Hoja de Trabajo 05

Gustavo de León y Andrés Urízar

3/24/2020

Objetivo

Predecir los precios con el algoritmo de Naives Bayes

```
  library(e1071)
  library(caret)
  library(plyr)
  library(dplyr)
  library(lattice)
  library(rpart)
```

Ubicacion de archivos en computadoras

```
  setwd("C:/Users/Gustavo/Desktop/SEPTIMO SEMESTRE/MINERIA/HDT5/Hoja-de-trabajo-5")
  #setwd("C:/Users/alber/Documents/UVG/Septimo semestre/Mineria de Datos/Hoja-Trabajo-5/Hoja-de-trabajo-5")
```

Selección de variables

Se seleccionarán variables que consideramos importantes para la decisión del precio de una casa. Se usa la misma semilla 123 para repetir el conjunto de entrenamiento y de prueba

```
  set.seed(123)
  datos <- read.csv("train.csv", stringsAsFactors = FALSE)

trainImportantes <- datos[c("MSSubClass","LotFrontage","LotArea","OverallCond","YearBuilt","YearRemodAdd","X2ndFlrSF","FullBath","TotRmsAbvGrd","GarageCars","SalePrice")]
trainImportantes[is.na(trainImportantes)]<-0
```

Clusters

Se harán 3 clusters (probados en hojas anteriores) para crear la variable categórica Con el Clustering se llegó a una variable categórica llamada grupos, esta variable se separa en barato, intermedio y caro. Barato tuvo el rango de 34,900 a 173000 Intermedio el rango de 173500 a 294000 Caro el rango de 295000 a 755000

```
km<-kmeans(trainImportantes,3)
trainImportantes$grupo<-km$cluster
```

```
trainImportantes$grupo <- mapvalues(trainImportantes$grupo, c(1,2,3), c("
Intermedio","Barato","caro"))
```

Se harán la división 70% para train y 30% para test

```
  porcentaje<-0.7
  corte <- sample(nrow(trainImportantes),nrow(trainImportantes)*porcentaj
e)
  train<-trainImportantes[corte,]
  test<-trainImportantes[-corte,]
```

Se realiza el calculo de naives bayes

```
  modelo<-naiveBayes(as.factor(grupo)~.,data=trainImportantes)
```

Matriz de confusion

```
  predBayes<-predict(modelo, newdata = test[,1:11])
  confusionMatrix(table(predBayes,test$grupo))
```

```
## Confusion Matrix and Statistics
##
##
## predBayes    Barato caro Intermedio
##   Barato        220    1         14
##   caro            0   34          8
##   Intermedio     17    2        143
##
## Overall Statistics
##
##                Accuracy : 0.9043
##                  95% CI : (0.8729, 0.9302)
##     No Information Rate : 0.5399
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8304
##
##  Mcnemar's Test P-Value : 0.18
##
## Statistics by Class:
##
##                      Class: Barato Class: caro Class: Intermedio
## Sensitivity                 0.9283     0.91892            0.8667
## Specificity                 0.9257     0.98010            0.9307
## Pos Pred Value              0.9362     0.80952            0.8827
## Neg Pred Value              0.9167     0.99244            0.9206
## Prevalence                  0.5399     0.08428            0.3759
## Detection Rate              0.5011     0.07745            0.3257
## Detection Prevalence        0.5353     0.09567            0.3690
## Balanced Accuracy           0.9270     0.94951            0.8987
```

Como se puede observar de 235 casas baratas acertó en 220 y falló diciendo que 1 es cara y 14 son intermedias. Como se puede observar de 42 casas caras acertó en 34 y no falló diciendo que alguna es barata y falló con 8 que son intermedias. Como se puede observar de 162 casas intermedias acertó en 143 y falló diciendo que 2 son caras y 17 que son intermedias. El modelo tuvo un 0.90 de Accurancy con 95% de índice de confianza puede ondular de 0.87 a 0.93. Analizando estos resultados se puede apreciar que solo hay un error del tipo que una casa sea barata y el modelo haya predicho que sea cara, y no hay ni uno que diga que sea barata cuando realmente es cara esto hace muy bueno el modelo. Los errores de intermedios son más comunes ya que está en medio de caro y barato, pero igual no fue muchos errores. Tiene una sensivity de 0.92 con las casas baratas y 0.918 con las caras. Con un Specifity de 0.92 para las baratas, 0.93 para las intermedias y 0.98 para las caras.

Overfitting

Se concluye que el modelo no tiene overfitting debido a que se ajustó bien con los datos de test como se habló previamente.

Cross Validation

```
## + Fold01: usekernel= TRUE, fL=0, adjust=1
## - Fold01: usekernel= TRUE, fL=0, adjust=1
## + Fold01: usekernel=FALSE, fL=0, adjust=1
## - Fold01: usekernel=FALSE, fL=0, adjust=1
## + Fold02: usekernel= TRUE, fL=0, adjust=1
## - Fold02: usekernel= TRUE, fL=0, adjust=1
## + Fold02: usekernel=FALSE, fL=0, adjust=1
## - Fold02: usekernel=FALSE, fL=0, adjust=1
## + Fold03: usekernel= TRUE, fL=0, adjust=1
## - Fold03: usekernel= TRUE, fL=0, adjust=1
## + Fold03: usekernel=FALSE, fL=0, adjust=1
## - Fold03: usekernel=FALSE, fL=0, adjust=1
## + Fold04: usekernel= TRUE, fL=0, adjust=1
## - Fold04: usekernel= TRUE, fL=0, adjust=1
## + Fold04: usekernel=FALSE, fL=0, adjust=1
## - Fold04: usekernel=FALSE, fL=0, adjust=1
## + Fold05: usekernel= TRUE, fL=0, adjust=1
## - Fold05: usekernel= TRUE, fL=0, adjust=1
## + Fold05: usekernel=FALSE, fL=0, adjust=1
## - Fold05: usekernel=FALSE, fL=0, adjust=1
## + Fold06: usekernel= TRUE, fL=0, adjust=1
## - Fold06: usekernel= TRUE, fL=0, adjust=1
## + Fold06: usekernel=FALSE, fL=0, adjust=1
## - Fold06: usekernel=FALSE, fL=0, adjust=1
## + Fold07: usekernel= TRUE, fL=0, adjust=1
## - Fold07: usekernel= TRUE, fL=0, adjust=1
## + Fold07: usekernel=FALSE, fL=0, adjust=1
## - Fold07: usekernel=FALSE, fL=0, adjust=1
## + Fold08: usekernel= TRUE, fL=0, adjust=1
## - Fold08: usekernel= TRUE, fL=0, adjust=1
```

```
## + Fold08: usekernel=FALSE, fL=0, adjust=1
## - Fold08: usekernel=FALSE, fL=0, adjust=1
## + Fold09: usekernel= TRUE, fL=0, adjust=1
## - Fold09: usekernel= TRUE, fL=0, adjust=1
## + Fold09: usekernel=FALSE, fL=0, adjust=1
## - Fold09: usekernel=FALSE, fL=0, adjust=1
## + Fold10: usekernel= TRUE, fL=0, adjust=1
## - Fold10: usekernel= TRUE, fL=0, adjust=1
## + Fold10: usekernel=FALSE, fL=0, adjust=1
## - Fold10: usekernel=FALSE, fL=0, adjust=1
## Aggregating results
## Selecting tuning parameters
## Fitting fL = 0, usekernel = TRUE, adjust = 1 on full training set
```

Matriz de confusion CV

```
prediccionCaret<-predict(modeloCaret,newdata = test[,1:11])
confusionMatrix(table(prediccionCaret,test$grupo))
```

```
## Confusion Matrix and Statistics
##
##
## prediccionCaret Barato caro Intermedio
##      Barato        223    2          6
##      caro            0   33          1
##      Intermedio     14    2        158
##
## Overall Statistics
##
##                Accuracy : 0.9431
##                  95% CI : (0.9171, 0.9628)
##     No Information Rate : 0.5399
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8984
##
##  Mcnemar's Test P-Value : 0.1367
##
## Statistics by Class:
##
##                      Class: Barato Class: caro Class: Intermedio
## Sensitivity                 0.9409     0.89189            0.9576
## Specificity                 0.9604     0.99751            0.9416
## Pos Pred Value              0.9654     0.97059            0.9080
## Neg Pred Value              0.9327     0.99012            0.9736
## Prevalence                  0.5399     0.08428            0.3759
## Detection Rate              0.5080     0.07517            0.3599
## Detection Prevalence        0.5262     0.07745            0.3964
## Balanced Accuracy           0.9507     0.94470            0.9496
```

El modelo con validación cruzada tuvo un mejor Accurancy con un 0.94 sin embargo tuvo un poco más de problemas con el caro y barato como se verá a continuación.

Como se puede observar de 231 casas baratas acertó en 223 y falló diciendo que 6 son caras y 2 son intermedias. Como se puede observar de 174 casas caras acertó en 158 y falló diciendo que 14 son baratas y falló con 2 que son intermedias. Como se puede observar de 34 casas intermedias acertó en 33 y no falló con las baratas y 1 cara.

El modelo tuvo un 0.94 de Accurancy con 95% de índice de confianza puede ondular de 0.91 a 0.96. Analizando estos resultados se puede apreciar que tuvo más problemas colocando alguna casa cara como barata o viceversa, sin embargo, le pegó a una gran cantidad de casas de manera correcta. Este modelo tuvo 0.98 de Specificity para las casas caras y 0.92 para las baratas teniendo buen balance para predecir.

Se realiza la prediccion con un árbol de clasificación

```
trainImportantes$SalePrice<-NULL
trainRowsNumber<-sample(1:nrow(trainImportantes),porcentaje*nrow(trainImp
ortantes))
train1<-trainImportantes[trainRowsNumber,]
test1<-trainImportantes[-trainRowsNumber,]
dt_model<-rpart(grupo~.,train1,method = "class")
prediccion <- predict(dt_model, newdata = test1[,1:10])
columnaMasAlta<-apply(prediccion, 1, function(x) colnames(prediccion)[whi
ch.max(x)])
test1$prediccion<-columnaMasAlta
cfm<-confusionMatrix(table(test1$prediccion, test1$grupo))
cfm

## Confusion Matrix and Statistics
##
##
##              Barato caro Intermedio
##   Barato        235    2         47
##   caro            0   23         12
##   Intermedio      9   10        101
##
## Overall Statistics
##
##                Accuracy : 0.8178
##                  95% CI : (0.7784, 0.8528)
##     No Information Rate : 0.5558
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.659
##
##  Mcnemar's Test P-Value : 3.689e-06
##
## Statistics by Class:
##
##                      Class: Barato Class: caro Class: Intermedio
```

```
## Sensitivity                 0.9631     0.65714           0.6312
## Specificity                 0.7487     0.97030           0.9319
## Pos Pred Value              0.8275     0.65714           0.8417
## Neg Pred Value              0.9419     0.97030           0.8150
## Prevalence                  0.5558     0.07973           0.3645
## Detection Rate              0.5353     0.05239           0.2301
## Detection Prevalence        0.6469     0.07973           0.2733
## Balanced Accuracy           0.8559     0.81372           0.7816
```

Al realizar la predicción con un árbol de clasificación se obtuvo un Accuracy de 0.81, siendo inferior al Accuracy de Naives Bayes que fue de 0.90, con lo cual se puede concluir que para predicción es mejor naives bayes. Pero al comparar los tiempos con un Profiler se mostró que el árbol de clasificación tomaba 10ms en ejecutarse, mientras que naives bayes tomo 100ms, hay una diferencia grande entre los tiempos de ejecución de ambos algoritmos de predicción, siendo el árbol de clasificación el ganador. Depende de la persona que realice los calculos, si prefiere más precisión para predicción o prefiere resultados en un tiempo más corto, que con un dataset mucho más grande quizá la diferencia de tiempos si sea aún más grande, pero lo más importante para estos algoritmos es tener una predicción exacta.