Trabajo 3

Aprendizaje Automático

Grado en Ingeniería Informática

Granada, 31 de Mayo de 2015.

Datos del estudiante

Fernández Bosch, Pedro
76422233-H

Ejercicio.-1 (3 puntos) (comentar los resultados de todos los apartados)
Usar el conjunto de datos OJ que es parte del paquete ISLR

1. Crear un conjunto de entrenamiento conteniendo una muestra aleatoria de 800 observaciones, y un conjunto de test conteniendo el resto de las observaciones. Ajustar un clasificador SVM (con núcleo lineal) a los datos de entrenamiento usando cost=0.01, con "Purchase" como la respuesta y las otras variables como predictores.

En este ejercicio se ha utilizado la base de datos OJ, que es parte del paquete ISLR.

```
> library(ISLR)
> library(e1071)
> library(ROCR)
> data(OJ)
> attach(OJ)
> ?OJ
```

Los datos de OJ contienen 1070 registros de compras, donde el cliente ha adquirido Citrus Hill o Minute Maid Orange Juice. La base de datos registra una serie de características sobre el cliente y el producto, como el ID de la tienda, precio cobrado por CH, precio cobrado por MM, descuento ofrecido por CH, descuento ofrecido por MM, etc.

En primer lugar se han ordenado de forma aleatoria las filas de la base de datos OJ. Para ello se ha utilizado la función sample() y se ha definido el subconjunto de entrenamiento (train) formado por 800 observaciones y también se ha definido el subconjunto de test formado por las observaciones restantes.

```
> train = sample(nrow(OJ), 800)
> OJ.train = OJ[train, ]
> OJ.test = OJ[-train, ]
```

La función svm() ha sido utilizada para entrenar el vector clasificador de soporte cuando se utiliza el argumento kernel = "lineal". El argumento "cost" permite especificar el costo, que para este problema se le ha definido valor 0.01. Cuando el costo es pequeño, los márgenes serán anchos y cuando el costo es grande, los márgenes serán estrechos.

```
> set.seed(1)
> svmfit = svm(Purchase ~ ., data = OJ.train, kernel = "linear", cost = 0.0
1)
```

2. Usar la función summary() para producir un resumen estadístico, y describir los resultados obtenidos. ¿Cuáles son las tasas de error de "training" y "test"?

Se ha obtenido la información básica sobre el clasificador SVM mediante el comando summary():

```
> summary(svmfit)
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01
      gamma:  0.05555556
Number of Support Vectors:  437
 ( 218 219 )
Number of Classes:  2
Levels:
 CH MM
```

El clasificador de SVM ha creado 438 vectores de soporte sobre los 800 puntos de train. Entre ellos 218 pertenecen al nivel CH y los 220 restantes pertenecen al nivel de MM.

A continuación se muestra la matriz de confusión para las muestras de entrenamiento:

```
> svmfit = svm(Purchase ~ ., data = OJ.train, kernel = "linear", cost = 0.0
1)
> train.pred = predict(svmfit, OJ.train)
> table(OJ.train$Purchase, train.pred)
    train.pred
      CH  MM
  CH 419  61
  MM  70 250
> mean(train.pred!= OJ.train$Purchase)
[1] 0.16375
```

Se ha obtenido una tasa de error para la muestra de entrenamiento del 16,37%.

A continuación se muestra la matriz de confusión para las muestras de test:

```
> test.pred = predict(svmfit, OJ.test)
> table(OJ.test$Purchase, test.pred)
    test.pred
      CH  MM
  CH 144  29
  MM  22  75
> mean(test.pred!= OJ.test$Purchase)
[1] 0.1888889
```

Se ha obtenido una tasa de error para la muestra de test del 18.88%.

3. Usar las función tune() para seleccionar un coste óptimo. Considerar los valores de "cost" del vector: [0.001, 0.01, 0.1, 1, 10]. Dibujar las curvas ROC para los diferentes valores del "cost".

La función tune(), realiza de forma predeterminada diez iteraciones de validación cruzada sobre un conjunto de datos. La función compara SVMs con un núcleo lineal, utilizando un rango de valores según el parámetro cost.

```
> set.seed (3)
> tune.out=tune(svm ,Purchase~.,data=OJ.train ,kernel = "linear", ranges =l
ist(cost=c(0.001 , 0.01, 0.1, 1,10) ))
> summary(tune.out)

Parameter tuning of 'svm':
- sampling method: 10-fold cross validation
- best parameters:
 cost
  0.1
- best performance: 0.16
- Detailed performance results:
   cost   error dispersion
1 1e-03 0.31375 0.08195739
2 1e-02 0.17125 0.04450733
3 1e-01 0.16000 0.04322101
4 1e+00 0.16625 0.04489571
5 1e+01 0.16625 0.04332131
```

A partir de los resultados obtenidos, se ha deducido que la tasa de error de validación cruzada más bajo es para cost = 0,1.

Ahora, resulta posible dibujar las curvas ROC para los diferentes valores del "cost". El paquete ROCR ha sido utilizado para producir curvas ROC, pero primero fue necesario utilizar una breve función para trazar una curva ROC dado un vector que contiene una puntuación numérica para cada observación (pred), y un vector que contiene la etiqueta de clase para cada observación (truth).

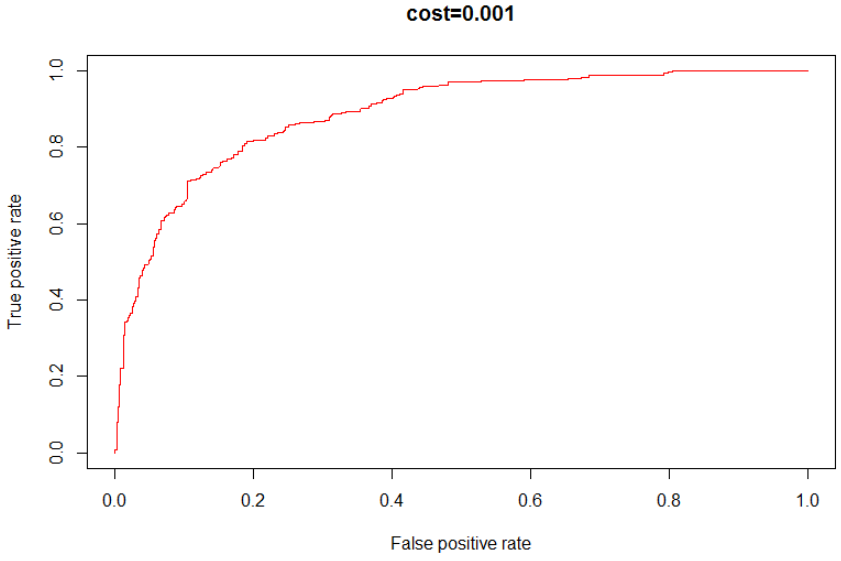
```
> rocplot = function(pred, truth, ...){
+ predob = prediction(pred,truth)
+ perf = performance(predob,"tpr","fpr")
+ plot(perf,...)
+ }
```

Para cost = 0,001

```
> svmfit=svm(Purchase~., data=OJ.train, kernel ="linear", cost=0.001, decis
ion.values =T)
> fitted =attributes (predict (svmfit ,OJ.train, decision.values =T))$decis
ion.values
```

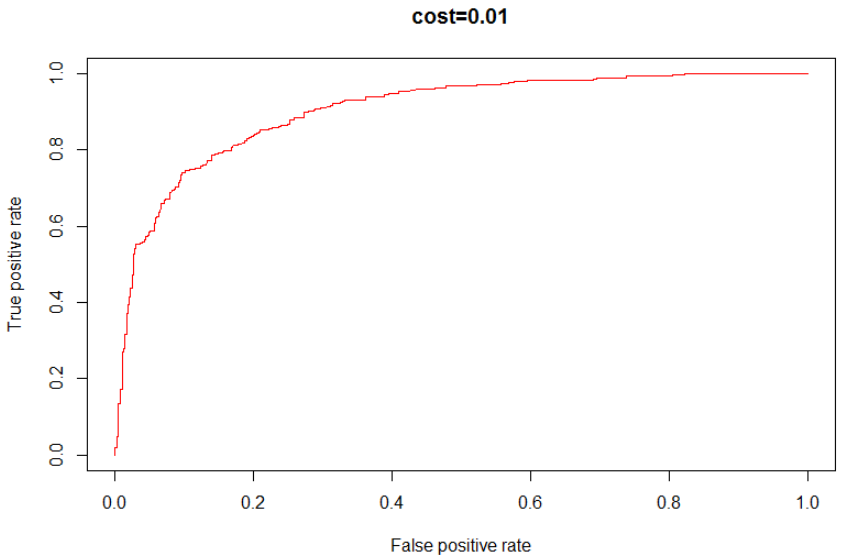
Ahora se puede generar el diagrama ROC.

```
> rocplot(fitted ,OJ.train[,"Purchase"], main="cost=0.001",col ="red ")
```



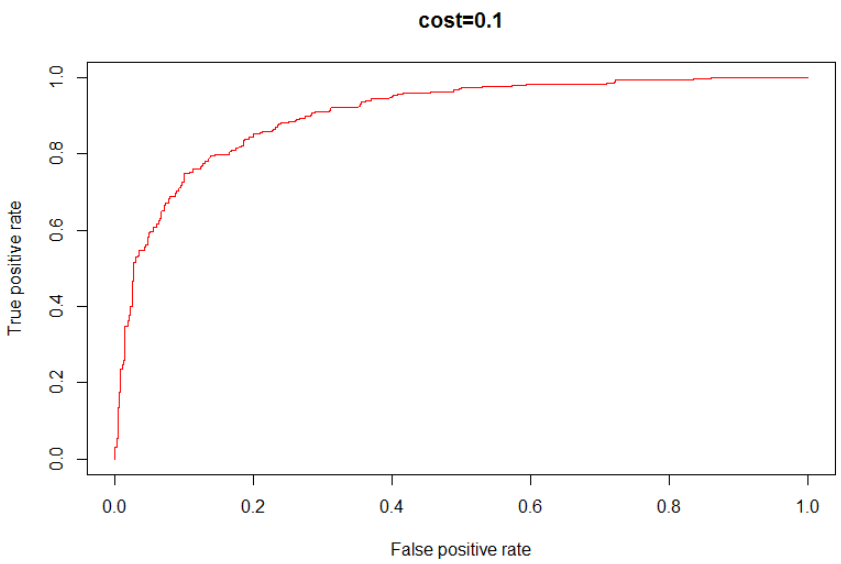
Para cost = 0,01

```
> svmfit=svm(Purchase~., data=OJ.train, kernel ="linear", cost=0.01, decisi
on.values =T)
> fitted =attributes (predict (svmfit ,OJ.train, decision.values =T))$decis
ion.values
> rocplot(fitted ,OJ.train[,"Purchase"], main="cost=0.01",col ="red ")
```



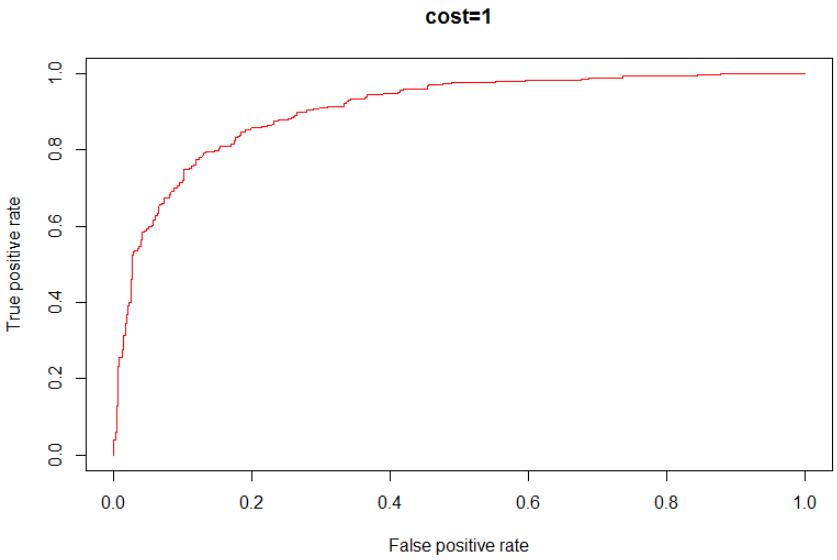
Para cost = 0,1

```
> svmfit=svm(Purchase~., data=OJ.train, kernel ="linear", cost=0.1, decisio
n.values =T)
> fitted =attributes (predict (svmfit ,OJ.train, decision.values =T))$decis
ion.values
> rocplot(fitted ,OJ.train[,"Purchase"], main="cost=0.1",col ="red ")
```



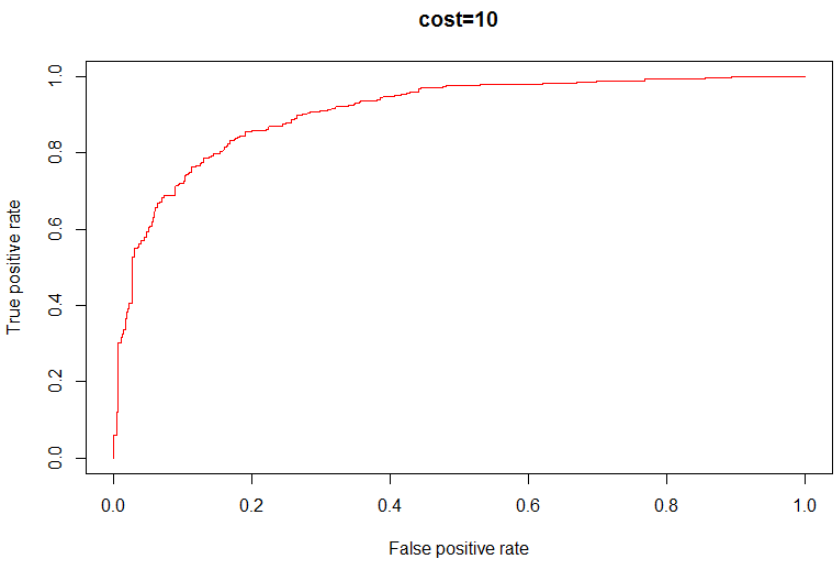
Para cost = 1

```
> svmfit=svm(Purchase~., data=OJ.train, kernel ="linear", cost=1, decision.
values =T)
> fitted =attributes (predict (svmfit ,OJ.train, decision.values =T))$decis
ion.values
> rocplot(fitted ,OJ.train[,"Purchase"], main="cost=1",col ="red ")
```



Para cost = 10

```
> svmfit=svm(Purchase~., data=OJ.train, kernel ="linear", cost=10, decision
.values =T)
> fitted =attributes (predict (svmfit ,OJ.train, decision.values =T))$decis
ion.values
> rocplot(fitted ,OJ.train[,"Purchase"], main="cost=10",col ="red ")
```



4. Calcular las tasas de error de "training" y "test" usando el nuevo valor de coste óptimo.

A continuación se ha calculado la matriz de confusión para las muestras de training usando el nuevo valor de coste óptimo cost=0,01:

```
> svmfit = svm(Purchase ~ ., data = OJ.train, kernel = "linear", cost = 0.1
)
> train.pred = predict(svmfit, OJ.train)
> table(OJ.train$Purchase, train.pred)
    train.pred
      CH  MM
  CH 417  63
  MM  62 258
> mean(train.pred!= OJ.train$Purchase)
[1] 0.15625
```

Usando el nuevo valor de coste óptimo se ha obtenido una nueva tasa de error de entrenamiento del 15,62%, que ha mejorado respecto a los datos obtenidos en el apartado anterior.

A continuación se va a mostrar la matriz de confusión para las muestras de test usando el nuevo valor de coste óptimo:

```
> test.pred = predict(svmfit, OJ.test)
> table(OJ.test$Purchase, test.pred)
    test.pred
      CH  MM
  CH 144  29
  MM  20  77
> mean(test.pred!= OJ.test$Purchase)
[1] 0.1814815
```

Usando el nuevo valor de coste óptimo se ha obtenido una nueva tasa de error de test del 18.14%, que ha mejorado respecto a los datos obtenidos en el apartado anterior.

Los resultados de utilizar el nuevo valor de coste óptimo han mejorado las tasas de error tanto en entrenamiento como en test.

5. Repetir apartados (2) a (4) usando un SVM con núcleo radial. Usar valores de gamma en el rango [10, 1, 0.1, 0.01, 0.001]. Discutir los resultados.

En este apartado se han especificado los valores de gamma en el rango [10, 1, 0.1, 0.01, 0.001] para un SVM con núcleo radial (kernel = "radial").

```
> svmfit.radial=svm(Purchase~., data=OJ.train, kernel ="radial", gamma=c(10
, 1, 0.1, 0.01, 0.001), cost=0.01, decision.values=T)
> summary(svmfit.radial)
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.01
      gamma:  10 1 0.1 0.01 0.001
Number of Support Vectors:  700
 ( 320 380 )
Number of Classes:  2
Levels:
 CH MM
```

En esta ocasión el número de Vectores de Soporte es de 700, a diferencia de los 437 de los apartados anteriores. El conjunto de Vectores de Soporte se encuentra formado por 320 vectores de una clase, y 380 vectores de la otra clase.

Se ha vuelto a utilizar la función tune() para seleccionar un coste óptimo, considerarando los valores de "cost" del vector: [0.001, 0.01, 0.1, 1, 10].

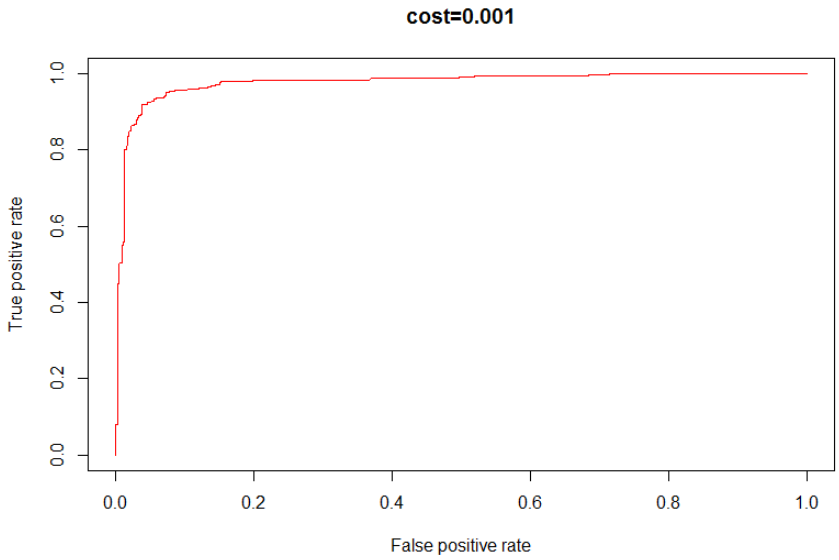
```
> set.seed (3)
> tune.out=tune(svm ,Purchase~.,data=OJ.train ,kernel = "radial",  gamma=c(
10, 1, 0.1, 0.01, 0.001), ranges =list(cost=c(0.001, 0.01, 0.1, 1,10)))
> summary(tune.out)

Parameter tuning of 'svm':
- sampling method: 10-fold cross validation
- best parameters:
 cost
   10
- best performance: 0.2575
- Detailed performance results:
   cost  error dispersion
1 1e-03 0.4000 0.08759915
2 1e-02 0.4000 0.08759915
3 1e-01 0.3925 0.08664262
4 1e+00 0.2600 0.07163759
5 1e+01 0.2575 0.05898446
```

A partir de los resultados, se ha deducido que la tasa de error de validación cruzada más bajo es para cost = 10.

Ahora, es posible dibujar las curvas ROC para los diferentes valores del "cost".

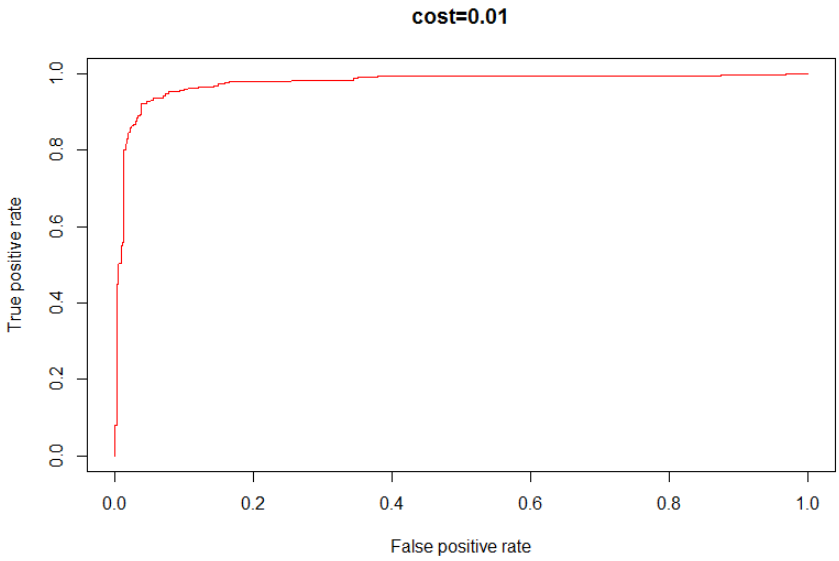
<u>Para cost = 0,001</u>

```
> svmfit.radial=svm(Purchase~., data=OJ.train, kernel ="radial", gamma=c(10
, 1, 0.1, 0.01, 0.001), cost=0.001, decision.values=T)
> fitted =attributes (predict (svmfit.radial ,OJ.train, decision.values =T)
)$decision.values
> rocplot(fitted ,OJ.train[,"Purchase"], main="cost=0.001",col ="red")
```



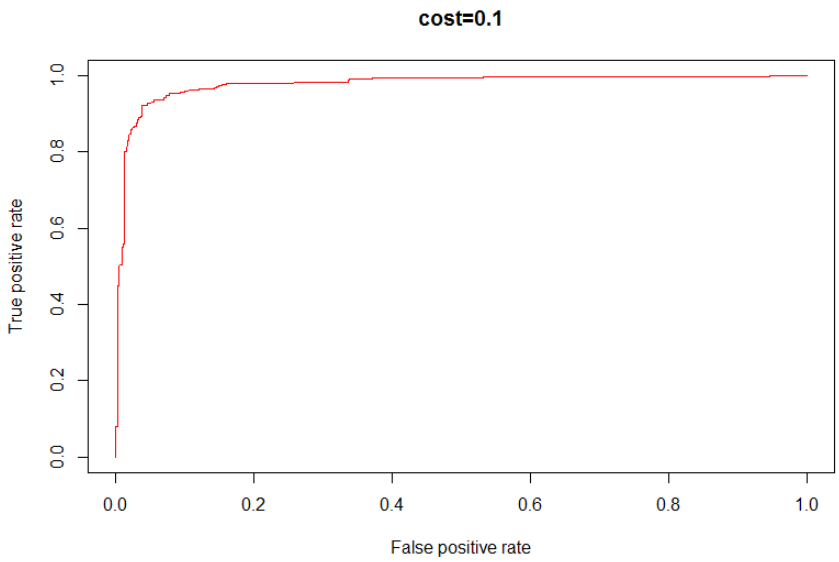
Para cost = 0,01

```
> svmfit.radial=svm(Purchase~., data=OJ.train, kernel ="radial", gamma=c(10
, 1, 0.1, 0.01, 0.001), cost=0.01, decision.values=T)
> fitted =attributes (predict (svmfit.radial ,OJ.train, decision.values =T)
)$decision.values
> rocplot(fitted ,OJ.train[,"Purchase"], main="cost=0.01",col ="red")
```



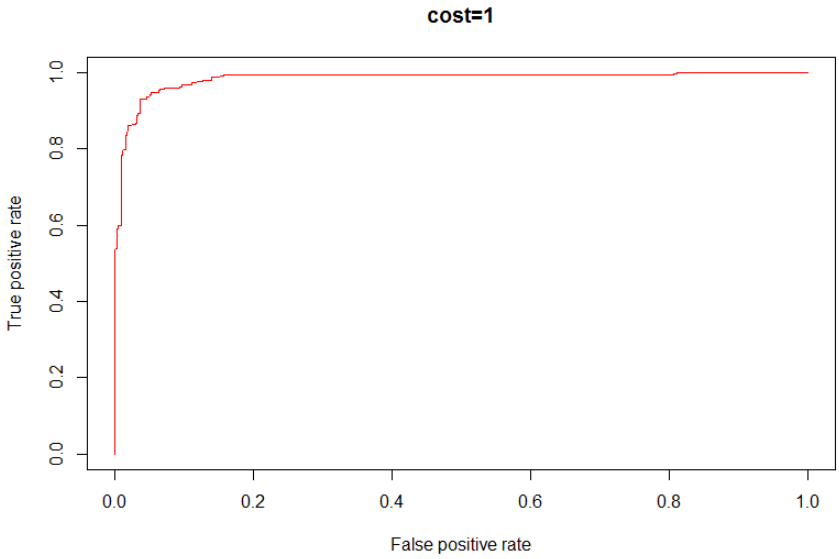
Para cost = 0,1

```
> svmfit.radial=svm(Purchase~., data=OJ.train, kernel ="radial", gamma=c(10
, 1, 0.1, 0.01, 0.001), cost=0.1, decision.values=T)
> fitted =attributes (predict (svmfit.radial ,OJ.train, decision.values =T)
)$decision.values
> rocplot(fitted ,OJ.train[,"Purchase"], main="cost=0.1",col ="red")
```



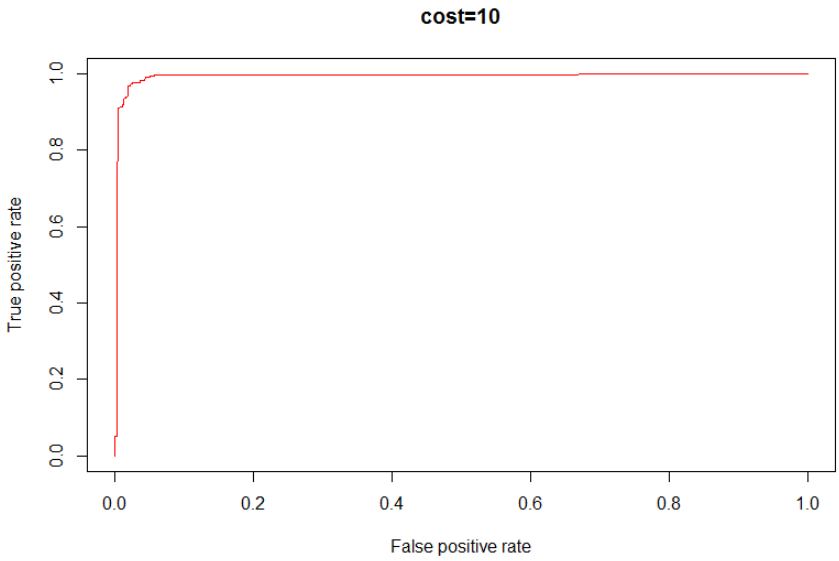
Para cost = 1

```
> svmfit.radial=svm(Purchase~., data=OJ.train, kernel ="radial", gamma=c(10
, 1, 0.1, 0.01, 0.001), cost=1, decision.values=T)
> fitted =attributes (predict (svmfit.radial ,OJ.train, decision.values =T)
)$decision.values
> rocplot(fitted ,OJ.train[,"Purchase"], main="cost=1",col ="red")
```



Para cost = 10

```
> svmfit.radial=svm(Purchase~., data=OJ.train, kernel ="radial", gamma=c(10
, 1, 0.1, 0.01, 0.001), cost=10, decision.values=T)
> fitted =attributes (predict (svmfit.radial ,OJ.train, decision.values =T)
)$decision.values
> rocplot(fitted ,OJ.train[,"Purchase"], main="cost=10",col ="red")
```



Finalmente, se desean calcular las tasas de error de “training” y “test” usando el nuevo valor de coste óptimo cost=10:

En primer lugar, se ha calculado la matriz de confusión para las muestras de entrenamiento usando el nuevo valor de coste óptimo:

```
> svmfit.radial=svm(Purchase~., data=OJ.train, kernel ="radial", gamma=c(10
, 1, 0.1, 0.01, 0.001), cost=10, decision.values=T)
> train.pred = predict(svmfit.radial, OJ.train)
> table(OJ.train$Purchase, train.pred)
    train.pred
      CH  MM
  CH 471   9
  MM  11 309
> mean(train.pred!= OJ.train$Purchase)
[1] 0.025
```

Usando el nuevo valor de coste óptimo se ha obtenido una nueva tasa de error de entrenamiento del 2,5%, que ha mejorado respecto a los datos obtenidos en el apartado anterior.

En segundo lugar, se ha calculado la matriz de confusión para las muestras de test usando el nuevo valor de coste óptimo:

```
> test.pred = predict(svmfit.radial, OJ.test)
> table(OJ.test$Purchase, test.pred)
    test.pred
      CH  MM
  CH 134  39
  MM  35  62
> mean(test.pred!= OJ.test$Purchase)
[1] 0.2740741
```

Usando el nuevo valor de coste óptimo se ha obtenido una nueva tasa de error de test del 27.40%, que ha empeorado respecto a los datos obtenidos en el apartado anterior.

Los resultados obtenidos respecto al apartado anterior, donde el núcleo era lineal, han mejorado respecto a las tasas de error en entrenamiento, pero han empeorado en test.

6. Repetir apartados (2) a (4) usando un SVM con un núcleo polinómico. Usar degree con valores 2,3,4,5,6. Discutir los resultados.

En este apartado se han especificado los valores de degree 2, 3, 4, 5, 6 para un SVM con núcleo polinómico (kernel = "polynomial").

```
> svmfit.polynomial=svm(Purchase~., data=OJ.train, kernel ="polynomial", de
gree=c(2,3,4,5,6), cost=0.01, decision.values=T)
> summary(svmfit.polynomial)
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  0.01
     degree:  2 3 4 5 6
      gamma:  0.05555556
     coef.0:  0
Number of Support Vectors:  644
 ( 320 324 )
```

```
Number of Classes:  2
Levels:
 CH MM
```

En esta ocasión, a diferencia de los apartados anteriores, el número de Vectores de Soporte es de 644. El conjunto de Vectores de Soporte se encuentra formado por 320 vectores de una clase, y 324 vectores de la otra clase.

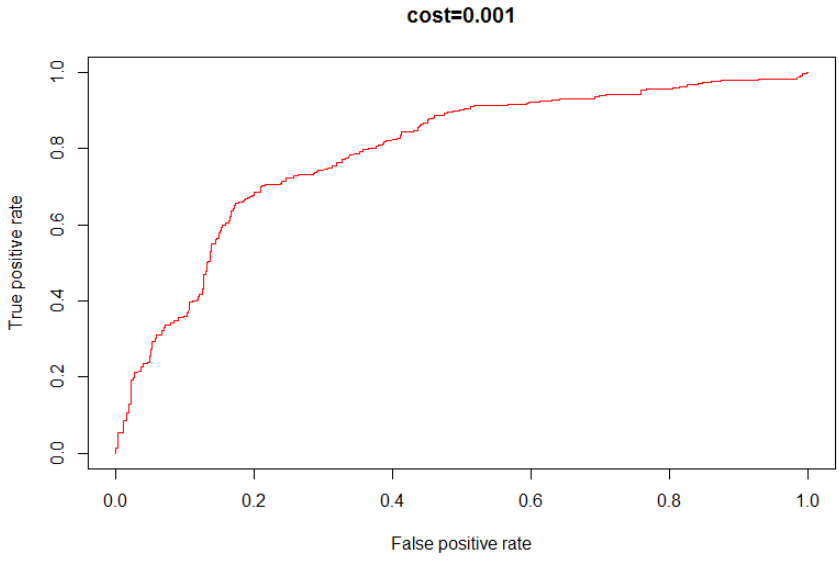
Se ha vuelto a utilizar la función tune() para seleccionar un coste óptimo, considerarando los valores de "cost" del vector: [0.001, 0.01, 0.1, 1, 10].

```
> set.seed (3)
> tune.out=tune(svm ,Purchase~.,data=OJ.train ,kernel = "polynomial", degre
e=c(2,3,4,5,6), ranges=list(cost=c(0.001, 0.01, 0.1, 1,10)))
> summary(tune.out)
Parameter tuning of 'svm':
- sampling method: 10-fold cross validation
- best parameters:
 cost
   10
- best performance: 0.18
- Detailed performance results:
   cost error dispersion
1 1e-03 0.400 0.08759915
2 1e-02 0.400 0.08759915
3 1e-01 0.325 0.05400617
4 1e+00 0.195 0.04721405
5 1e+01 0.180 0.04456581
```

A partir de los resultados, se ha deducido que la tasa de error de validación cruzada más bajo es para cost = 10.
Ahora, es posible dibujar las curvas ROC para los diferentes valores del "cost".

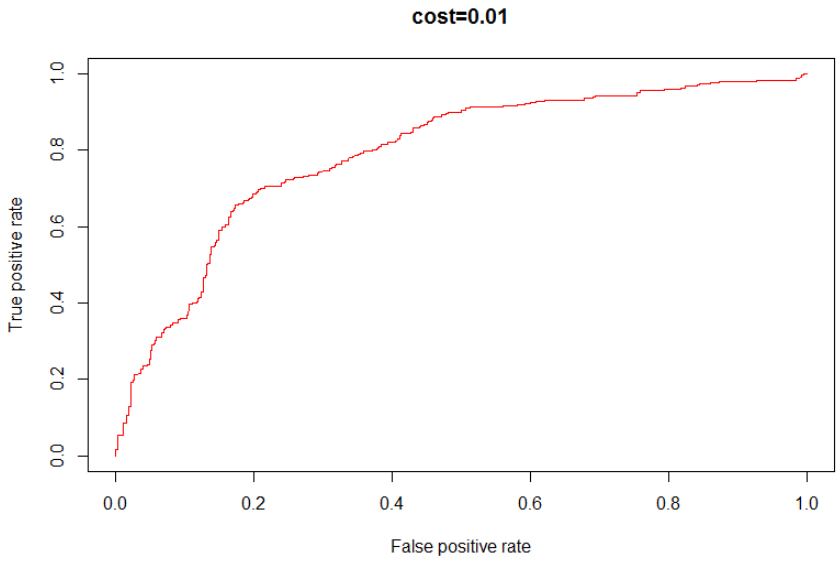
Para cost = 0,001

```
> svmfit.polynomial=svm(Purchase~., data=OJ.train, kernel ="polynomial", de
gree=c(2,3,4,5,6), cost=0.001, decision.values=T)
> fitted =attributes (predict (svmfit.polynomial,OJ.train, decision.values
=T))$decision.values
> rocplot(fitted ,OJ.train[,"Purchase"], main="cost=0.001",col ="red")
```



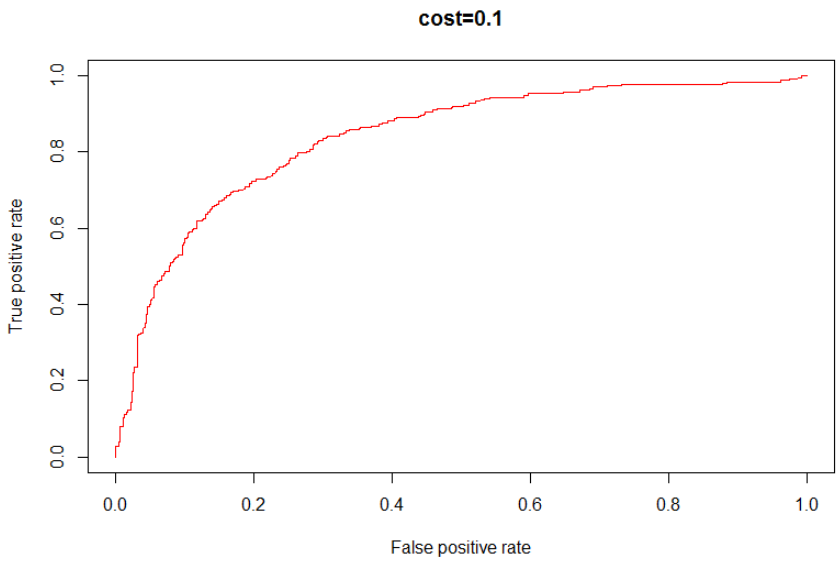
Para cost = 0,01

```
> svmfit.polynomial=svm(Purchase~., data=OJ.train, kernel ="polynomial", de
gree=c(2,3,4,5,6), cost=0.01, decision.values=T)
> fitted =attributes (predict (svmfit.polynomial,OJ.train, decision.values
=T))$decision.values
> rocplot(fitted ,OJ.train[,"Purchase"], main="cost=0.01",col ="red")
```



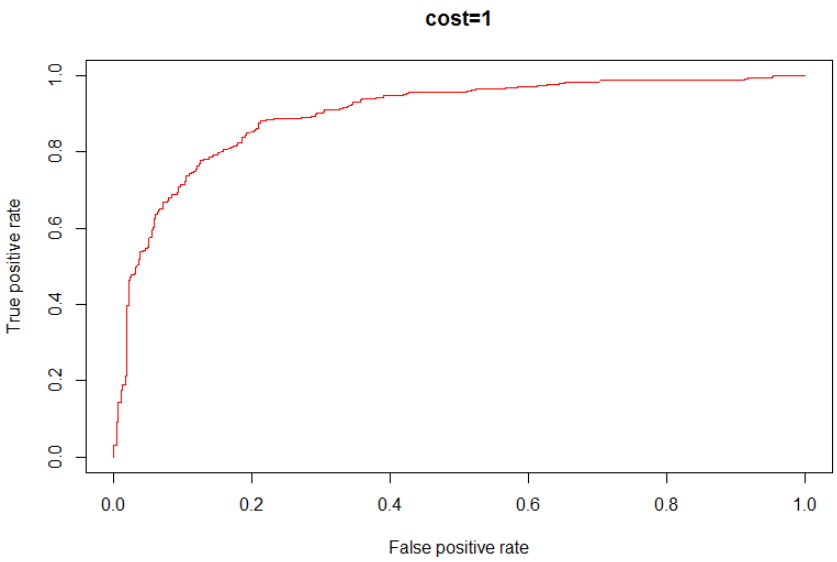
Para cost = 0,1

```
> svmfit.polynomial=svm(Purchase~., data=OJ.train, kernel ="polynomial", de
gree=c(2,3,4,5,6), cost=0.1, decision.values=T)
> fitted =attributes (predict (svmfit.polynomial,OJ.train, decision.values
=T))$decision.values
> rocplot(fitted ,OJ.train[,"Purchase"], main="cost=0.1",col ="red")
```



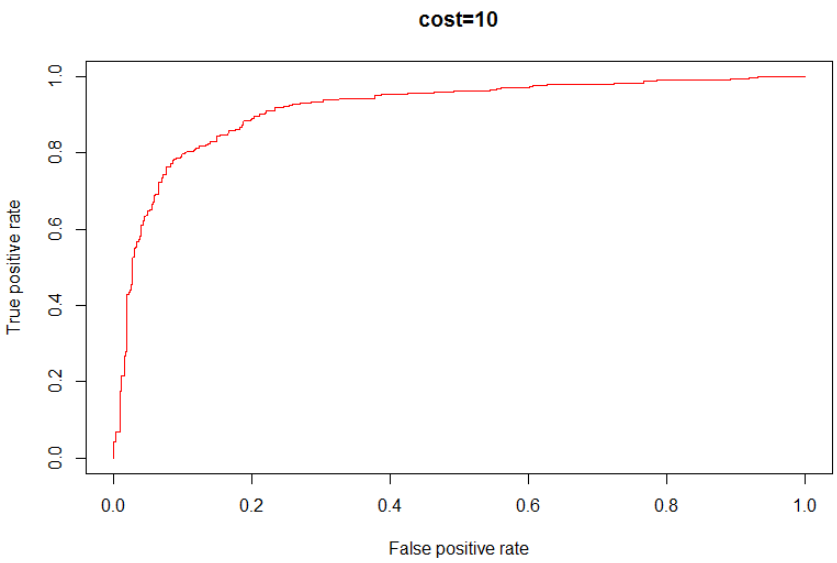
Para cost = 1

```
> svmfit.polynomial=svm(Purchase~., data=OJ.train, kernel ="polynomial", de
gree=c(2,3,4,5,6), cost=1, decision.values=T)
> fitted =attributes (predict (svmfit.polynomial,OJ.train, decision.values
=T))$decision.values
> rocplot(fitted ,OJ.train[,"Purchase"], main="cost=1",col ="red")
```



Para cost = 10

```
> svmfit.polynomial=svm(Purchase~., data=OJ.train, kernel ="polynomial", de
gree=c(2,3,4,5,6), cost=10, decision.values=T)
> fitted =attributes (predict (svmfit.polynomial,OJ.train, decision.values
=T))$decision.values
> rocplot(fitted ,OJ.train[,"Purchase"], main="cost=10",col ="red")
```



Finalmente, se desean calcular las tasas de error de “training” y “test” usando el nuevo valor de coste óptimo cost=10:

En primer lugar, se ha calculado la matriz de confusión para las muestras de entrenamiento usando el nuevo valor de coste óptimo:

```
> svmfit.polynomial=svm(Purchase~., data=OJ.train, kernel ="polynomial", de
gree=c(2,3,4,5,6), cost=10, decision.values=T)
> train.pred = predict(svmfit.polynomial, OJ.train)
> table(OJ.train$Purchase, train.pred)
    train.pred
      CH  MM
  CH 439  41
  MM  73 247
> mean(train.pred!= OJ.train$Purchase)
[1] 0.1425
```

Usando el nuevo valor de coste óptimo se ha obtenido una nueva tasa de error de entrenamiento del 14,25%, que ha mejorado respecto a los datos obtenidos en los apartados anteriores.

En segundo lugar, se ha calculado la matriz de confusión para las muestras de test usando el nuevo valor de coste óptimo:

```
> test.pred = predict(svmfit.polynomial, OJ.test)
> table(OJ.test$Purchase, test.pred)
    test.pred
      CH  MM
  CH 152  21
  MM  30  67
> mean(test.pred!= OJ.test$Purchase)
[1] 0.1888889
```

Usando el nuevo valor de coste óptimo se ha obtenido una nueva tasa de error de test del 18.88%, que ha empeorado respecto a los datos obtenidos en el apartado anterior.

Los resultados obtenidos respecto a la aproximación lineal, han mejorado las tasas de error en entrenamiento y empeorado en test. Respecto a la aproximación radial, han empeorado las tasas de error en entrenamiento y mejorado en test.

7. En global, ¿qué aproximación da el mejor resultado sobre estos datos?

Para evaluar la aproximación que da el mejor resultado, se ha elaborado una tabla con los resultados obtenidos en las matrices de confusión correspondientes a cada núcleo con su valor coste óptimo.

	Lineal	Radial	Polinómico
Error Entrenamiento	15,62%	2,5%	14,25%
Error Test	18,14%	27,40%	18.88%

Los núcleos radial y polinómico presentan mejores resultados en entrenamiento que el núcleo lineal, sin embargo, los resultados de test del núcleo linear son los mejores. Por lo tanto, la aproximación con núcleo lineal da el mejor resultado sobre estos datos.

Ejercicio.-2 (**3 puntos**) (**comentar los resultados de todos los apartados**)

Usar el conjunto de datos OJ que es parte del paquete ISLR

1. Crear un conjunto de entrenamiento conteniendo una muestra aleatoria de 800 observaciones, y un conjunto de test conteniendo el resto de las observaciones. Ajustar un árbol a los datos de "training", usando "Purchase" como la respuesta y las otras variables excepto "Buy" como predictores.

La biblioteca tree se ha utilizado para construir árboles de clasificación y regresión:

```
> library(tree)
```

En este ejercicio se ha vuelto a utilizar la base de datos OJ del apartado anterior, que es parte del paquete ISLR.

```
> ?OJ
```

Primeramente, se ha creado un conjunto de entrenamiento conteniendo una muestra aleatoria de 800 observaciones, y un conjunto de test conteniendo el resto de las observaciones.

```
> set.seed(1)
> train=sample(1:nrow(OJ), 800)
> OJ.train=OJ[train, ]
> OJ.test=OJ[-train, ]
```

Una vez creado el conjunto de entrenaimiento y test, se ha ajustado un árbol a los datos de "training", usando "Purchase" como la respuesta y las otras variables excepto "Buy" como predictores.

Dado que la variable Buy no aparecía en la base de datos, se ha a realizar el ajuste con todas las variables como predictores salvo la variable respuesta "Purchase".

```
> arbol.OJ=tree(Purchase~., data=OJ.train)
```

2. Usar la función summary() para generar un resumen estadístico acerca del árbol y describir los resultados obtenidos: tasa de error de "training", número de nodos del árbol, etc. Teclee el nombre del objeto árbol y obtendrá una salida en texto. Elija un nodo e interprete su contenido.

El cometido de la función de summary() ha sido enumerar las variables que se utilizan como nodos internos del árbol, el número de nodos terminales, y la tasa de error de entrenamiento.

```
> summary(arbol.OJ)

Classification tree:
tree(formula = Purchase ~ ., data = OJ.train)
Variables actually used in tree construction:
[1] "LoyalCH"       "PriceDiff"     "SpecialCH"     "ListPriceDiff"
Number of terminal nodes:  8
Residual mean deviance:  0.7305 = 578.6 / 792
Misclassification error rate: 0.165 = 132 / 800
```

El arbol entrenado ha obtenido un error de entrenamiento de 0,165 y 15 nodos de los cuales 8 son nodos terminales. Parece que la desviación es pequeña, por lo que se puede deducir que el árbol tiene un buen ajuste sobre los datos de entrenamiento.

Las variables implicadas en la construcción del árbol han sido: LoyalCH, PriceDiff, SpecialCH y ListPriceDiff.

Se ha elegido analizar e interpretar el contenido del nodo número 2:

```
> arbol.OJ
2) LoyalCH < 0.508643 350  409.30 MM (0.27143 0.72857 )
```

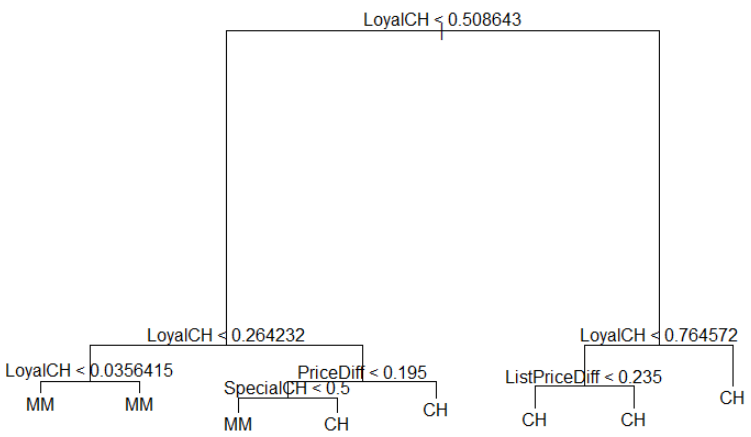
En primera instancia ha sido posible determinar el criterio de división LoyalCH < 0.508943. Los datos que se encuentran por debajo de esta cifra son decantados hacia una rama y los que se encuentran por encima son decantados hacia otra rama diferente. El número de observaciones de la rama actual es de 350 sobre el número total de observaciones para entrenamiento.

Su desviación es de 419.000. La predicción general para esta rama es MM. Y la fracción de observaciones que divide esta rama es de (27.143% y 72.857%).

3. Crear un dibujo del árbol. Extraiga las reglas de clasificación más relevantes definidas por el árbol (al menos 4).

Una de las propiedades más atractivas de árboles es que pueden ser mostrados gráficamente. Para ello se ha utilizado la función plot() que mostrar la estructura de árbol, y la función text() que muestra las etiquetas de nodo.

```
> plot(arbol.OJ)
> text(arbol.OJ, pretty=0)
```



A partir del gráfico ha sido posible comprobar que la variable más relevante para “Purchase” parece ser "LoyalCH", ya que los tres principales nodos utilizan "LoyalCH".

```
> arbol.OJ
node), split, n, deviance, yval, (yprob)
      * denotes terminal node
 1) root 800 1064.00 CH ( 0.61750 0.38250 )
   2) LoyalCH < 0.508643 350  409.30 MM ( 0.27143 0.72857 )
     4) LoyalCH < 0.264232 166  122.10 MM ( 0.12048 0.87952 )
       8) LoyalCH < 0.0356415 57   10.07 MM ( 0.01754 0.98246 ) *
       9) LoyalCH > 0.0356415 109  100.90 MM ( 0.17431 0.82569 ) *
     5) LoyalCH > 0.264232 184  248.80 MM ( 0.40761 0.59239 )
      10) PriceDiff < 0.195 83   91.66 MM ( 0.24096 0.75904 )
        20) SpecialCH < 0.5 70   60.89 MM ( 0.15714 0.84286 ) *
        21) SpecialCH > 0.5 13   16.05 CH ( 0.69231 0.30769 ) *
      11) PriceDiff > 0.195 101  139.20 CH ( 0.54455 0.45545 ) *
   3) LoyalCH > 0.508643 450  318.10 CH ( 0.88667 0.11333 )
     6) LoyalCH < 0.764572 172  188.90 CH ( 0.76163 0.23837 )
      12) ListPriceDiff < 0.235 70   95.61 CH ( 0.57143 0.42857 ) *
      13) ListPriceDiff > 0.235 102   69.76 CH ( 0.89216 0.10784 ) *
     7) LoyalCH > 0.764572 278   86.14 CH ( 0.96403 0.03597 ) *
```

- **2) LoyalCH < 0.508643:** La regla de clasificación del nodo raíz parece relevante. Esta regla de clasificación se basa en la división de las observaciones que tienen un valor inferior a 0.508643 para la variable LoyalCH.

- **4) LoyalCH < 0.264232:** La regla de clasificación de este nodo parece relevante. Esta regla de clasificación se basa en la división de las observaciones que tienen un valor inferior a 0.264232 para la variable LoyalCH.

- **10) PriceDiff < 0.195:** La regla de clasificación de este nodo parece relevante. Esta regla de clasificación se basa en la división de las observaciones que tienen un valor inferior a 0.195 para la variable PriceDiff.

- **6) LoyalCH < 0.764572:** La regla de clasificación de este nodo parece relevante. Esta regla de clasificación se basa en la división de las observaciones que tienen un valor inferior a 0.764572 para la variable LoyalCH.

4. Predecir la respuesta de los datos de test, y generar e interpretar la matriz de confusión de los datos de test. ¿Cuál es la tasa de error del test? ¿Cuál es la precisión del test?

```
> tree.pred=predict(arbol.OJ, OJ.test, type = "class")
> table(tree.pred, OJ.test$Purchase)
tree.pred  CH  MM
       CH 147  49
       MM  12  62
```

Los resultados obtenidos por la matriz de confusión indican que se han clasificado correctamente 147 observaciones como CH y 62 como MM. Y por otro lado, que se han clasificado erróneamente 12 observaciones como CH y 49 como MM.

```
> mean(tree.pred!=OJ.test$Purchase)
[1] 0.2259259
```

Se ha obtenido una tasa de error de test del 22.59%.

```
> mean(tree.pred==OJ.test$Purchase)
[1] 0.7740741
```

Se ha obtenido una tasa de precisión de test del 77.40%.

5. Aplicar la función cv.tree() al conjunto de "training" para determinar el tamaño óptimo del árbol.

Con la función cv.tree() se ha realizado la validación cruzada con el fin de determinar el nivel óptimo de la complejidad del árbol; el coste de la complejidad de la poda se utiliza para seleccionar una secuencia de árboles a considerar. La función indica el número de nodos terminales de cada árbol considerado (tamaño), así como la tasa de error y el valor del costo-complejidad utilizado.

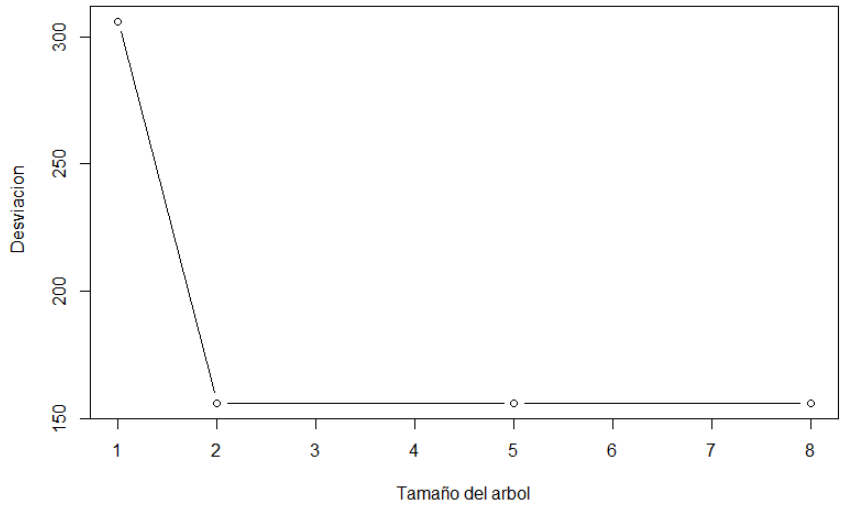
```
> cv.OJ=cv.tree(arbol.OJ, FUN = prune.misclass)
> cv.OJ
$size
[1] 8 5 2 1
$dev
[1] 156 156 156 306
$k
[1]       -Inf   0.000000   4.666667 160.000000
$method
[1] "misclass"
attr(,"class")
[1] "prune"         "tree.sequence"
```

A partir de los resultados obtenidos, se ha deducido que para un árbol de tamaño 8 se obtiene la tasa de error de validación cruzada más baja, al igual que el árbol de tamaño 2 es el óptimo dado que es más pequeño.

6. Generar un gráfico con el tamaño del árbol en el eje x y la tasa de error de validación cruzada en el eje y. ¿Qué tamaño de árbol corresponde a la tasa más pequeña de error de clasificación por validación cruzada?

En este apartado se ha generado un gráfico con el tamaño del árbol en el eje x y la tasa de error de validación cruzada en el eje y:

```
> plot(cv.OJ$size, cv.OJ$dev, type="b", xlab = "Tamaño del arbol", ylab = "
Desviacion")
```

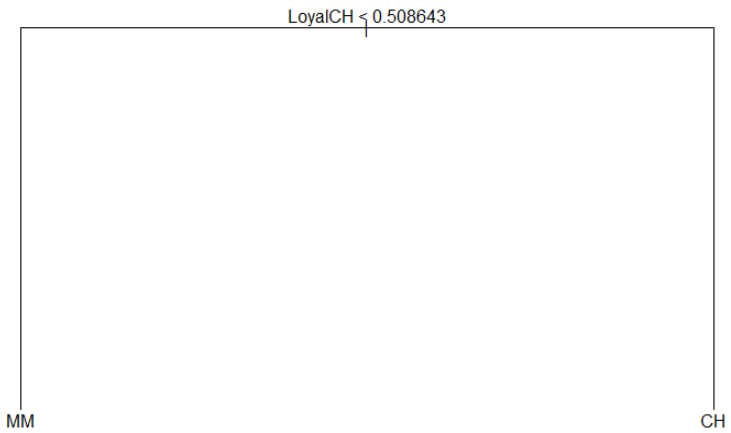


El tamaño de árbol que corresponde a la tasa más pequeña de error de clasificación por validación cruzada es 2, 5 y 8.

7. Ajustar el árbol podado correspondiente al valor óptimo obtenido en 6. Comparar los errores sobre el conjunto de training y test de los árboles ajustados en 6 con el árbol podado. ¿Cuál es mayor?

Para ajustar el árbol podado correspondiente al valor óptimo obtenido en el apartado anterior, se ha aplicado la función prune.misclass() para podar el árbol y obtener el árbol de dos nodos:

```
> prune.OJ=prune.misclass(arbol.OJ, best = 2)
> plot(prune.OJ)
> text(prune.OJ, pretty = 0)
```



En el siguiente apartado se han comparado los errores sobre el conjunto de training y test de los árboles ajustados en el apartado anterior con el árbol podado.

Conjunto de Training:

```
> tree.pred=predict(prune.OJ, OJ.train, type = "class")
> table(tree.pred, OJ.train$Purchase)
tree.pred  CH  MM
       CH 399  51
       MM  95 255
> mean(tree.pred!=OJ.train$Purchase)
[1] 0.1825
```

Se ha obtenido una tasa de error del 18.25% para los datos de training.

Conjunto de Test:

```
> tree.pred=predict(prune.OJ, OJ.test, type = "class")
> table(tree.pred, OJ.test$Purchase)
tree.pred  CH  MM
       CH 119  30
       MM  40  81
> mean(tree.pred!=OJ.test$Purchase)
[1] 0.2592593
```

Se ha obtenido una tasa de error del 25.92% para los datos de test.

En los apartados anteriores se ha obtenido una tasa de error en training del 0,16% y una tasa de error en test del 22,59% . A la vista de los resultados, es posible concluir que la tasa de error de clasificación es levemente superior con el árbol podado.

Ejercicio.-3 (**3 puntos**) (**comentar los resultados de todos los apartados**)

Usar el conjunto de datos Hitters

1. Eliminar las observaciones para las que la información del salario es desconocido y aplicar una transformación logarítmica al resto de valores de salario. Crear un conjunto de "training" con 200 observaciones y un conjunto de "test" con el resto

La biblioteca gbm se ha utilizado para utilizar boosting mediante la función gbm() y entrenar los árboles de regresión:

```
> library(gbm)
```

La base de datos Hitters contiene los datos de las temporadas 1986 y 1987 de la Liga Mayor de Béisbol. Contiene un conjunto de 322 observaciones de los principales jugadores de la liga, distribuidos en 20 variables como por ejemplo: Número de golpes, número de home runs, número de carreras, etc.

```
> data(Hitters)
> attach(Hitters)
> ?Hitters
```

En primer lugar, han sido eliminadas las observaciones para las que la información del salario es desconocido:

```
> Hitters=na.omit(Hitters)
> Hitters
```

En segundo lugar, se ha aplicado una transformación logarítmica al resto de valores de salario:

```
> Hitters$Salary=log(Hitters$Salary)
> Hitters$Salary
```

Y finalmente, se ha creado un conjunto de "training" con 200 observaciones y un conjunto de "test" con el resto:

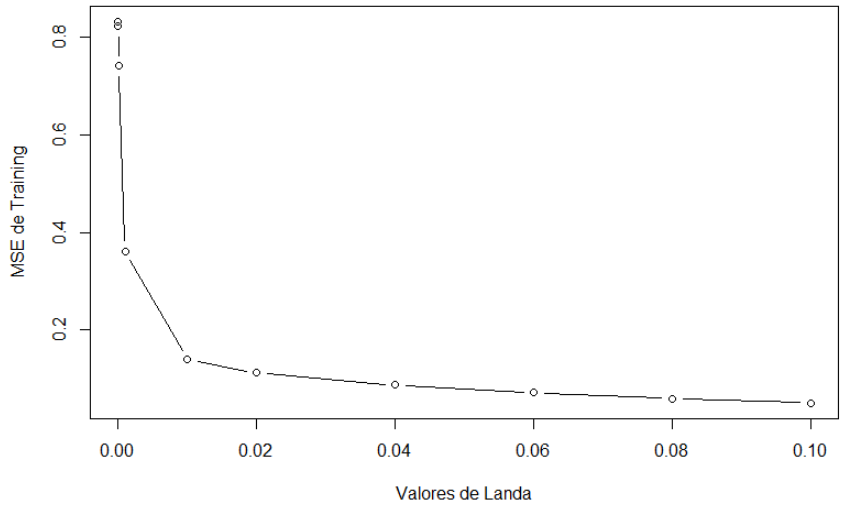
```
> train=1:200
> Hitters.train=Hitters[train, ]
> Hitters.test=Hitters[-train, ]
```

2. Realizar boosting sobre el conjunto de entrenamiento con 1,000 árboles para un rango de valores del parámetro de ponderación λ. Realizar un gráfico con el eje x mostrando diferentes valores de λ y los correspondientes valores de MSE de "training" sobre el eje y.

Como ya se ha argumentado anteriormente, para realizar boosting se ha empleado la función gbm() que entrena los árboles de regresión de la base de datos de Hitters. El argumento "n.trees=1000" implica 1,000 árboles tal y como indica el enunciado del problema.

```
> set.seed(1)
> landa <- c(0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.02, 0.04,
0.06, 0.08, 0.1)
> train.error <- rep(NA, 11)
> for (i in 1:11) {
```

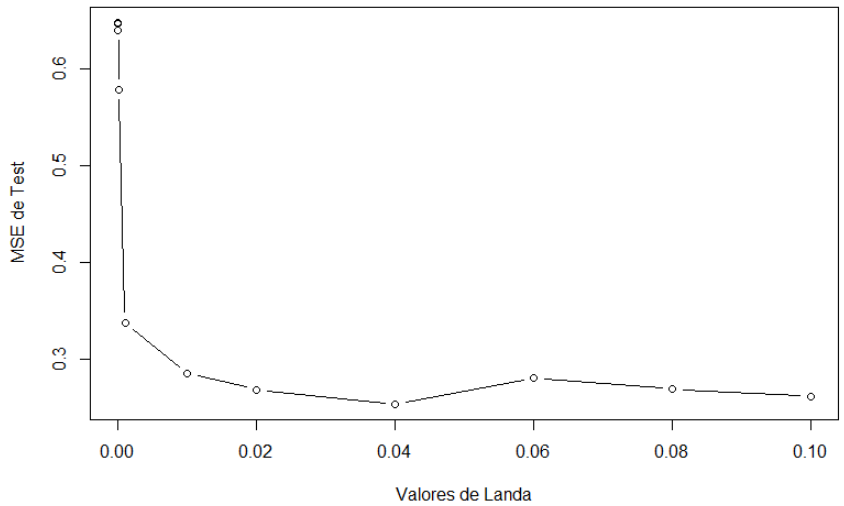
```
+     boost.hitters=gbm(Salary~., data = Hitters.train, distribution ="gaus
sian", n.trees=1000, shrinkage=landa[i])
+     yhat.boost=predict(boost.hitters, Hitters.train, n.trees=1000)
+     train.error[i]=mean((yhat.boost-Hitters.train$Salary)^2)
+ }
> plot(landa, train.error, type="b", xlab="Valores de Landa", ylab="MSE de
Training")
```



A partir del gráfico se ha deducido que el valor más pequeño de MSE se obtiene con λ = 0.10, es decir, que para valores de λ mayores, parece que el MSE de training es menor.

3. Realizar el mismo gráfico del punto anterior pero usando los valores de MSE del conjunto de test. Comparar los valores de MSE obtenidos con boosting para el conjunto test con los obtenidos con los métodos de regresión múltiple y LASSO respectivamente para los mismos datos.

```
> set.seed(1)
> landa <- c(0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.02, 0.04,
0.06, 0.08, 0.1)
> train.error <- rep(NA, 11)
> for (i in 1:11) {
+     boost.hitters=gbm(Salary~., data = Hitters.train, distribution ="gaus
sian", n.trees=1000, shrinkage=landa[i])
+     yhat.boost=predict(boost.hitters, Hitters.test, n.trees=1000)
+     train.error[i]=mean((yhat.boost-Hitters.test$Salary)^2)
+ }
> plot(landa, train.error, type="b", xlab="Valores de Landa", ylab="MSE de
Test")
```

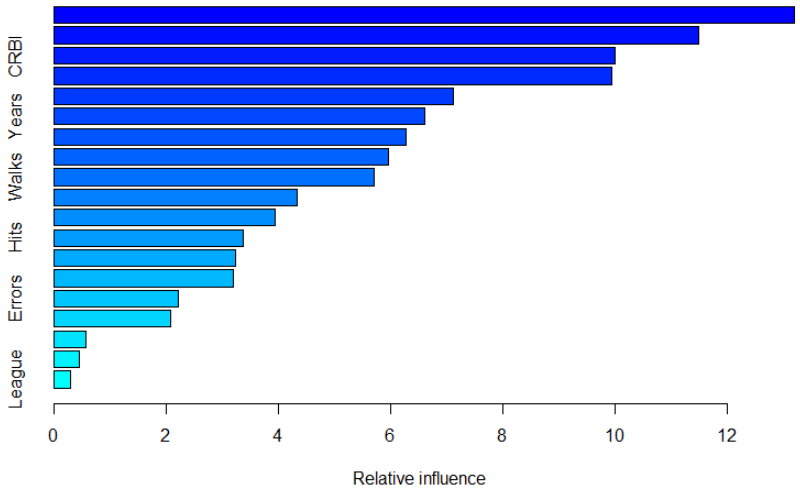


A partir del gráfico se ha deducido que el valor más pequeño de MSE se obtiene con λ = 0.4, es decir, que para este valor de λ, parece que el MSE de test es menor.

4. ¿Qué variables aparecen como las más importantes en el modelo de "boosting"?

Para determinar qué variables aparecen como las más importantes en el modelo de "boosting", se ha escogido el modelo con menor MSE para test.

```
> boost.04=gbm(Salary~.,data = Hitters.train, distribution = "gaussian", n.
trees = 1000, shrinkage=0.04)
> summary(boost.04)
                  var    rel.inf
CAtBat         CAtBat 13.1986818
CRuns           CRuns 11.4962593
CRBI             CRBI 10.0025599
CWalks         CWalks  9.9404224
CHits           CHits  7.1119811
Years           Years  6.6080486
CHmRun         CHmRun  6.2684526
PutOuts       PutOuts  5.9627608
Walks           Walks  5.6999231
RBI               RBI  4.3418645
AtBat           AtBat  3.9501304
Hits             Hits  3.3639294
HmRun           HmRun  3.2366136
Assists       Assists  3.2024743
Errors         Errors  2.2067761
Runs             Runs  2.0810900
Division     Division  0.5766658
NewLeague NewLeague  0.4485355
League         League  0.3028308
```



A la vista de los resultados y del gráfico, se ha deducido que las variables más importantes serian "CAtBat" y "CRuns", puesto que presentan los valores más altos en "rel.inf".

5. Aplicar bagging al conjunto de "training" y volver a estimar el modelo. ¿Cuál es el valor de MSE para el conjunto de test en este caso?

En este apartado se ha aplicado bagging al conjunto de "training" y se ha vuelto a estimar el modelo. Para lograr tal fin, se ha aplicado bagging y random forests a los datos de Hitters. El argumento mtry = 19 ha establecido que los 19 predictores deben ser considerados para cada división del árbol.

```
> library (randomForest)
> set.seed (1)
> bag.Hitters=randomForest(Salary~., data=Hitters.train, mtry=19, importace
=TRUE)
> bag.Hitters

Call:
 randomForest(formula = Salary ~ ., data = Hitters.train, mtry = 19,      i
mportace = TRUE)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 19

          Mean of squared residuals: 0.2181427
                    % Var explained: 73.78
```

Según los resultados obtenidos, parece ser que se han utilizado 500 árboles, que el valor MSR ha sido de 0.2181427 y que el porcentaje de variables explicadas ha sido del 73.78%.

```
> yhat.bag=predict(bag.Hitters, newdata=Hitters.test)
> mean((yhat.bag-Hitters.test$Salary)^2)
[1] 0.2274529
```

El valor MSE para el conjunto de test, en este caso, ha sido de 0.2274529.