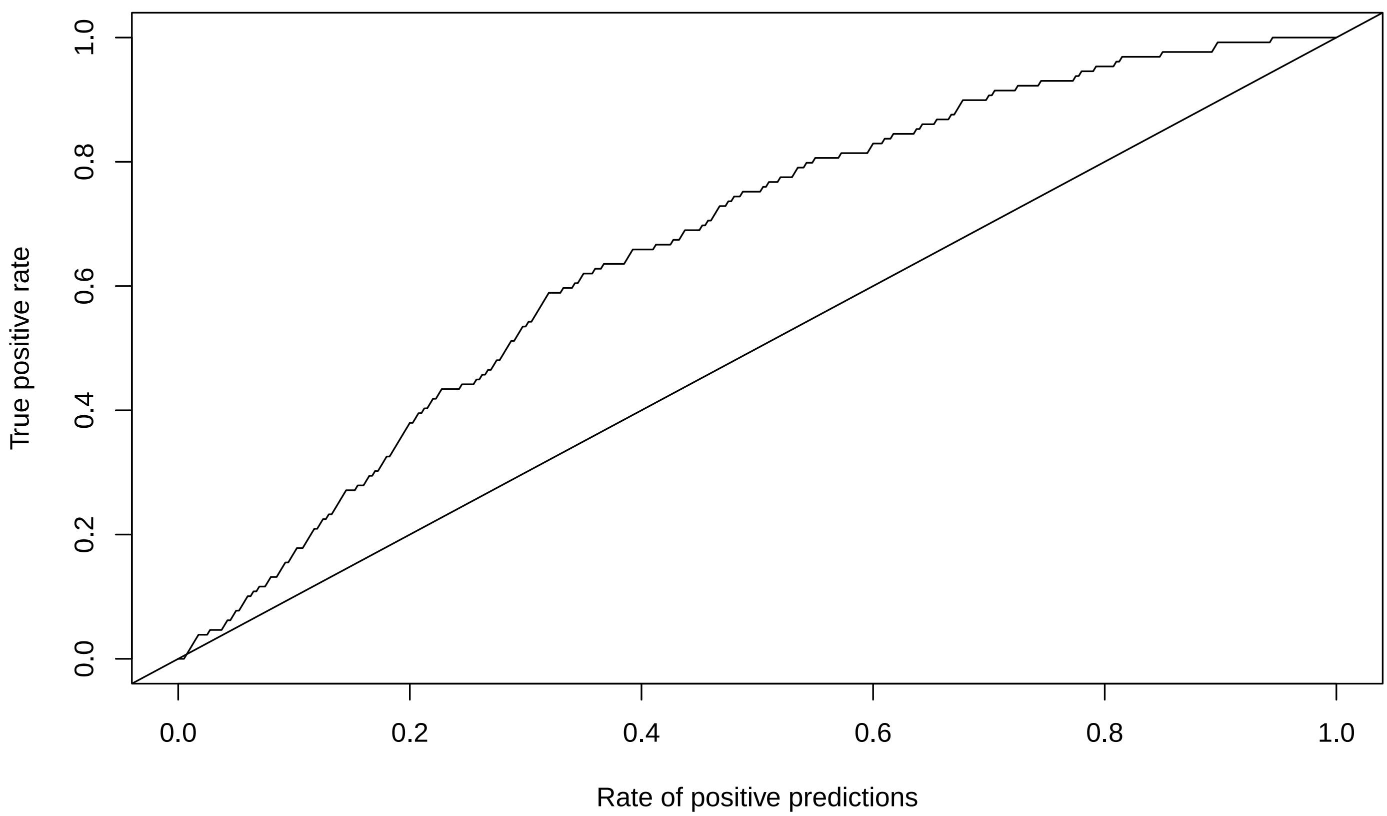
```r
# Calculate and display the AUC (Area Under the Curve)
performance(predrocr, "auc")@y.values[[1]]
```

```
## [1] 0.746074
```

- Note that we used the test data here but in practice, the value of Y would not be available for the new data.
- Hence, we would need to compute the ROC curve, AUC and lift chart with the training data.
- In that case, we must remember to use **proper estimation of the probabilities** (like the ones obtained by CV above), in order to get honest estimates.

#### lift curve

```r
# Calculate and plot the lift chart
lift1 = performance(predrocr, "tpr", "rpp")

# Plot the lift chart
plot(lift1)

# Add a diagonal reference line for a random classifier
abline(a = 0, b = 1)
```



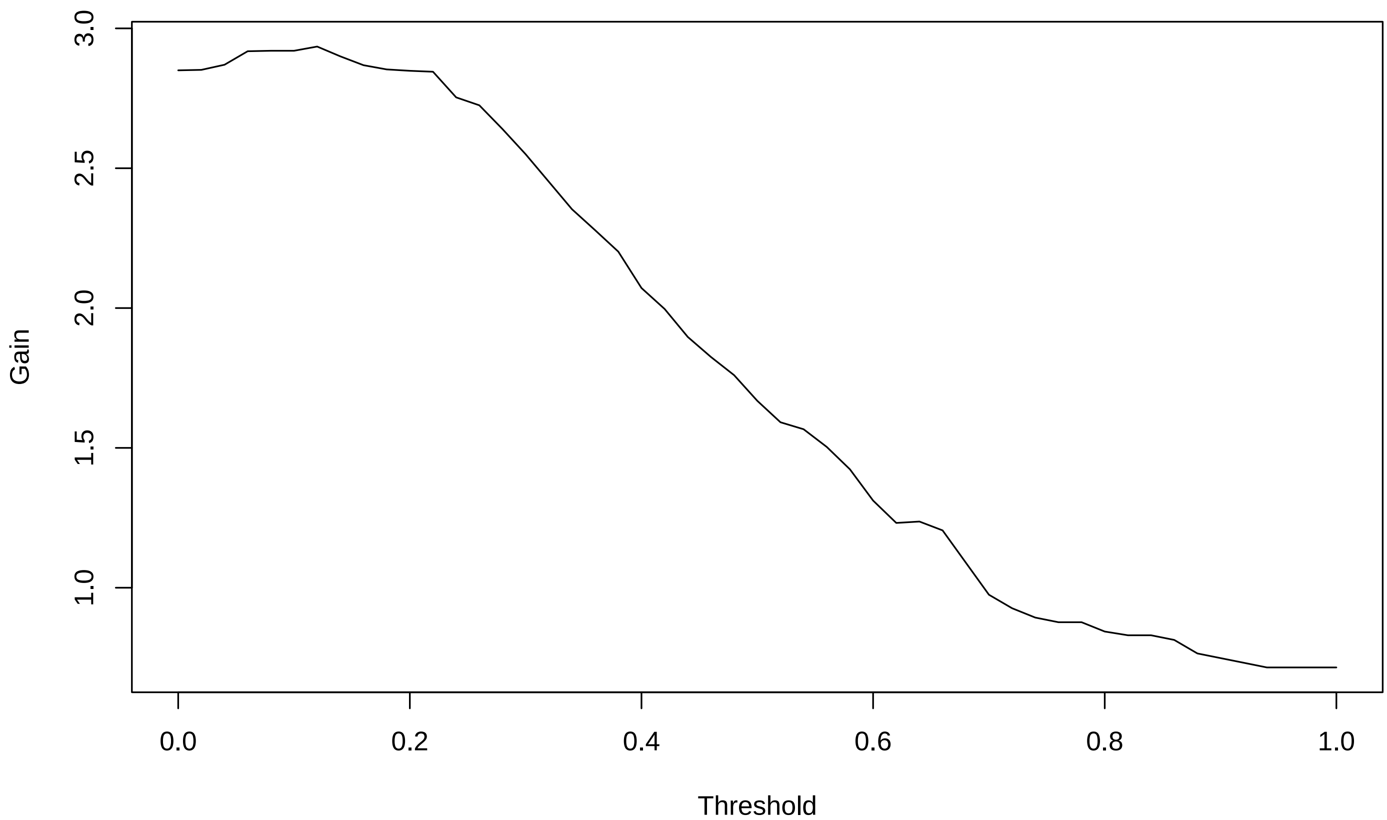

## Customizing the gain matrix

Instead of using the identity gain matrix, we can use a custom gain matrix to reflect the fact that the cost of a false positive is not the same as the cost of a false negative.

Ex.

$$G = \begin{pmatrix} 1 & 0 \\ 0 & 10 \end{pmatrix}$$

```r
# Set the random seed for reproducibility
set.seed(18965)

# Estimate the best threshold with a gain matrix that favors detecting bad risks
res1 = bestcutp(pred, gercredtrain$V21, plotit = TRUE, gainmat = rbind(c(1,0), c(0,10)))
```




```r
# Display the threshold with the associated mean gain
res1[[2]]
```

```
##  cutp  gain
## 0.120 2.935
```

In this case the optimal threshold is clearly much slower, since we now want to **classify more people as bad risks** because the reward is higher if we are right.

```r
# Function to compute the C-index with a binary target
cindexbasic=function(phat,y)
{
n=length(phat)
cc=0
npair=0
for(i in 1:(n-1))
    {
```

```r
    for(j in (i+1):n)
        {
        if(y[i]!=y[j])
            {
            cc=cc+(phat[i] > phat[j])*(y[i]>y[j])+(phat[i] < phat[j])*(y[i]<y[j])+ 0.5*(phat[i]==phat[j])
            npair=npair+1
            }
        }
    }
cc/npair
}

# We get the same value as the AUC
cindexbasic(predlassoger,gercredtest$V21)
```

```
## [1] 0.746074
```

And we see that it ise indeed the same as the AUC.