Assignment 3: Binary Classification

Ian Wallgren, Wanchang Zhang, Lavinia Hriscu, Victor Jimenez

In this assignment we are asked to implement three different classification methods for binary categorical data. We will work with a SPAM e-mail database that contains the frequency of different words and characters on every message and a binary target value with its true label: SPAM or not SPAM.

Upon new e-mails, our predictor should be able to automatically decide whether to send them to the SPAM box or not. The specificity and sensitivity of the different rules will be compared.

Reading the SPAM e-mail database

The code for reading the data was provided by the professor. The loaded database contains 4601 e-mails among which 39% are SPAM.

```
## n = 4601, p = 57
```

```
##
## Proportion of spam e-mails = 0.39
```

2. Generating training and test sets

The target binary variable is `spam.01`. We will divide the database in a way that 1/3 of emails tagged as SPAM and 1/3 of emails tagged as not SPAM will be used as a test set. The remaining data will be used as a test set.

```
# For SPAM emails:
spam.1 = which(spam$spam.01 == 1)
n.1 = length(spam.1)
train.ind.1 = sample(1:n.1, size = round(2/3 * n.1), replace = FALSE)
test.ind.1 = setdiff(1:n.1, train.ind.1)

# For non-SPAM emails:
spam.0 = which(spam$spam.01 == 0)
n.0 = length(spam.0)
train.ind.0 = sample(1:n.0, size = round(2/3 * n.0), replace = FALSE)
test.ind.0 = setdiff(1:n.0, train.ind.0)

# Now we combine:
train.ind = c(train.ind.1, train.ind.0)
test.ind = c(test.ind.1, test.ind.0)

# Check:
length(train.ind) + length(test.ind)
```

```
## [1] 4601
```

```
100*length(train.ind)/(length(train.ind) + length(test.ind))
```

```
## [1] 66.68116
```

For convenience, we will work with sub-matrices of the dataset containing the training and the test observations.

```
# Train dataset
Xtr = as.matrix(spam[train.ind, -(p+1)])
ytr = as.matrix(spam[train.ind, p+1])
# Test dataset
Xte = as.matrix(spam[test.ind, -(p+1)])
yte = as.matrix(spam[test.ind, p+1])
```

3. Classification of the data

We will use three different classification methods to classify the e-mails. As a binary variable, $y|X_i = x_i$ can be described by a Bernoulli process with probability of success $E(y|X_i = x_i)$. The methods described in this assignment are based in the estimation of the probability of the e-mail to be considered SPAM, or equivalently:

$$g_S(x) = \hat{Pr}(y = 1|X = x)$$

For each method, a classification rule will be determined by the cut point parameter $c \in [0, 1]$:

$$h_S(x) = \begin{cases} 0 & \text{if } g_S(x) \leq c \\ 1 & \text{if } g_S(x) > c \end{cases}$$

3.1 Logistic regression — IRWLS

The first classification will be performed by fitting the data to a logistic model using iteratively re-weighted least squares method. The function performing `IRWLS` has been provided by the professor. This function trains the model with the inputs `x` (explanatory) and `y` (target) and estimates the $E(y = 1|X = x)$ for the observations `x.new`, providing as an output g_S(`x.new`).

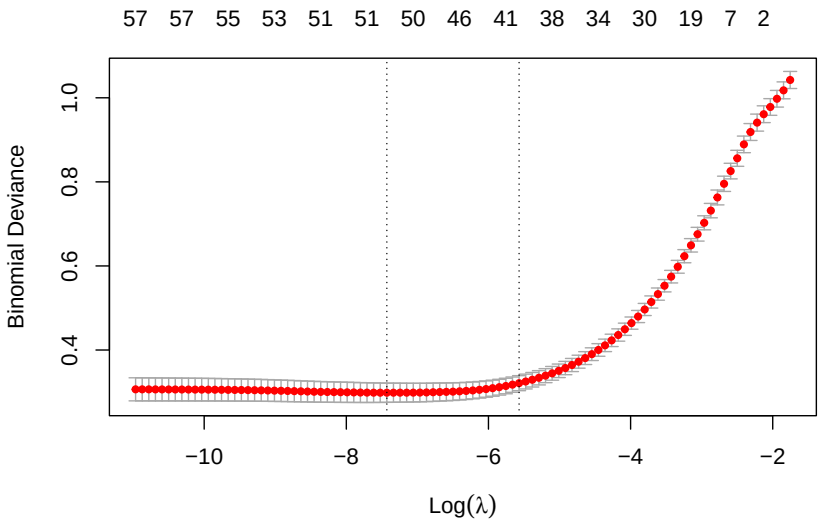
```
class.IRWLS = logistic.IRWLS(x=Xtr, y=ytr, weights.out=1, max.iter=10, eps.beta=1e-5)
```

3.2 Logistic regression — Lasso

A fit to the logistic model can also be performed by lasso, by means of the `glmnet` function with `alpha=1`. First, we will perform cross-validation to determine the best value of the shrinkage parameter λ_{min}, and then fit the model with it.

```
cv.lasso = cv.glmnet(Xtr, ytr, alpha=1, family='binomial') # cross-validation
class.lasso = glmnet(Xtr, ytr, alpha=1, family = "binomial", lambda = cv.lasso$lambda.min)
cv.lasso$lambda.min
```

```
## [1] 0.0005933267
```

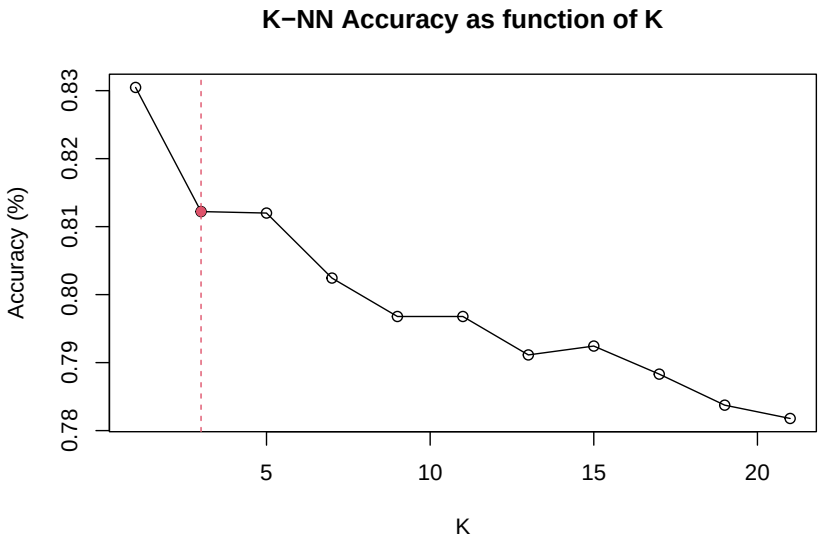


3.3 Binary regression — K-NN

Finally, the k-nearest neighbors classification method will be tested, by means of the function `knn` of the library class. We will first perform cross-validation with knn.cv to determine the reach of the classification of the points. The best k will be chosen so to provide the maximum accuracy of the model $A = Pr(1|1) + Pr(0|0)$; that is, the maximum proportion of correct classifications in the training set .

In order to minimize ties, it's better to use an odd number of K-nearest neighbors, that is why a sequence between 1 and 21, with steps of 2, is tried in this case.

```
kval <- seq(1,21,2)
knn.accuracy = numeric(length(kval))
i <- 1
for (k in kval){
  knn.cv.k = knn.cv(spam[,-ncol(spam)], cl=spam$spam.01, k = k)
  conf_matrix = table(knn.cv.k, spam$spam.01)
  knn.accuracy[i] = sum(diag(conf_matrix))/sum(conf_matrix)
  i <- i+1
}
```



We will discard the value $K = 1$, as it can lead to overfitting.

Now we can perform the classification. In contrast with the former example, we are required to demand `prob=TRUE` so that the probability for the classification of each point is provided. In order to obtain $g_S(x)$, which is the probability of the classification as SPAM, we will have to operate for the classified as non-SPAM:

$$\hat{P}(Y = 1|X = x) = 1 - \hat{P}(Y = y|X = x)\Big|_{y=0}$$

```
class.knn = knn(Xtr, Xte, ytr, k = knn.cv.kbest, prob=TRUE)
prob.knn = attr(class.knn, "prob") # P(Y=y|X=x)
class.knn = as.numeric(class.knn)-1 # encode it to 0/1

ypre.knn = prob.knn
ypre.knn[which(class.knn == 0)] = 1 - prob.knn[which(class.knn == 0)]
```

```
##     Pr(Y=y)   Pr(Y=1) k-NN class. True class.
## 1 0.6666667 0.3333333           0           1
## 2 0.6666667 0.3333333           0           1
## 3 0.6666667 0.3333333           0           1
## 4 0.6666667 0.3333333           0           1
## 5 0.5000000 0.5000000           0           1
## 6 0.6666667 0.3333333           0           1
```

4. ROC curve for each method

In order to provide metrics and compare the behavior of the algorithms, we will use the test subset (`Xte` and `yte`) to obtain the cross-table between the real and the predicted classifications, also known as confusion matrix M. Then, the proportion of false positives $(1|0)$ $M_{01}/(M_{00} + M_{01})$ or *precision (P)* against the proportion of true positives $(1|1)$ $M_{11}/(M_{10} + M_{11})$ or *recall (R)* will be plotted by varying the value of c, so that the behavior of the method under different rules can be obtained. The ROC curve can be expressed as follows:

$$\{\mathrm{P}(c), \mathrm{R}(c) : c \in [0, 1]\}$$

If the model was perfect, $c = 0.5$ would imply $P = 0$ and $R = 1$, and any value above or below that would make the recall to decrease or the precision to increase, respectively. The best method will thus be that one closer to the ideal ROC, that has area under the curve (AUC) equal to one.

We will plot the ideal curve over the predicted ROC for each of our methods. We will use the `pROC` package, which plots instead the *specificity* (1-P) and the *sensitivity* (R) of the method.

4.1 Logistic regression — IRWLS

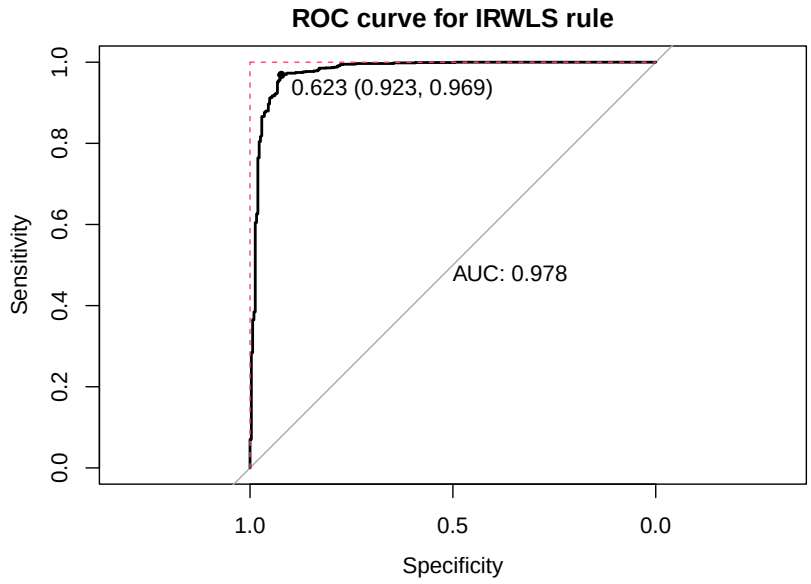
We will use the same function as before, but now also providing the explanatory set of test observations as `x.new=Xte`. The regression model will be fitted with the training data and the predicted values `ypre.IRWLS` will be compared with the true classification of the test set to provide the metrics.

```
ypre.IRWLS = logistic.IRWLS(x=Xtr, y=ytr, x.new=Xte, weights.out=1, max.iter=10, eps.beta=1e-5)
IRWLS.roc = roc(as.numeric(yte), ypre.IRWLS$predicted.values)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot(IRWLS.roc, print.thres = "best", print.auc = TRUE, main = "ROC curve for IRWLS rule", xlim=c(1,0),
segments(1,0,1,1, col=2, lty=2)
segments(1,1,0,1, col=2, lty=2)
```



The method seems to perform slightly better at $c < 1/2$, as the proportion of false positives does not reach zero unless c is way bigger than $1/2$, which indicates that the model tends to over-estimate the probability of the e-mail to be classified as SPAM. In spite of this slight bias, the classification is quite correct, as $AUC = 0.978$ is close to one.

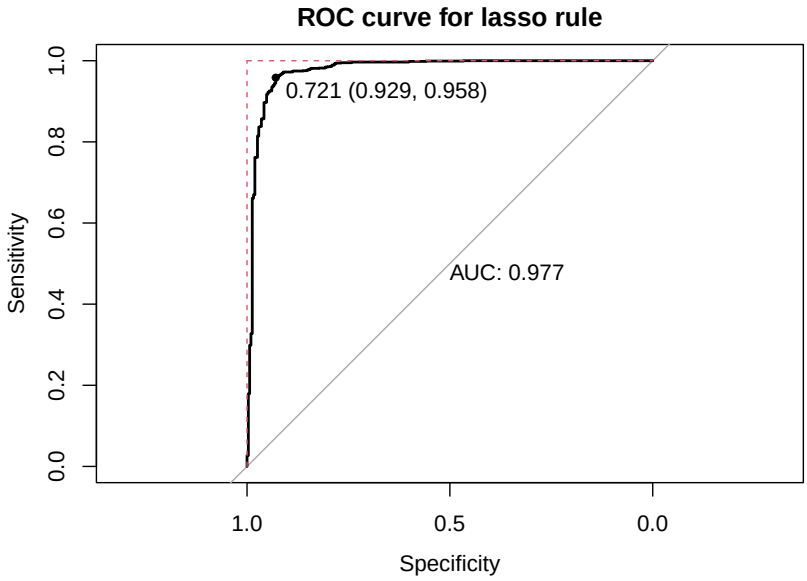
4.2 Logistic regression — Lasso

We will use the `predict` function with `newx=Xte` to obtain the predicted probabilities. The ROC curve will be plotted as before.

```
ypre.lasso = predict(class.lasso, newx=Xte, s=class.lasso$lambda.min, type='response')
lasso.roc = roc(as.numeric(yte), ypre.lasso)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```



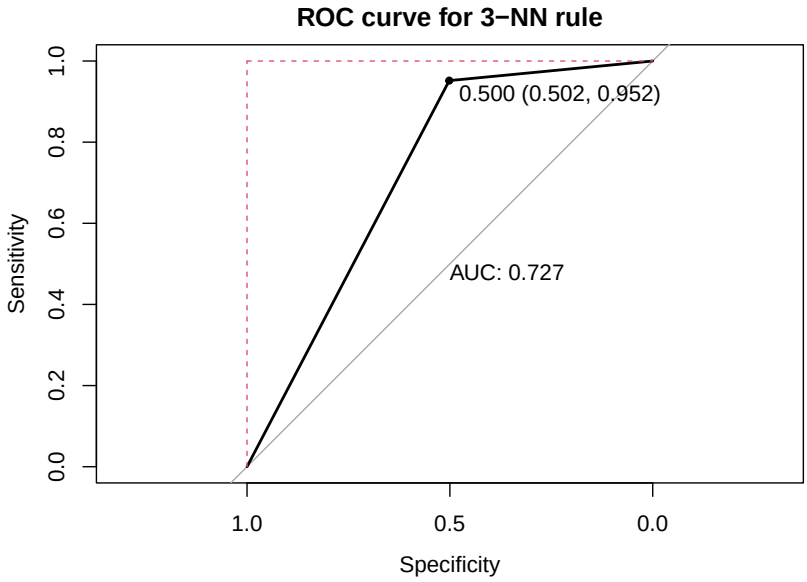
As observed in the IRWLS regression, the classification seems to be somewhat biased towards the SPAM label. Nevertheless, $AUC = 0.977$, only slightly below the value of the last method.

4.3 Binary regression — K-NN

The estimated probabilities of the k-NN method were already computed.

```
knn.roc = roc(as.numeric(yte), as.numeric(class.knn))
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```



In contrast with the previous two methods, five-nearest-neighbors classification does not seem to provide adequate results on the test set. $AUC = 0.727$ is well below the ideal value, as the model seems to over-predict

SPAM emails.

5. Classification metrics for $c = 1/2$

In order to apply the classification rule $h_S(x)$ with $c = 1/2$ we will define a round function that takes every value below c to zero and every value above or equal to c to one.

```
round2 = function(x, digits) {
  posneg = sign(x)
  z = abs(x)*10^digits
  z = z + 0.5 + sqrt(.Machine$double.eps)
  z = trunc(z)
  z = z/10^digits
  z*posneg
}
round2(c(0.49999, 0.50000, 0.50001), 0)
```

```
## [1] 0 1 1
```

Then, the predicted classification `ypre.?` over the test dataset will be compared with the real labels `yte` and some metrics will be provided:

- The confusion matrix M, as the cross-table between `ypre.?` and `yte`.
- The misclassification rate, as the anti-trace of M divided by the number of observations.
- The F-score of the classification, defined as the harmonic mean between the precision and recall of the method.

```
metrics = function(y_pred, y_test){
  conf_matrix = table(y_pred, test=y_test)
  print(conf_matrix)

  precision = conf_matrix[1,1]/sum(conf_matrix[,1])
  recall = conf_matrix[1,1]/sum(conf_matrix[1,])
  f_score = 2*precision*recall/(precision+recall)

  cat("\n")
  cat("Precision: ", precision, "\n")
  cat("Recall: ", recall, "\n")
  cat("F-score: ", f_score, "\n")

  misscl = sum(diag(conf_matrix[nrow(conf_matrix):1, ]))/length(y_test)
  cat("Misclassification error: ", misscl*100, "% \n")
}
```

5.1 Logistic regression — IRWLS

```
ypre.IRWLS.bin = round2(ypre.IRWLS$predicted.values,0)
metrics(ypre.IRWLS.bin, yte)
```

```
##       test
## y_pred    0    1
##      0  273   31
##      1   38 1191
##
```

```
## Precision:  0.8778135
## Recall:  0.8980263
## F-score:  0.8878049
## Misclassification error:  4.500978 %
```

5.2 Logistic regression — Lasso

```
ypre.lasso.bin = round2(ypre.lasso,0)
metrics(ypre.lasso.bin, yte)
```

```
##       test
## y_pred    0    1
##      0  266   30
##      1   45 1192
##
## Precision:  0.8553055
## Recall:  0.8986486
## F-score:  0.8764415
## Misclassification error:  4.892368 %
```

5.3 Binary regression — K-NN

```
ypre.knn.bin = round2(ypre.knn,0)
metrics(ypre.knn.bin, yte)
```

```
##       test
## y_pred    0    1
##      0  145   55
##      1  166 1167
##
## Precision:  0.4662379
## Recall:  0.725
## F-score:  0.5675147
## Misclassification error:  14.41618 %
```

By looking at these metrics we can claim that logistic regression (both by IRWLS and lasso) provides an acceptable classification with an error rate of around 5%, whereas k-NN method is not suitable for this problem, probably due to the high-dimensionality of the dataset.

Regarding the differences between IRWLS and lasso regression, we already showed that both methods are slightly biased towards the classification as SPAM. With this particular test dataset, $AUC_{IRWLS} > AUC_{lasso}$. Even if the differences are small, IRWLS has the highest F-score and is shown to provide the lowest misclassification error.

6. Compute l_{val} for each rule

A different way to evaluate a classification rule is by computing the log-likelihood function of the joint distribution of the n binomial random variables corresponding to the observed data (x_i, y_i), divided by n. If a validation set is available, the expression to calculate would be:

$$l_{val}(g_S) = \frac{1}{m}\sum_{j=1}^{m}\left(y_j^V \log g_s(x_j^V) + (1-y_j^V)\log(1-g_s(x_j^V))\right)$$

We will apply the following function to our `ypre.?` vectors, which correspond to $\log g_s(x_j^V)$. The main idea of the log-likelihood for a probabilistic model for binary classification is to sum up the logs of the predicted probabilities where the real response value is one, and add this to the sum of the logs(1-probabilities) when the real response was zero. In order to implement this idea, the first part of the formula is applied to the positions where `yte = 1` (i.e. which correspond to `y_j^V = 1`), and the second one to the parts where `yte = 0`.

```
lval = function(y_val, y_pre){
  m = length(y_val)
  yval.ind <- which(yte==1)
  mlval <- sum(log(y_pre[yval.ind])) + sum(log(1-y_pre[-yval.ind]))
  return(mlval/m)
}
```

6.1 Logistic regression — IRWLS

```
lval.IRWLS = lval(yte, ypre.IRWLS$predicted.values)
lval.IRWLS
```

```
## [1] -0.138235
```

6.2 Logistic regression — Lasso

```
lval.lasso = lval(yte, ypre.lasso)
lval.lasso
```

```
## [1] -0.1379339
```

6.3 Binary regression — K-NN

```
lval.knn = lval(yte, ypre.knn)
lval.knn
```

```
## [1] -Inf
```

As seen before, the result of `knn` differs of the two other algorithms and this is reflected in the value of the log-likelihood. The fact that l_{val} is computed to be `-Inf` in this case is because there are some observations x_i with true value $y_i = 1$ which are being predicted with $g_S(x_i) \approx 0$, and equivalently for $y_i = 0$. Then the function calculates the logarithm of zero, which goes to `-Inf`.

```
sum(yte[which(ypre.knn<0.0000001)])
```

```
## [1] 8
```

```
err.1 = yte[which(ypre.knn>0.9999999)]
length(err.1) - sum(err.1)
```

```
## [1] 71
```