CLASIFICACIÓN LINEAL Y K-NN

EVALUACIÓN DE MODELOS DE PREDICCIÓN

PRÁCTICA

Datos y funciones

Del Campus Virtual UB

`wine.csv`

Este dataset también aparece en el UCI Machine Learning

Del paquete `datasets` (que se carga automáticamente), los datos `iris` (clasificación de las flores Iris, procedente del artículo de R.A. Fisher (1936).

Del paquete `ElemStatLearn`, los datos `SAheart` (clasificación de un conjunto de pacientes de enfermedad cardíaca), los datos `spam` (clasificación de correos electrónicos entre spam y mensajes), los datos `vowel.train` y `vowel.test`, los datos `zip.train` y zip.test.

UCI Machine Learning Repository

`http://archive.ics.uci.edu/ml/`

contiene numerosos conjuntos de datos documentados para pruebas de métodos de Aprendizaje Automàtico. Los datos wine sirven para una clasificación de vinos en tres clases a partir de medidas de carácter organoléptico.

```
wine.url<-"http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data"
wine<-read.csv(wine.url,header=FALSE)
```

```
colnames(wine)<-c("Type","Alcohol","Malic","Ash",
"Alcalinity","Magnesium","Phenols","Flavonoids",
"Nonflavonoids","Proanthocyanins","Color","Hue",
"Dilution","Proline")
```

```
wine$Type <- as.factor(wine$Type)
```

Esta última línea tiene el propósito de informar al intérprete R que esta variable es de carácter cualitativo, para una clasificación.

el conjunto de datos 'Adult'

http://archive.ics.uci.edu/ml/datasets/Adult

y el conjunto de datos 'concrete' (Concrete Compressive Strength)

http://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength

Los datos 'Adult', ya preparados en R, vienen en:

```
library(rebmix)
```

```
data(adult)
```

Como suele ocurrir con datos reales, hay observaciones incompletas. Para esta práctica descartaremos estas observaciones:

```
adult <- adult[complete.cases(adult), ]
```

Este conjunto de datos consiste en datos de diversos parámetros socioeconómicos de individuos, tomados del censo de USA en 1994. Se trata de un problema de clasificación en dos clases, individuos que ganan más de $50K y los que no.

Los datos 'concrete' también vienen en R:

```
library(regsel)
```

```
data(concrete)
```

Para instalar el package `regsel` se requieren, como dependencias, los `glmnet, Matrix, foreach`.

El problema aquí es de regresión de la respuesta CompressiveStrength en función de las demás variables.

Matriz de confusión

Para una clasificación en g clases, la matriz de confusión es una matriz con g filas y g columnas, que muestra la calidad de un algoritmo de clasificación.

Partiendo de un conjunto de prueba para el que se conoce la clasificación correcta de cada elemento, se aplica el algoritmo y, en cada fila i-ésima de la matriz se pone en cada columna j-ésima el número de elementos de la clase i que han sido clasificados en el grupo j por el algoritmo.

Una clasificación perfecta daría lugar a una matriz de confusión con elementos no nulos sólo en la diagonal principal. La proporción de elementos fuera de esta diagonal es una estimación de la probabilidad de clasificación errónea.

Clasificador k-NN

Aparte de la implementación "hecha a mano" que estudiamos hace unos días, tenéis una implementación del clasificador por k-NN en el paquete `class`. Con los datos `mixture.example` del paquete `ElemStatLearn` (y el código que aparece en el help del dataset) reconstruid la clasificación de estos datos por k-NN, con varios posibles valores de k.

Clasificación lineal por mínimos cuadrados

La función `lm()` sirve para ajustar modelos lineales por mínimos cuadrados en R. Mirad el help de la función para ver detalles de la sintaxis.

`Preparamos los datos wine:`

```
y<-wine$Type
Y<-cbind((y=="1"),(y=="2"),(y=="3"))
colnames(Y)<-c("Y1","Y2","Y3")
wine2<-data.frame(wine[,-1],Y)
```

Subdividimos el conjunto en dos partes, `train` y `test`:

```
n<-nrow(wine2)
ntrain<-ceiling(0.6*n)
ntest<-n-ntrain
Itrain<-sample(1:n,ntrain,replace=FALSE)
wine2.train<-wine2[Itrain,]
wine2.test<-wine2[-Itrain,]
```

Ajustamos el modelo lineal:

```
wine2.lm1<-lm(cbind(Y1,Y2,Y3)~.,data=wine2.train)

summary(wine2.lm1)
```

Preparamos los datos para predicción:

```
wine2.test.to.predict<-wine2.test[,-(14:16)]
```

Hacemos la predicción:

```
Yhat<-predict(wine2.lm1,newdata<-wine2.test.to.predict)
```

Comprobamos la estructura de Yhat, que es efectivamente una matriz de 3 columnas y que cada fila suma 1.

```
str(Yhat)
apply(Yhat,1,sum)
```

En cambio, los elementos no están entre 0 y 1, lo que causa ambigüedad en la interpretación de las clasificaciones. Esta es una razón por la que es inusual clasificar por este método; seguimos con él a modo de ejercicio. Optaremos por asignar cada individuo a la clase (columna) con el máximo valor en la fila correspondiente.

Hallamos la matriz `Yhat.ind` que tiene en cada fila el indicador del máximo de la correspondiente fila de Yhat:

```
Yhat.max<-apply(Yhat,1,max)
Yhat.ind<-1*cbind(Yhat[,1]==Yhat.max,Yhat[,2]==Yhat.max,
```

```
    Yhat[,3]==Yhat.max)
```

Cálculo de la matriz de confusión mediante un producto de matrices (atención, son matrices de ceros y unos!):

```
Y.test<-Y[-Itrain,]

C<-t(Y.test)%*%Yhat.ind
```

Cálculo de la matriz de confusión por otro procedimiento:

```
y.test<-y[-Itrain]
yhat<-as.factor(apply(Yhat,1,which.max))
C<-table("True"=y.test,"Predicted"=yhat)
```

Realizad la clasificación de los datos `iris` y de los datos `SAheart` por el método de mínimos cuadrados.

Regresión logística

Como se ha visto en la clase de teoría, la regresión logística es un modelo estadístico, concretamente un modelo lineal generalizado *(generalized linear model – GLM)* y se ajusta mediante la función `glm()` del paquete `stats` de R. La sintaxis es muy parecida a la de la función `lm()` que ya conocemos.

Por ejemplo, para los datos `SAheart`:

```
require(ElemStatLearn)
data(SAheart)
n<-nrow(SAheart)
ntrain<-ceiling(0.60*n)
Itrain<-sample(1:n,ntrain,replace=FALSE)
n<-nrow(SAheart)
ntrain<-ceiling(0.60*n)
Itrain<-sample(1:n,ntrain,replace=FALSE)

SAheart.train<-SAheart[Itrain,]
```

```
SAheart.test<-SAheart[-Itrain,]


SAheart.logit1<-
glm(chd~.,data=SAheart.train,family=binomial)

summary(SAheart.logit1)
```

Podéis ver que aparece un listado muy parecido al del modelo lineal, incluyendo p-valores que miden la significación de los coeficientes. Esto es posible por tratarse de un modelo estadístico, en vez de un simple procedimiento de clasificación. Otra consecuencia de este hecho es la posibilidad de realizar la selección de predictores por el mismo procedimiento que en el `lm()`. Por ejemplo:

```
step( SAheart.logit1)
```

lleva a un modelo:

```
chd ~ tobacco + ldl + famhist + typea + age
```

en el que se han eliminado `sbp, adiposity, obesity, alcohol` de la lista de predictores por no ser (suficientemente) significativos. Podemos ajustar este modelo haciendo:

```
SAheart.logit2<-glm(chd ~ tobacco + ldl + famhist + typea
+ age,data=SAheart.train,family=binomial)
```

Para realizar predicciones, la sintaxis para obtener valores de probabilidad de que la respuesta binaria 0/1 tome el valor 1 (en este caso `chd=1`) es, por ejemplo:

```
Saheart.pred<-predict(SAheart.logit1,

    newdata=SAheart.test,type="response")
```

Si se ha de hacer una asignación definida 0 o 1, se ha de decidir un punto de corte, por ejemplo `L=0.5` (aunque no es obligatorio este valor, puede

depender del caso, o de las probabilidades a priori) y clasificar como 0 o 1 dependiendo de si la probabilidad es menor o mayor que este umbral:

```
SAheart.pred.crisp<-1*(SAheart.pred>=0.5)
```

Por ejemplo, para calcular la matriz de confusión:

```
C<-table("True"=SAheart.test$chd,
     "Predicted"=SAheart.pred.crisp)
```

Detalles sobre los cálculos en la regresión logística

En las láminas `Detalles.Reg.Logistica.slides.esp.pdf` tenéis una explicación del modelo de regresión logística y del procedimiento numérico de cálculo de las estimaciones de máxima verosimilitud. Se trata de una optimización por el método de Newton-Raphson, que da lugar a un cálculo iterativo por aproximaciones sucesivas, cada una de los cuales equivale a un cálculo de mínimos cuadrados ponderado (con un peso para cada observación). En cada etapa se actualizan estos pesos. El procedimiento se llama IWLS o IRLS, por *Iteratively (Re)Weighted Least Squares*. En el script `IWLS.r` podéis ver un ejemplo simple de implementación de este algoritmo.

En los datos `separation.txt` tenéis un caso de no existencia de estimador de máxima verosimilitud. Probad estos datos con la función `IWLS` de este script y con la función `glm` de R.

Discriminador lineal de Fisher

El discriminador lineal de Fisher está implementado en la función `lda` del paquete `MASS`.

Por ejemplo, para clasificar los datos `iris` de Fisher, haciendo una evaluación de la matriz de confusión con una parte de la muestra:

```
require(MASS)
```

```
iris.train<-iris[Itrain,]
iris.test<-iris[-Itrain,]
iris.lda<-lda(Species~.,data=iris.train)
iris.pred<-predict(iris.lda,newdata=iris.test)

    C<-table("True"=iris.test$Species,
    "Predicted"=iris.pred$class)
```

Con los datos SAheart: Observad que la respuesta, la variable chd, está codificada como numérica, mientras que uno de los factores de reisgo, la variable famhist, está codificada como un factor con dos niveles. Puede convenir para algunos métodos hacer el cambio de factor a numérica de la variable predictora, o bien de numérica a factor de la respuesta. Por ejemplo, la función `glm()`, que usamos para ajustar la regresión logística, puede tratar adecuadamente los factores que aparecen como variables predictoras, pero para otros métodos es mejor comprobarlo.

```
require(ElemStatLearn)
data(SAheart)
n<-nrow(SAheart)
ntrain<-ceiling(0.60*n)
Itrain<-sample(1:n,ntrain,replace=FALSE)
n<-nrow(SAheart)
ntrain<-ceiling(0.60*n)
Itrain<-sample(1:n,ntrain,replace=FALSE)

SAheart.train<-SAheart[Itrain,]
SAheart.test<-SAheart[-Itrain,]

SAheart.lda1<-lda(chd~.,data=SAheart.train)
SAheart.pred<-predict(SAheart.lda1,newdata=SAheart.test)

C<-table("True"=SAheart.test$chd,
        "Predicted"=SAheart.pred$class)
```

Ajustar y validar el modelo de predicción

El objetivo de esta práctica es ajustar un buen modelo de predicción en los casos de clasificación y regresión, evitando el sobre-entrenamiento mediante los métodos de evaluación de la calidad de la capacidad predictiva del modelo estudiados en clase:

1. *Hold-out* (separar un conjunto de test sobre el que se aplica el modelo de predicción ajustado sobre el conjunto 'train' de entrenamiento),
2. *k-fold cross-validation*,
3. *Leave-one-out*,
4. *Bootstrap*.

Los tres primeros métodos ya son conocidos. En la sección siguiente dedicamos algún espacio a estudiar el método (4).

Evaluación bootstrap de un modelo de predicción

El bootstrap sobre un conjunto x de n observaciones parte de realizar un cierto número de remuestras, es decir, muestras de tamaño n de x, realizadas con reposición, es decir, de forma que un mismo elemento puede aparecer más de una vez (y también puede no aparecer). Ejemplo:

```
x<-1:12
n<-length(x)
x1<-sample(x,n,replace = TRUE)
x1
 5 11 11  9  3  5  3  7  7  4 12 11
```

Como hay observaciones repetidas, otras observaciones no aparecen en la remuestra. El conjunto de observaciones que no aparece se suele llamar OOB *(Out-of-bag)*. Podemos detectarlas haciendo:

```
oob<-x[is.na(match(x,x1))]
oob
[1]  1  2  6  8 10
```

Una forma de aplicar el procedimiento bootstrap para evaluar un modelo de predicción consiste en usar la remuestra como conjunto train de

entrenamiento y el conjunto OOB como conjunto de test, y hacer esta operación repetidamente con un número B de remuestras. Alternativamente, se puede tomar como conjunto de test el total de la muestra original. Si se hace así, algunas observaciones aparecen en ambos subconjuntos, el de entrenamiento y el de test. En cualquier caso, como estimación bootstrap del error de predicción, `E.boot`, se toma la media aritmética de los B errores.

La regla 0.632

Sabemos que el estimador `E.subst` del error de predicción obtenido usando como conjunto de test el mismo conjunto usado para el entrenamiento tiene un sesgo optimista, debido al hecho de que los mismos datos ya son conocidos en cierto modo por la máquina de predicción. En cambio, el estimador bootstrap descrito más arriba, `E.boot`, tiene un sesgo pesimista.

En sus artículos, el de 1983: *"Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation"* y el de 1997: *"Improvements on Cross-Validation: The .632+ Bootstrap Method",* Bradley Efron, el iniciador del bootstrap, propuso la idea, que se ha impuesto generalmente, de promediar ambos estimadores, a fin de compensar los sesgos. Concretó la idea en la regla 0.632, consistente en adoptar, como estimador del error de predicción el valor `E.632` definido por:

```
E.632 = 0.632 * E.boot + 0.368 * E.subst
```

El motivo del coeficiente 0.632 viene del cálculo aproximado de la probabilidad de que una observación dada aparezca en una remuestra, que es:

$$1-(1-1/n)^n \approx 1-1/e.$$

Tarea para entregar

Se trata de aplicar los cuatro métodos de ajustar y validar el modelo, razonando detalladamente en el informe la justificación de elegir un modelo:

1. Para la clasificación de los datos `spam` y `wine`, con todos los métodos estudiados que sean aplicables en cada caso, comparando la calidad de la clasificación, calculando la matriz de confusión y una estimación de la probabilidad de clasificación errónea. Los datos `spam` tienen dos categorías

(un mensaje de correo puede ser o no spam), con estos se puede aplicar la regresión logística, que no es directamente aplicable a los datos `wine`, que tienen tres categorías.

2. Para la regresión, con los datos `concrete`.

Indicación: Hay muchos packages de `R` con funciones que implementan los diversos métodos de validación de modelos de predicción, por ejemplo `caret`, `CVST`, `cvTools`, `dprep`, `sortinghat`. Uno de ellos (`CVST`) está dedicado a una técnica de validación más sofisticada que las cuatro vistas en esta práctica, y los otros son genéricos, con aplicaciones a algunos modelos de predicción que todavía no hemos visto, o que no hemos de ver en este curso. Para vuestra tarea podéis usar alguna de estas funciones, o bien escribir un código vuestro, dedicado a implementar, por ejemplo, la validación *k-fold* o *leave-one-out.* Yo me inclino más bien hacia esta segunda opción: no es gran dificultad programar esto y, siempre, usar código ajeno representa entender la forma de expresar los datos y extraer los resultados, pero dejo a vuestro criterio la elección. La calificación de la tarea no depende de ella.