

Entregable: Algoritmos básicos de aprendizaje

Laura Rodriguez Navas

Febrero 2020

Los Datos

El fichero glass.data incluye el estudio de la clasificación de los tipos de vidrio con 6 variables explicativas. El estudio fue motivado por una investigación criminológica. En la escena de un crimen, el vidrio que queda puede usarse como evidencia ... ¡si se identifica correctamente!

Las variables que se guardaron son:

1. Id number: 1 to 214
2. RI: refractive index
3. Na: Sodium (unit measurement: weight percent in corresponding oxide, as are attributes 4-10)
4. Mg: Magnesium
5. Al: Aluminum
6. Si: Silicon
7. K: Potassium
8. Ca: Calcium
9. Ba: Barium
10. Fe: Iron
11. Type of glass: (class attribute)
 - 1 building_windows_float_processed
 - 2 building_windows_non_float_processed
 - 3 vehicle_windows_float_processed
 - 4 vehicle_windows_non_float_processed (none in this database)
 - 5 containers
 - 6 tableware
 - 7 headlamps

Primero debemos cargar los datos como data frame, y le damos nombres a las columnas.

```
glass <- read.table("glass.data", header = FALSE, sep = ",")
names(glass) <- c("Id number", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe", "Type of glass")
colnames(glass) <- make.names(colnames(glass))
str(glass)
```

```
## 'data.frame':    214 obs. of  11 variables:
## $ Id.number      : int  1 2 3 4 5 6 7 8 9 10 ...
## $ RI             : num  1.52 1.52 1.52 1.52 1.52 ...
## $ Na             : num  13.6 13.9 13.5 13.2 13.3 ...
## $ Mg             : num  4.49 3.6 3.55 3.69 3.62 3.61 3.6 3.61 3.58 3.6 ...
## $ Al             : num  1.1 1.36 1.54 1.29 1.24 1.62 1.14 1.05 1.37 1.36 ...
## $ Si             : num  71.8 72.7 73 72.6 73.1 ...
## $ K              : num  0.06 0.48 0.39 0.57 0.55 0.64 0.58 0.57 0.56 0.57 ...
## $ Ca             : num  8.75 7.83 7.78 8.22 8.07 8.07 8.17 8.24 8.3 8.4 ...
## $ Ba             : num  0 0 0 0 0 0 0 0 0 0 ...
```

```
## $ Fe      : num  0 0 0 0 0 0.26 0 0 0 0.11 ...
## $ Type.of.glass: int  1 1 1 1 1 1 1 1 1 1 ...
```

Vemos como el conjunto de datos está formado por nueve variables predictoras y la variable de clase Type.of.glass. Y todas las variables són numéricas.

Además, podemos observar que la primera columna (cuyo nombre es “Id.number”) es redundante (denota el identificador de cada instancia), por lo que borraremos esta columna. También le cambiaremos el nombre a la última columna (cuyo nombre es “Type.of.glass”) para que se entienda mejor que respresenta la variable clase.

```
glass <- subset(glass, select = -Id.number)
colnames(glass)[10] <- "Class"
head(glass, 10)
```

```
##      RI      Na  Mg   Al   Si    K   Ca Ba   Fe Class
## 1  1.52101 13.64 4.49 1.10 71.78 0.06 8.75 0 0.00     1
## 2  1.51761 13.89 3.60 1.36 72.73 0.48 7.83 0 0.00     1
## 3  1.51618 13.53 3.55 1.54 72.99 0.39 7.78 0 0.00     1
## 4  1.51766 13.21 3.69 1.29 72.61 0.57 8.22 0 0.00     1
## 5  1.51742 13.27 3.62 1.24 73.08 0.55 8.07 0 0.00     1
## 6  1.51596 12.79 3.61 1.62 72.97 0.64 8.07 0 0.26     1
## 7  1.51743 13.30 3.60 1.14 73.09 0.58 8.17 0 0.00     1
## 8  1.51756 13.15 3.61 1.05 73.24 0.57 8.24 0 0.00     1
## 9  1.51918 14.04 3.58 1.37 72.08 0.56 8.30 0 0.00     1
## 10 1.51755 13.00 3.60 1.36 72.99 0.57 8.40 0 0.11     1
```

Aunque a primera vista los datos parecen que estan ordenados por la variable clase Class, los ordenamos; y miramos que no existan valores codificados como NA.

```
glass <- glass[order(glass$Class), ]
any(is.na(glass))
```

```
## [1] FALSE
```

Como se ha comentado anteriormente, la columna Class representa la variable clase. Concretamente en este estudio contamos con seis variables explicativas.

A continuación, vamos a generar una representación binária. Primero, nos quedaremos solo con los valores de la variable clase {1, 2}, los renombraremos con {1 = “positive”, 2 = “negative”} y factorizaremos.

```
glass <- subset(glass , glass$Class < 3)
for (i in 1:146) {
  if (glass$Class[i] == 1){
    glass$Class[i] <- 'positive'
  }
  else {
    glass$Class[i] <- 'negative'
  }
}
glass[,10] <- as.factor(glass[,10])
str(glass)
```

```
## 'data.frame':   146 obs. of  10 variables:
## $ RI      : num  1.52 1.52 1.52 1.52 1.52 ...
## $ Na      : num  13.6 13.9 13.5 13.2 13.3 ...
## $ Mg      : num  4.49 3.6 3.55 3.69 3.62 3.61 3.6 3.61 3.58 3.6 ...
## $ Al      : num  1.1 1.36 1.54 1.29 1.24 1.62 1.14 1.05 1.37 1.36 ...
## $ Si      : num  71.8 72.7 73 72.6 73.1 ...
```

```
## $ K      : num  0.06 0.48 0.39 0.57 0.55 0.64 0.58 0.57 0.56 0.57 ...
## $ Ca      : num  8.75 7.83 7.78 8.22 8.07 8.07 8.17 8.24 8.3 8.4 ...
## $ Ba      : num  0 0 0 0 0 0 0 0 0 0 ...
## $ Fe      : num  0 0 0 0 0 0.26 0 0 0 0.11 ...
## $ Class: Factor w/ 2 levels "negative","positive": 2 2 2 2 2 2 2 2 2 ...
```

La decisión de quedarnos con los tipos de vidrio {1, 2} es porqué del tipo de vidrio {4} no tenemos valores en este conjunto de datos para clasificar, juntamente con el tipo de vidrio {3}; y los {5, 6, 7} no pueden generar una representación biaria con los otros tipos de vidrio.

División de los datos en muestra de entrenamiento y de test

Tomaremos una muestra del 70% del conjunto de datos como entrenamiento y el 30% del mismo como un conjunto de datos de prueba.

La validación cruzada usada cinco veces con muestreo estratificado se utilizó para dividir el conjunto de datos.

```
set.seed(123)
inTraining <- createDataPartition(glass$Class, p = .7, list = FALSE)
training <- glass[ inTraining,]
testing <- glass[-inTraining,]
fitControl <- trainControl(method = "cv",
                           number = 5,
                           classProbs = FALSE)

str(training)

## 'data.frame':    103 obs. of  10 variables:
## $ RI      : num  1.52 1.52 1.52 1.52 1.52 ...
## $ Na      : num  13.9 13.5 13.2 13.3 12.8 ...
## $ Mg      : num  3.6 3.55 3.69 3.62 3.61 3.6 3.61 3.6 3.66 3.43 ...
## $ Al      : num  1.36 1.54 1.29 1.24 1.62 1.14 1.05 1.36 1.27 1.4 ...
## $ Si      : num  72.7 73 72.6 73.1 73 ...
## $ K       : num  0.48 0.39 0.57 0.55 0.64 0.58 0.57 0.57 0.6 0.69 ...
## $ Ca      : num  7.83 7.78 8.22 8.07 8.07 8.17 8.24 8.4 8.56 8.05 ...
## $ Ba      : num  0 0 0 0 0 0 0 0 0 0 ...
## $ Fe      : num  0 0 0 0 0.26 0 0 0.11 0 0.24 ...
## $ Class: Factor w/ 2 levels "negative","positive": 2 2 2 2 2 2 2 2 2 ...

str(testing)

## 'data.frame':    43 obs. of  10 variables:
## $ RI      : num  1.52 1.52 1.52 1.52 1.52 ...
## $ Na      : num  13.6 14 12.7 13.9 13 ...
## $ Mg      : num  4.49 3.58 3.46 3.73 3.54 3.48 3.48 3.45 3.53 3.48 ...
## $ Al      : num  1.1 1.37 1.56 1.18 1.21 1.41 1.33 1.21 1.32 1.35 ...
## $ Si      : num  71.8 72.1 73.2 72.1 73 ...
## $ K       : num  0.06 0.56 0.67 0.06 0.65 0.59 0.56 0.56 0.51 0.64 ...
## $ Ca      : num  8.75 8.3 8.09 8.89 8.53 8.43 8.43 8.57 8.78 8.68 ...
## $ Ba      : num  0 0 0 0 0 0 0 0 0.11 0 ...
## $ Fe      : num  0 0 0.24 0 0 0 0 0 0 0 ...
## $ Class: Factor w/ 2 levels "negative","positive": 2 2 2 2 2 2 2 2 2 ...
```

Entrenamiento de los modelos y evaluación

Se utilizarán tres algoritmos de clasificación diferentes para entrenar, OneR, kNN y Multilayer Perceptrón.

1. OneR.

```

oner <- train(Class ~ ., data = training, method = "OneR", trControl = fitControl)
oner

```

```

## Single Rule Classification
##
## 103 samples
## 9 predictor
## 2 classes: 'negative', 'positive'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 83, 82, 82, 82, 83
## Resampling results:
##
## Accuracy Kappa
## 0.8452381 0.688327

```

```

onerPredict <- predict(oner, newdata = testing)
cm_oner <- confusionMatrix(onerPredict, testing$Class)
cm_oner

```

```

## Confusion Matrix and Statistics
##
##           Reference
## Prediction negative positive
## negative      18         7
## positive       4        14
##
##           Accuracy : 0.7442
##           95% CI : (0.5883, 0.8648)
##       No Information Rate : 0.5116
##       P-Value [Acc > NIR] : 0.001574
##
##           Kappa : 0.4864
##
## Mcnemar's Test P-Value : 0.546494
##
##           Sensitivity : 0.8182
##           Specificity : 0.6667
##       Pos Pred Value : 0.7200
##       Neg Pred Value : 0.7778
##           Prevalence : 0.5116
##       Detection Rate : 0.4186
##       Detection Prevalence : 0.5814
##       Balanced Accuracy : 0.7424
##
##       'Positive' Class : negative
##

```

2. kNN con k=1, k=3 y con peso por distancia (tres configuraciones en total).

```

grid_knn <- expand.grid(k = seq(1, 3))
knn <- train(Class ~ ., data = training, method = "knn", trControl = fitControl, tuneGrid = grid_knn)
knn

```

```

## k-Nearest Neighbors

```

```
##
## 103 samples
## 9 predictor
## 2 classes: 'negative', 'positive'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 83, 82, 82, 82, 83
## Resampling results across tuning parameters:
##
## k Accuracy Kappa
## 1 0.7852381 0.5701258
## 2 0.7657143 0.5325650
## 3 0.7361905 0.4718654
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 1.
```

```
knnPredict <- predict(knn, newdata = testing)
cm_knn <- confusionMatrix(knnPredict, testing$Class)
cm_knn
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction negative positive
## negative      16         2
## positive       6        19
##
##           Accuracy : 0.814
##           95% CI : (0.666, 0.9161)
##   No Information Rate : 0.5116
##   P-Value [Acc > NIR] : 3.932e-05
##
##           Kappa : 0.6293
##
## Mcnemar's Test P-Value : 0.2888
##
##           Sensitivity : 0.7273
##           Specificity : 0.9048
##           Pos Pred Value : 0.8889
##           Neg Pred Value : 0.7600
##           Prevalence : 0.5116
##           Detection Rate : 0.3721
##   Detection Prevalence : 0.4186
##           Balanced Accuracy : 0.8160
##
##           'Positive' Class : negative
##
```

3. Multilayer Perceptrón con una sola capa oculta y 3, 5, y 7 unidades ocultas en la misma (tres configuraciones en total).

```
grid_mlp = expand.grid(layer1 = 3, layer2 = 5, layer3 = 7)
mlp <- train(Class ~., data = training,
              method = "mlpML",
```

```

        trControl = fitControl,
        tuneGrid = grid_mlp)
mlp

## Multi-Layer Perceptron, with multiple layers
##
## 103 samples
##   9 predictor
##   2 classes: 'negative', 'positive'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 82, 83, 82, 82, 83
## Resampling results:
##
##   Accuracy   Kappa
##   0.5242857   0
##
## Tuning parameter 'layer1' was held constant at a value of 3
## Tuning
##   parameter 'layer2' was held constant at a value of 5
## Tuning parameter
##   'layer3' was held constant at a value of 7

mlpPredict <- predict(mlp, newdata = testing)
cm_mlp <- confusionMatrix(mlpPredict,testing$Class)
cm_mlp

## Confusion Matrix and Statistics
##
##               Reference
## Prediction negative positive
##   negative      22      21
##   positive       0       0
##
##               Accuracy : 0.5116
##               95% CI : (0.3546, 0.6669)
##   No Information Rate : 0.5116
##   P-Value [Acc > NIR] : 0.561
##
##               Kappa : 0
##
## Mcnemar's Test P-Value : 1.275e-05
##
##               Sensitivity : 1.0000
##               Specificity : 0.0000
##   Pos Pred Value : 0.5116
##   Neg Pred Value :   NaN
##   Prevalence : 0.5116
##   Detection Rate : 0.5116
##   Detection Prevalence : 1.0000
##   Balanced Accuracy : 0.5000
##
##   'Positive' Class : negative

```

##

Las métricas más adecuadas para la clasificación son Accuracy y F-measure (Sensitivity y Specificity)

Kappa 0?

Utilizando la información que nos dan las matrices de confusion contruimos una tabla con estas métricas.

TABLA

Análisis de la tabla. el accuray mayor contienen menos errores, es mejor modelo.

F measure.

Análisis ROC

Para el análisis ROC volveremos a predecir, pero ahora con el conjunto de datos de prueba y construiremos el objeto de predicción para cada modelo de clasificación utilizando el vector de estimación de probabilidades para la variable clase positiva.

1. OneR.

```
onerPredictProb <- predict(oner, newdata = testing, type = "prob")
onerPred <- prediction(onerPredictProb$positive, testing$Class)
onerPerf <- performance(onerPred, "tpr", "fpr")
```

2. kNN con k=1, k=3 y con peso por distancia (tres configuraciones en total).

```
knnPredictProb <- predict(knn, newdata = testing, type = "prob")
knnPred <- prediction(knnPredictProb$positive, testing$Class)
knnPerf <- performance(knnPred, "tpr", "fpr")
```

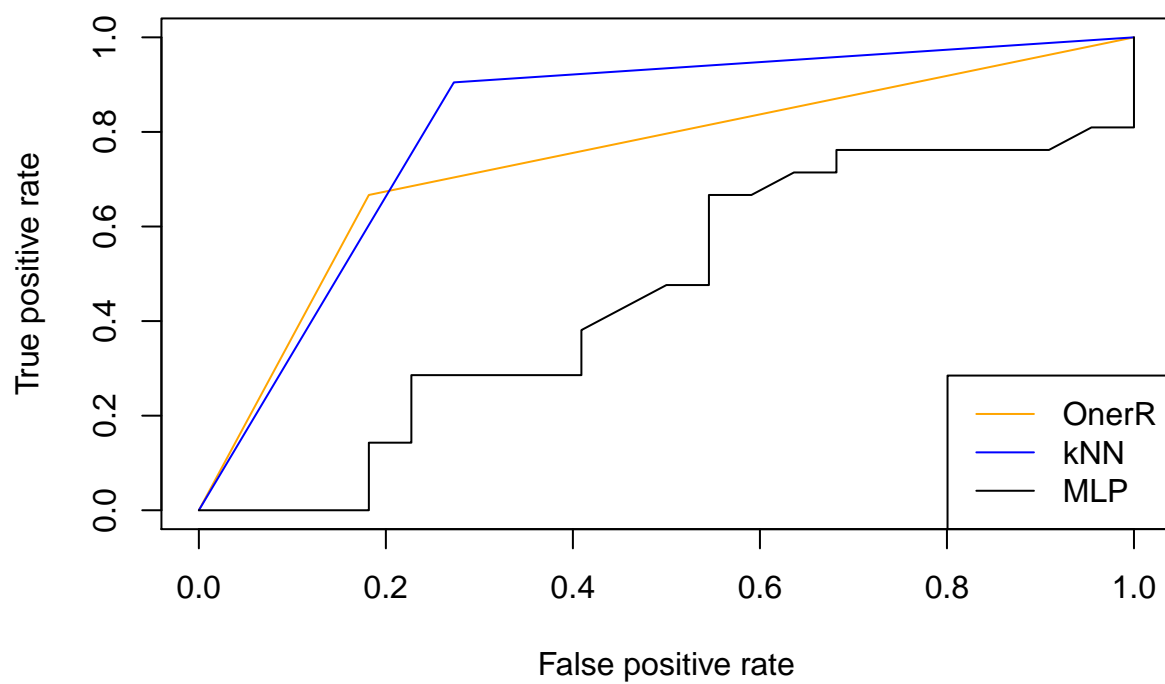
3. Multilayer Perceptrón con una sola capa oculta y 3, 5, y 7 unidades ocultas en la misma (tres configuraciones en total).

```
mlpPredictProb <- predict(mlp, newdata = testing, type = "prob")
mlpPred <- prediction(mlpPredictProb$positive, testing$Class)
mlpPerf <- performance(mlpPred, "tpr", "fpr")
```

Dibujamos las curvas ROC de los modelos.

```
plot(onerPerf, col = "orange", add = FALSE)
plot(knnPerf, col = "blue", add = TRUE)
plot(mlpPerf, col = "black", add = TRUE)
title(main = "CURVAS ROC")
legend("bottomright", legend = c("OnerR", "kNN", "MLP"), col = c("orange", "blue", "black"), lty = 1, l
```

CURVAS ROC



Si nos fijamos en el gráfico de las curvas ROC que refleja el análisis ROC podemos observar que el modelo de clasificación k-Nearest Neighbors para $k=1$ es el mejor para clasificar.