Aprendizaje Automático: Memoria Práctica 3

Óscar López Arcos. 75571640

1. Regresión

Problema a resolver

Este problema consiste en predecir el nivel de presión del sonido (en decibelios) de diferentes modelos de superficies aerodinámicas, testeadas por la NASA en varios túneles de sonido. Para aprender, disponemos de un conjunto de datos con variables que nos idican las circustancias bajo las que se obtuvo cada resultado, tales como el ángulo de ataque, la frecuencia, etc.

Nuestro objetivo será ser capaces de predecir los niveles de presión del sonido dado un conjunto de variables de los que se desconoce el resultado. Para ello, aplicaremos diferentes modelos de aprendizaje sobre los datos para obtener predicciones de calidad.

Lectura de datos y particionado

Creamos una partición *training* y una *test* con el 70% de los datos respectivamente. No será necesario un conjunto de validación, puesto que aplicaremos validación cruzada.

```
library("leaps")
library("caret")
```

```
## Warning: package 'caret' was built under R version 3.4.4
```

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

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

```
datos <- read.table("./datos/airfoil_self_noise.csv", quote="\"", sep = ",",
                    comment.char="", stringsAsFactors=FALSE)

set.seed(1)
names(datos)[ncol(datos)] = "LABELS"
train_index = createDataPartition(datos$LABELS, p = .70, list = FALSE)
training = datos[train_index,]
test = datos[-train_index,]
```

Preprocesamiento

Comenzamos eliminando aquellos atributos (columnas) cuya varianza sea 0 o próxima a este número. Estos datos no son relevantes ya que, al ser bastante similares para todos los casos, no aportan información o ésta es ínfima respecto a la magnitud del problema.

```
deleteNearZeroVar <- function(conjunto,columnasABorrar){
  conjunto = conjunto[-columnasABorrar]
  conjunto
}

columnasABorrar = nearZeroVar(training)
if(length(columnasABorrar) > 0){
```

```
  training = deleteNearZeroVar(training, columnasABorrar)
  test = deleteNearZeroVar(test, columnasABorrar)
}
```

Posteriormente, aplicando el método *preProcess* ajustamos los datos para poder trabajar con ellos.

```
ObjetoTrans =  preProcess(training, method = c("YeoJohnson", "center", "scale"),thres=0.8)
ObjetoTrans
```

```
## Created from 1055 samples and 6 variables
##
## Pre-processing:
##   - centered (6)
##   - ignored (0)
##   - scaled (6)
##   - Yeo-Johnson transformation (4)
##
## Lambda estimates for Yeo-Johnson transformation:
## 0.02, 0.34, -2.78, -0.15
```

```
training = predict(ObjetoTrans,training)
#training
test = predict(ObjetoTrans, test)
#test
```

Los tres métodos utilizados para preprocesar son:

-YeoJohnson: Extensión del método Box-Cox que funciona mejor cuando hay variables negativas o de valor 0 (como en nuestro caso). Su función consiste en corregir sesgos en la distribución de errores, corregir varianzas desiguales (para diferentes valores de la variable predictora) y principalmente para corregir la no linealidad en la relación (mejorar correlación entre las variables).
-center: Resta a cada variable predictora la media de éstas, haciendo que la media de los predictores sea igual a 0 y el valor de la intersección tenga sentido puesto que el 0 está en el rango.
-scale: Divide cada variable predictora por la desviación típica de éstas. Junto al método anterior, se encargan de normalizar el conjunto ($z = \frac{x-\mu}{\sigma}$)

En esta ocasión no es necesario aplicar **pca** dado que el número de variables ya es pequeño de por sí.
Una vez hemos preprocesado los datos, ya podemos comenzar con el proceso de experimentación para la obtención de un buen modelo.

Experimentación

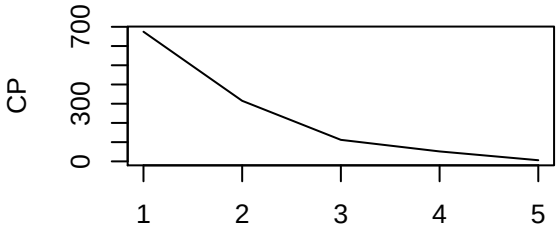
Vamos a comprobar cuál sería el número de variables óptimas a utilizar

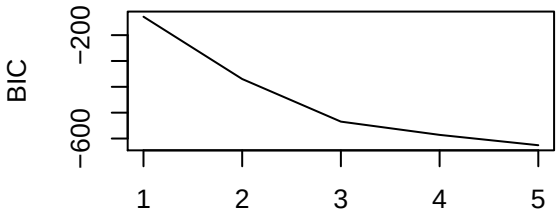
```
subset = regsubsets(training$LABELS ~ . , data = training,
                    nvmax = ncol(training), method = "forward")
summary = summary(subset)

par(mfrow=c(2,2))
plot(summary$cp, xlab = "Número de variables.", ylab = "CP", type = "l")
plot(summary$bic, xlab = "Número de variables.", ylab = "BIC", type = "l")
CP = which.min(summary$cp)
BIC = which.min(summary$bic)
cat("Mejor número de características - CP:", CP, "\n")
```

```
## Mejor número de características - CP: 5
```

```r
cat("Mejor número de características - BIC:", BIC, "\n")
```

```
## Mejor número de características - BIC: 5
```





En ambos casos coincide en que el número óptimo de características sería utilizarlas todas, por tanto, seguiré trabajando con los conjuntos *training* y *set* sin modificar.

Aplicación de distintos modelos

A la hora de aplicar los modelos, utilizaré la valicación cruzada que nos proporciona el método *train*, diviendo el conjunto *training* en 5 particiones.

```r
ctrol = trainControl(method="cv", number=5)
```

Para comenzar, voy a aplicar alguna transformación a los datos ya la clase de funciones lineales parece no ser la más adecuada.

La función *lm* usada para los modelos lineales, sirve para ajustar problemas de regresión. Comparando resultados con la métrica MSE decideremos la mejor transformación polinómica.

```r
# Datos sin modificar
lmMod1 <- train( LABELS ~ . , data=training, method ="lm", trControl = ctrol)
# Transformación cuadrada
lmMod2 <- train( LABELS ~ .^2, data=training, method ="lm", trControl = ctrol)
# Transformación cúbica
lmMod3 <- train( LABELS ~ .^3, data=training, method ="lm", trControl = ctrol)

# Error interno de los datos sin modificar
predTr1 <- predict(lmMod1, training)
etr1 = mean((predTr1 - training$LABELS)^2)
etr1
```

```
## [1] 0.5305585
```

```r
# Error interno de los datos al cuadrado
predTr2 <- predict(lmMod2, training)
etr2 = mean((predTr2 - training$LABELS)^2)
etr2
```

```
## [1] 0.381974
```

```r
# Error interno de los datos al cuadrado
predTr3 <- predict(lmMod3, training)
etr3 = mean((predTr3 - training$LABELS)^2)
etr3
```

```
## [1] 0.3369549
```

Parece que, conforme aumentamos el grado de la transformación, el error interno disminuye. Sin embargo, no queremos caer en el sobreajuste, así que dejaremos el Grado 3 como transformación final.

Necesidad de regularización:

Ahora, vamos a evaluar diferentes modelos y comprobar qué E_{in} es menor. Además del mejor modelo anterior, compararemos tanto con *ridge* como con *lasso*.

```
lmMod1 <- train( LABELS ~ .^3, data=training, method ="ridge", trControl = ctrol)
lmMod2 <- train( LABELS ~ .^3, data=training, method ="lasso", trControl = ctrol)

# Error interno sin regularización
predTr <- predict(lmMod3, training)
etr = mean((predTr - training$LABELS)^2)
etr
```

```
## [1] 0.3369549
```

```
# Error interno con regularización ridge
predTr <- predict(lmMod1, training)
etr = mean((predTr - training$LABELS)^2)
etr
```

```
## [1] 0.3369559
```

```
# Error interno con regularización lasso
predTr <- predict(lmMod2, training)
etr = mean((predTr - training$LABELS)^2)
etr
```

```
## [1] 0.3372494
```

En definitiva, la regularización no ha sido de gran ayuda en nuestro problema, es por eso que finalmente elegimos no aplicarla y obtener nuestro modelo final.

Obtención del E_{out} y conclusión

```
# Error fuera de la muestra aplicando nuestro modelo
predTst <- predict(lmMod3, test)
etst = mean((predTst-test$LABELS)^2)
etst
```

```
## [1] 0.3603172
```

Conclusión

Para la complejidad del problema a resolver, un error en torno al 30% es satisfactorio si tenemos en cuenta el reducido número de variables de las que hemos aprendido y que sólo hemos ajustado modelos lineales con clases de funciones polinómicas. Utilizando modelos más avanzados podrá reducirse el E_{in}, como veremos en el Proyecto Final de la asignatura.

2. Clasificación

Problema a resolver

Para este problema de reconocimiento de dígitos escritos a mano, disponemos de dos archivos *train* y *test*. Cada dígito es representado por 64 características, de las que deberemos aprender para lograr una correcta clasificación para datos de clase desconocida.

Es un problema multiclase, ya que cada dígito pertenece a una categoría del 0 al 9.

Construcción del algoritmo

Para resolver el problema de clasificación multiclase he decidido implementar el algoritmo $1vsALL$ que genera tantos clasificadores distintos como clases existentes. Como nuestro problema consta sólo de 10 clases, obtiene un resultado muy satisfactorio sin excederse en tiempo de computación.

Leemos los datos *train* y *test* de cada fichero respectivamente.

```
tr <- read.table("./datos/optdigits_tra.csv", quote="\"", sep = ",",
                        comment.char="", stringsAsFactors=FALSE)

tst <- read.table("./datos/optdigits_tes.csv", quote="\"", sep = ",",
                    comment.char="", stringsAsFactors=FALSE)
```

Defino la función *Mayor_clase* que, dado un vector de probabilidades, devolverá la clase (0-9) donde la probabilidad sea más alta. Esto será útil para, una vez obtenidos los 10 clasificadores, quedarnos con la clase que tenga mayor probabilidad.

```
Mayor_clase <- function(v){
  max = -1
  clase = -1
  for(i in 1:length(v)){
    if(v[i] > max){
      max = v[i]
      clase = i-1
    }
  }
  clase
}
```

A continuación, guardo las Clases a las que pertenecen los datos de *training* y *test* e inicializo las matrices donde posteriormente almacenaré las probabilidades de los distintos clasificadores.

```
etiquetasRealesTr = as.matrix(tr[,ncol(tr)])
etiquetasRealesTst = as.matrix(tst[,ncol(tst)])

ProbabilidadesTr <- matrix(nrow=10 , ncol=nrow(tr))
ProbabilidadesTst <- matrix(nrow=10 , ncol=nrow(tst))
regularizacion <- list() # Guarda los datos para estudiar Regularización
```

Una vez está todo listo, implemento el bucle que se encarga de generar los clasificadores, que explicaré a continuación.

Este bucle se repetirá 10 veces y en cada iteración crearemos un clasificador que distinga si el número pertenece a la clase "i" o es distinto de ésta. Cuando termine, obtendremos 10 clasificadores binarios que representen lo siguiente: "0/otro", "1/otro", "2/otro","3/otro", "4/otro", "5/otro", "6/otro", "7/otro","8/otro", "9/otro".

```
for (i in 0:9){

training = tr
test = tst

# Modificación de las etiquetas 0/1
etiquetasTr = rep(0, length(etiquetasRealesTr))
etiquetasTr[etiquetasRealesTr == i] = 1
training = training[,-ncol(training)]

etiquetasTst = rep(0, length(etiquetasRealesTst))
etiquetasTst[etiquetasRealesTst == i] = 1
test = test[,-ncol(test)]

# Preprocesamiento de datos
columnasABorrar = nearZeroVar(training)
training = deleteNearZeroVar(training, columnasABorrar)
test = deleteNearZeroVar(test, columnasABorrar)
ObjetoTrans = preProcess(training, method = c("YeoJohnson", "center",
                                              "scale", "pca"),thres=0.8)
training_preprocess = predict(ObjetoTrans,training)
test_preprocess = predict(ObjetoTrans, test)

# Almaceno las etiquetas 0/1 junto a los datos
training_preprocess = as.data.frame(cbind(etiquetasTr,
                                          training_preprocess))
test_preprocess = as.data.frame(cbind(etiquetasTst,
                                      test_preprocess))

# Obtener las variables más representativos
subsetTraining = regsubsets(training_preprocess$etiquetasTr ~ . ,
                            data = training_preprocess,
                             nvmax = ncol(training_preprocess),
                            method = "forward")

summaryTraining = summary(subsetTraining)
CP = which.min(summaryTraining$cp)

# Obtengo datos
datosTrainModeloCP = training_preprocess[, c(1, as.vector
(which(summaryTraining$outmat[CP,] == "*")) + 1)]

datosTestModeloCP = as.data.frame(test_preprocess[,as.vector
(which(summaryTraining$outmat[CP,] == "*")) + 1])

regularizacion[[i+1]] = datosTestModeloCP
# Aplico el modelo
glm1 = glm(datosTrainModeloCP$etiquetasTr ~ ., data = datosTrainModeloCP)

# Selección y ajuste final
datosTrainModeloCP = datosTrainModeloCP[,-datosTrainModeloCP$etiquetasTr]

probTrainModelo = predict(glm1, datosTrainModeloCP, type="response")
```

```
probTestModelo = predict(glm1, datosTestModeloCP, type="response")

ProbabilidadesTr[i+1,] <- probTrainModelo
ProbabilidadesTst[i+1,] <- probTestModelo
}
```

Modificación de las etiquetas 0/1

En cada iteración modifico las etiquetas (clases) de los números a 0 ó 1. Si el número pertenece a la clase "i" se le asignará un 1, 0 en caso contrario.

Preprocesamiento de datos

De forma parecida al apartado anterior, preprocesaremos los datos para poder trabajar y aprender de ellos. En esta ocasión sí utilizaré (además de los usados en regresión) en método "pca", puesto que el número de variables es bastante elevado.
-pca: Preprocesado con Análisis Principal de Componentes. Reduce el número de predictores para explicar los datos, reduce la media de ruido.

De esta forma, conseguimos reducir el total de variables de 64 a unas 15 en cada iteración.

Obtención de las variables más representativas

De igual modo, utilizo de la función *regsubsets* para calcular el CP y utilizar el número de variables óptimo a la hora de clasificar. Como en este caso el óptimo suele ser mejor al total, modifico ambos conjuntos utilizando ese número de variables.

Aplicación del modelo

Finalmente, aplico regresión logística (*glm*) al conjunto de datos resultante. Tras predecir las probabilidades del clasificador para *train* y *test*, las almaceno en las variables *ProbabilidadesTr* y *ProbabilidadesTst* respectivamente.

Necesidad de regularización

Para este problema no ha sido necesario aplicar regularización porque, como se puede ver en el siguiente fragmento, las desviaciones en el caso de usarla (mejor lambda estimado) o no (lambda 0) son muy parecidas.

```
library("glmnet")
```

```
## Warning: package 'glmnet' was built under R version 3.4.4
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-16
```

```
diferencia_dev = NULL
for(i in 1:10){
  datosTrainModeloCP = regularizacion[[i]]
  mejorLambda = cv.glmnet(x = as.matrix(datosTrainModeloCP[,-1]),
                y = as.matrix(datosTrainModeloCP[,1]), alpha = 0)$lambda.min
```

```
  # Sin regularización
  sin_reg = glmnet(x = as.matrix(datosTrainModeloCP[,-1]),
         y = as.matrix(datosTrainModeloCP[,1]), alpha = 0,
         lambda = 0, standardize = FALSE)[6]
  # Con regularización
  con_reg = glmnet(x = as.matrix(datosTrainModeloCP[,-1]),
         y = as.matrix(datosTrainModeloCP[,1]), alpha = 0,
         lambda = mejorLambda, standardize = FALSE)[6]

  diferencia_dev[i] = abs(as.numeric(con_reg)-as.numeric(sin_reg))
}
diferencia_dev
```

```
##  [1] 0.0006410798 0.0005228407 0.0003483235 0.0006448198 0.0008619277
##  [6] 0.0008082128 0.0007465701 0.0008716873 0.0014848189 0.0013023697
```

Obtención de Errores y conclusión

Para predecir las etiquetas utilizamos la función $Mayor_clase$ explicada anteriormente, para las probabilidades de cada una de las muestras.

```
predTrainModelo = rep(0, length(probTrainModelo))

for(i in 1:ncol(ProbabilidadesTr)){
  predTrainModelo[i] = Mayor_clase( ProbabilidadesTr[,i] )
}
```

Aplicamos la fórmula del error en clasificación, y vemos que el E_{in} es bastante bajo.

```
errorTrainModelo = mean(predTrainModelo != etiquetasRealesTr)
errorTrainModelo
```

```
## [1] 0.09469003
```

```
confusionMatrix(data=factor(predTrainModelo), reference=factor(etiquetasRealesTr))$table
```

```
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 375   0   2   0   3   2   2   0   2  11
##          1   0 317   1   0   5   0   3   2  18   7
##          2   0  14 358   9   0   3   0   2   5   3
##          3   0   5   2 357   0   2   0   0   4  22
##          4   1   3   0   0 348   0   1   2   4  11
##          5   0   5   0   9   0 335   0   0   3   7
##          6   0   8   0   0   7   5 371   0  16   0
##          7   0   3   3   3   9   0   0 381   2  14
##          8   0  20  13   5   4   1   0   0 323  11
##          9   0  14   1   6  11  28   0   0   3 296
```

```
confusionMatrix(data=factor(predTrainModelo), reference=factor(etiquetasRealesTr))$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9053100      0.8947898      0.8955857      0.9144093      0.1017526
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

Siguiendo el mismo procedimiento calculo el E_{out} y observo que, efectivamente, la predicción es más que satisfactoria.

```
predTestModelo = rep(0, length(probTestModelo))
for(i in 1:ncol(ProbabilidadesTst)){
  predTestModelo[i] = Mayor_clase( ProbabilidadesTst[,i] )
}
errorTestModelo = mean(predTestModelo != etiquetasRealesTst)
errorTestModelo
```

```
## [1] 0.1229827
```

```
confusionMatrix(data=factor(predTestModelo), reference=factor(etiquetasRealesTst))$table
```

```
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 177   0   1   2   3   1   0   0   1   2
##          1   0 126   3   0   2   0   3   0  21   2
##          2   0  14 162   6   0   0   0   0   1   0
##          3   0   2   5 155   0   0   0   0   1   9
##          4   1   3   0   0 170   0   1   7   5   7
##          5   0   3   0   5   0 175   3   5   4   4
##          6   0  10   0   4   0   1 174   0   9   2
##          7   0   1   2   4   4   0   0 165   1   3
##          8   0  12   4   4   2   0   0   1 127   6
##          9   0  11   0   3   0   5   0   1   4 145
```

```
confusionMatrix(data=factor(predTestModelo), reference=factor(etiquetasRealesTst))$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.8770173      0.8633468      0.8609366      0.8918552      0.1018364
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

Conclusión

El método $1vsALL$ ofrece un resultado de clasificación muy bueno, como nos indican los errores obtenidos y el valor de Accuracy. Si observamos la matriz de confusión, puede apreciarse que los errores se concentran en dígitos que ciertamente son bastante parecidos entre sí (por ejemplo, 3 y 9). Para este problema sí ha sido suficiente con la clase de funciones lineales.