UNIVERSIDAD DE GRANADA

Clasificación

Carlos Morales Aguilera

28/11/2020

Lectura de datos

El primer paso en el problema es leer correctamente los datos, para ello se utiliza la función `read_excel` de la librería **readxl**, a continuación se transforma el conjunto de datos obtenido en una estructura del tipo **dataframe** propia de *R*.

```
f <- "C:\\Users\\power\\Desktop\\TID\\practica3\\eBayAuctions.xls"
# Read dataframe
bd_eBay <- as.data.frame(read_excel(f))
```

Una de las buenas prácticas aprendidas de las prácticas anteriores es la de eliminar valores perdidos, por lo que previamente a tratar con los datos, nos aseguramos de eliminar los posibles valores perdidos si hubieran.

```
# Omit NAs
bd_eBay <- na.omit(bd_eBay)
```

Tras cargar los datos, visualizamos cuantas instancias hay de cada clase utilizando para ello la primera variable del conjunto de datos, referente a la competitividad de una subasta.

```
# Group by class
classes <- table(bd_eBay$`Competitive?`)
classes
```

```
##
##    0    1
##  906 1066
```

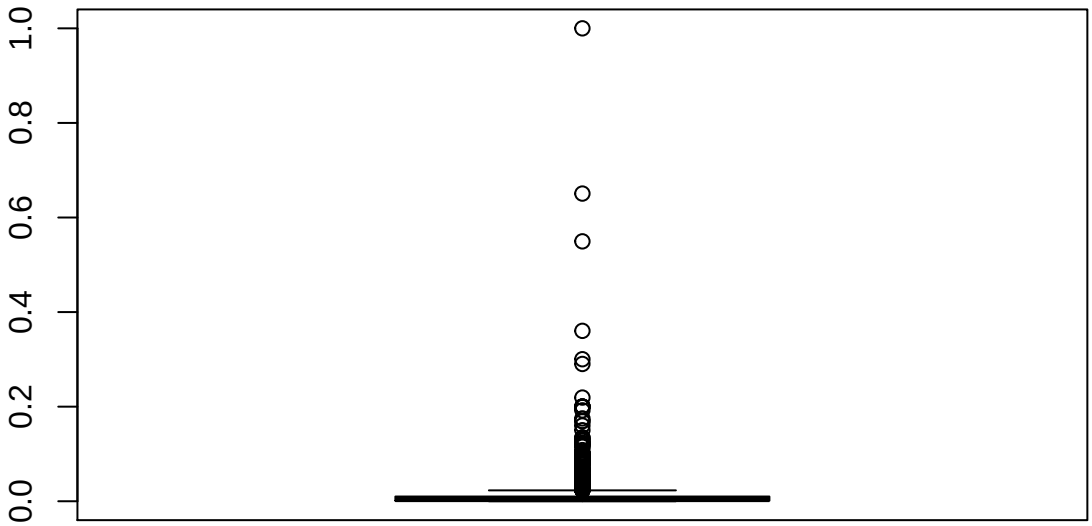
Preprocesamiento de los datos

El primer paso consiste en normalizar las variables númericas que posee el conjunto, para ello se emplea una normalización aplicando el algoritmo **Min-max**, para poder trabajar con los datos de forma adecuada.

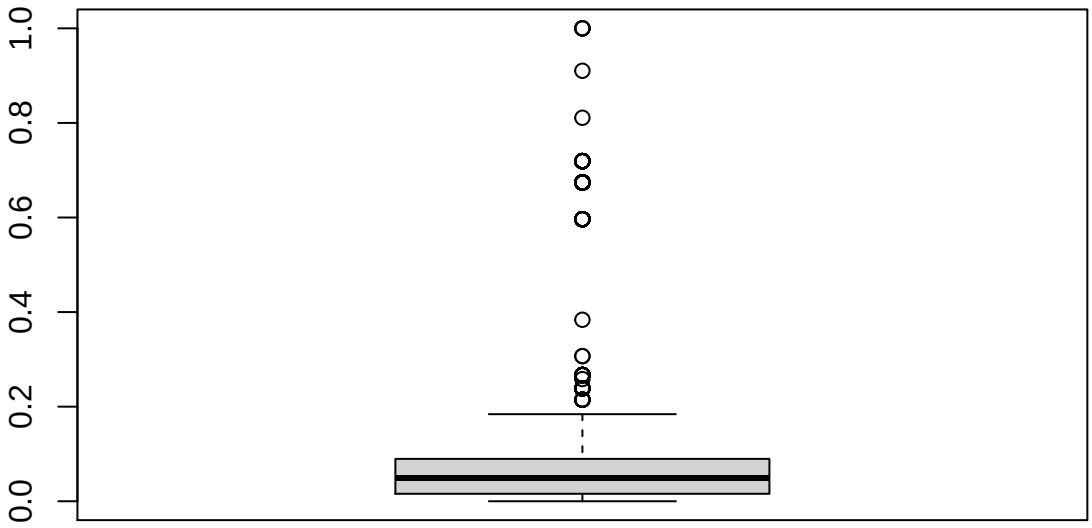
```
# Definition of min_max normalization function
min_max_norm <- function(x) {
    (x - min(x)) / (max(x) - min(x))
}

# Apply normalization to our dataframe
bd_eBay$sellerRating <- min_max_norm(bd_eBay$sellerRating)
bd_eBay$Duration <- min_max_norm(bd_eBay$Duration)
bd_eBay$ClosePrice <- min_max_norm(bd_eBay$ClosePrice)
bd_eBay$OpenPrice <- min_max_norm(bd_eBay$OpenPrice)
```

A continuación, se procede a evaluar los posibles *outliers* de las variables numéricas, para ello se realizan **boxplots** de las variables numéricas, y a continuación se evaluan los posibles *outliers* para determinar si son casos especiales o atípicos.

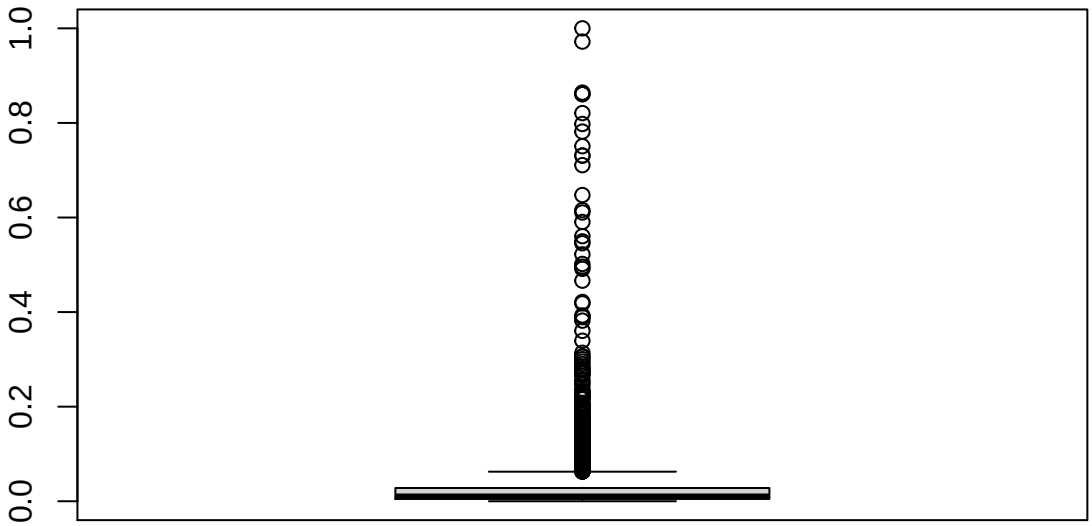
```
# Boxplot OpenPrice
boxplot(bd_eBay$OpenPrice)
```




```
# Boxplot sellerRating
boxplot(bd_eBay$sellerRating)
```




```
# Boxplot ClosedPrice
boxplot(bd_eBay$ClosePrice)
```



Tras analizar los posibles *outliers* visualizados, podemos observar que realmente se tratan de casos especiales, pero no se encuenta ningún valor que se pueda considerar realmente atípico en el contexto del problema, por lo que no se eliminará ninguno de los elementos.

Una vez examinados los *outliers*, se procede a evaluar las variables categóricas, de cara a poder ser procesadas fácilmente por los siguientes modelos, por lo que se ha optado por hacer una transformación en Dummy variables.

Para ello se utiliza la función dummy_cols del paquete fastDummies, con el que las variables categóricas se convierten en variables *dummy*, por ejemplo, la variable **endDay** con valores (*Mon*, *Tue*, *Wed*, *Thu*, *Fri*, *Sat*, *Sun*) se convertirá en las siguientes variables: *endDay_Mon*, *endDay_Tue*, *endDay_Wed*, *endDay_Thu*, *endDay_Fri*, *endDay_Sat* y *endDay_Sun*, como factores de valores 0 y 1.

Nota: Las variables catégoricas con un caracter / son tratadas para suprimir dicho caracter, ya que en ciertos modelos se consideran nombres no permitidos de variables.

```
# Create dummy variables for categorical variables
bd_eBay <- dummy_cols(bd_eBay)

# Remove categorical variables
bd_eBay[, "Category"] <- NULL
bd_eBay[, "currency"] <- NULL
bd_eBay[, "endDay"] <- NULL

# Get column names and convert to factors
cols <- colnames(bd_eBay[,6:33])
bd_eBay[cols] <- lapply(bd_eBay[cols], factor)
```

```r
# Get column names and erase '/' character
cols <- colnames(bd_eBay)
for(i in 1:length(cols)) cols[i] <- gsub("/", "", cols[i])

# Set column names
names(bd_eBay) <- cols
```

Una de las técnicas aprendidas previamente es la de selección de características, por lo que para ello se utilizan diversos métodos con el fin de detectar la importancia de las diferentes variables a tratar:

- Boruta, utilizando para ello una selección de variables significativas, teniendo en cuenta tentativas y confirmar si las variables deben permanecer en el modelo o pueden ser eliminadas, y recibiendo información sobre su importancia.
- Entrenar un modelo lineal y observar que variables son necesarias para construir dicho modelo.

Una vez analizados ambos resultados, se realizará un análisis y se decidirá que variables se han de eliminar.

Para el modelo de **Boruta** se ha utilizado las funciones Boruta, getSelectedAttributes, TentativeRoughFix y attStats del paquete Boruta.

Para ello entrenamos un modelo con **Boruta**, obteniendo los atributos seleccionados y realizando un arreglo para obtener las tentativas (si las hay). A continuación obtendremos de nuevos los atributos y mostramos finalmente la información obtenida.

```r
# Perform Boruta search
boruta_output <- Boruta(`Competitive?`~., data = bd_eBay, doTrace=0)

# Get significant variables including tentatives
boruta_signif <- getSelectedAttributes(boruta_output, withTentative = TRUE)

# Do a tentative rough fix
roughFixMod <- TentativeRoughFix(boruta_output)
boruta_signif <- getSelectedAttributes(roughFixMod)

# Variable Importance Scores
imps <- attStats(roughFixMod)
imps2 = imps[imps$decision != 'Rejected', c('meanImp', 'decision')]

# Print variable importance
print(imps2[order(-imps2$meanImp), ])
```

Tras realizar el análisis con **Boruta**, se procede a realizar el **Modelo lineal**. Para ello se ha utilizado la función stepAIC, la cual nos permite hacer distintos tipos de regresiones de características, pero en nuestro caso indicamos que realice de tipo *both* (*backward* y *forward*). Para ello primero entrenamos el modelo con un modelo lineal utilizando la función lm.

Una vez construido el modelo, se examinan las variables que se han utilizado para construir el modelo y cuales se pueden descartar para analizarlas junto al modelo de **Boruta**:

```r
# Train the linear model
model <- lm(bd_eBay$`Competitive?`~., data = bd_eBay)
# Get the stepwise regression model
step.model <- stepAIC(model, direction = "both", trace = FALSE)
```

```r
# Get the model
anova <- step.model$anova
anova
```

```
## Stepwise Model Path
## Analysis of Deviance Table
##
## Initial Model:
## bd_eBay$`Competitive?` ~ sellerRating + Duration + ClosePrice +
##     OpenPrice + Category_AntiqueArtCraft + Category_Automotive +
##     Category_Books + Category_BusinessIndustrial + Category_ClothingAccessories +
##     Category_CoinsStamps + Category_Collectibles + Category_Computer +
##     Category_Electronics + Category_EverythingElse + Category_HealthBeauty +
##     Category_HomeGarden + Category_Jewelry + Category_MusicMovieGame +
##     Category_Photography + Category_PotteryGlass + Category_SportingGoods +
##     Category_ToysHobbies + currency_EUR + currency_GBP + currency_US +
##     endDay_Fri + endDay_Mon + endDay_Sat + endDay_Sun + endDay_Thu +
##     endDay_Tue + endDay_Wed
##
## Final Model:
## bd_eBay$`Competitive?` ~ sellerRating + Duration + ClosePrice +
##     OpenPrice + Category_Automotive + Category_Books + Category_ClothingAccessories +
##     Category_CoinsStamps + Category_Electronics + Category_EverythingElse +
##     Category_HealthBeauty + Category_Jewelry + Category_MusicMovieGame +
##     Category_PotteryGlass + Category_SportingGoods + currency_GBP +
##     endDay_Mon + endDay_Sat + endDay_Thu + endDay_Tue
##
##
##                             Step Df    Deviance Resid. Df Resid. Dev       AIC
## 1                                                   1942   405.9643 -3056.822
## 2                   - endDay_Wed  0 0.000000000      1942   405.9643 -3056.822
## 3                 - currency_US  0 0.000000000      1942   405.9643 -3056.822
## 4         - Category_ToysHobbies  0 0.000000000      1942   405.9643 -3056.822
## 5            - Category_Computer  1 0.002072021      1943   405.9664 -3058.812
## 6        - Category_Collectibles  1 0.002929466      1944   405.9693 -3060.797
## 7         - Category_Photography  1 0.013141740      1945   405.9824 -3062.733
## 8                   - endDay_Sun  1 0.013184557      1946   405.9956 -3064.669
## 9                 - currency_EUR  1 0.048988336      1947   406.0446 -3066.432
## 10         - Category_HomeGarden  1 0.080918520      1948   406.1255 -3068.039
## 11 - Category_BusinessIndustrial  1 0.092452047      1949   406.2180 -3069.590
## 12                  - endDay_Fri  1 0.089621837      1950   406.3076 -3071.155
## 13    - Category_AntiqueArtCraft  1 0.166380119      1951   406.4740 -3072.347
```

Tras analizar los modelos, y las variables seleccionadas para su revisión, se decide eliminar las siguientes variables del conjunto de datos:

```r
bd_eBay[, "endDay_Wed"] <- NULL
bd_eBay[, "endDay_Fri"] <- NULL
bd_eBay[, "endDay_Sun"] <- NULL
bd_eBay[, "currency_US"] <- NULL
bd_eBay[, "currency_EUR"] <- NULL
bd_eBay[, "Category_ToysHobbies"] <- NULL
bd_eBay[, "Category_Computer"] <- NULL
```

```
bd_eBay[, "Category_Collectibles"] <- NULL
bd_eBay[, "Category_Photography"] <- NULL
bd_eBay[, "Category_HomeGarden"] <- NULL
bd_eBay[, "Category_BusinessIndustrial"] <- NULL
bd_eBay[, "Category_AntiqueArtCraft"] <- NULL
bd_eBay[, "Category_EverythingElse"] <- NULL
```

Creación de grupos de entrenamiento y test

Para comprobar que el modelo de árbol de decisión se realiza una validación cruzada, por lo que se necesitan dos conjuntos: *train* y *test*. El conjunto de entrenamiento comprenderá el 80% de los datos tomados para entrenar el árbol de decisión que clasifique el problema, y el conjunto de testeo se empleará para validad el modelo de ajuste obtenido.

```
create_train_test <- function(data, size = 0.8, train = TRUE) {
  n_row = nrow(data)
  total_row = size * n_row
  train_sample <- 1: total_row
  if (train == TRUE) {
    return (data[train_sample, ])
  } else {
    return (data[-train_sample, ])
  }
}
```

Modelos de clasificación

Para la realización de la práctica se ha decidido tomar diferentes modelos de clasificación y ver como estos son capaces de adaptarse a las condiciones del problema. Los modelos escogidos son:

- Árboles de decisión, con la función rpart.
- Random Forest, con la función randomForest.
- Red Neuronal Artificial, con nnet.
- Naive Bayes, con naiveBayes.
- K-Nearest Neighbour, con knn.

Tras observar los resultados obtenidos, se estableceran una serie de conclusiones y se decidirá sobre los modelos obtenidos cuales son las mejores decisiones que podría tomar *eBay* de cara a sus posibles mejores. Todos estos modelos a su vez serán comprobados mediante matrices de confusión con la función confusionMatrix de la librería caret y mediante curvas ROC mediante la función evalmod de la librería prerec.

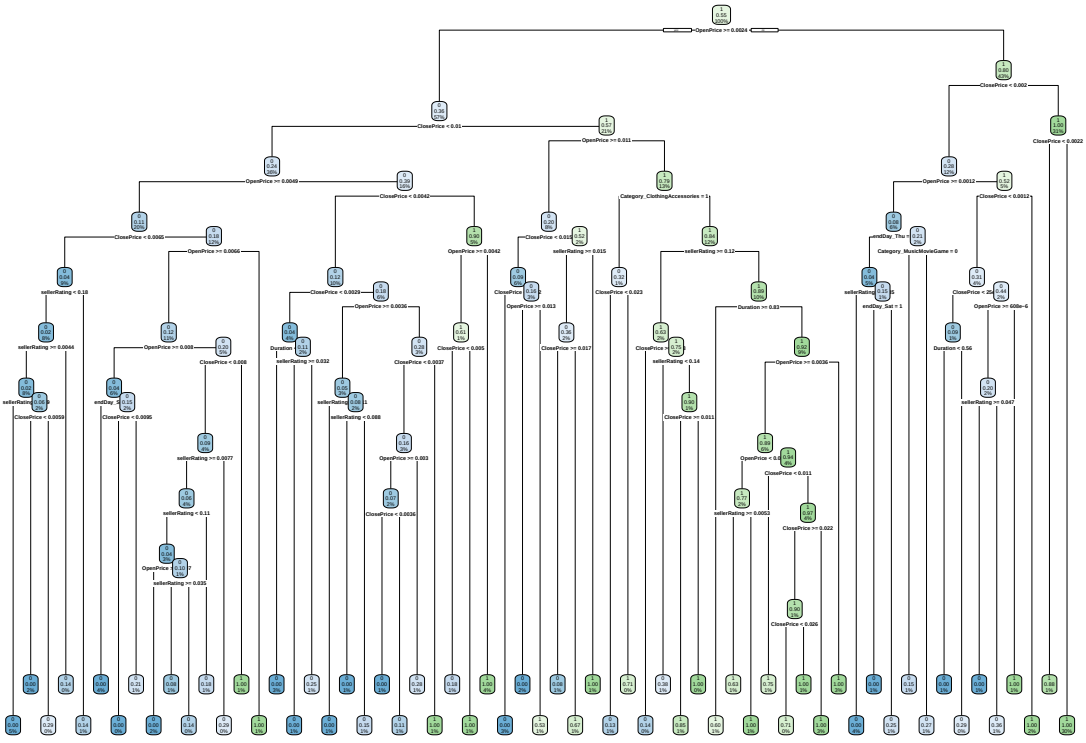
Árboles de decisión

El primer modelo de clasificación escogido es el árbol de decisión binario, el cual dado un conjunto de datos, divide este por características de forma que se tome una decisión o su opuesta, para clasificar. Se ha definido una función que solicita un conjunto de entrenamiento y realiza la clasificación en un árbol de decisión binario. Para ello se ha utilizado la librería rpart, concretamente haciendo uso tanto de la función rpart como de su representación mediante rpart.plot, perteneciente a la librería rpart.plot.

Además, se indica el parémtro `cp=-1` para que se explore el árbol entero, que aunque no se represente entero en el árbol dibujado, el modelo tendrá un mayor ajuste.

```
# Set train and test data
data_train <- create_train_test(bd_eBay, 0.8, train = TRUE)
data_test <- create_train_test(bd_eBay, 0.8, train = FALSE)

# Adjust the model
fit <- rpart(`Competitive?`~., data = data_train, method = 'class', cp=-1)

# Plot the model
rpart.plot(fit)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



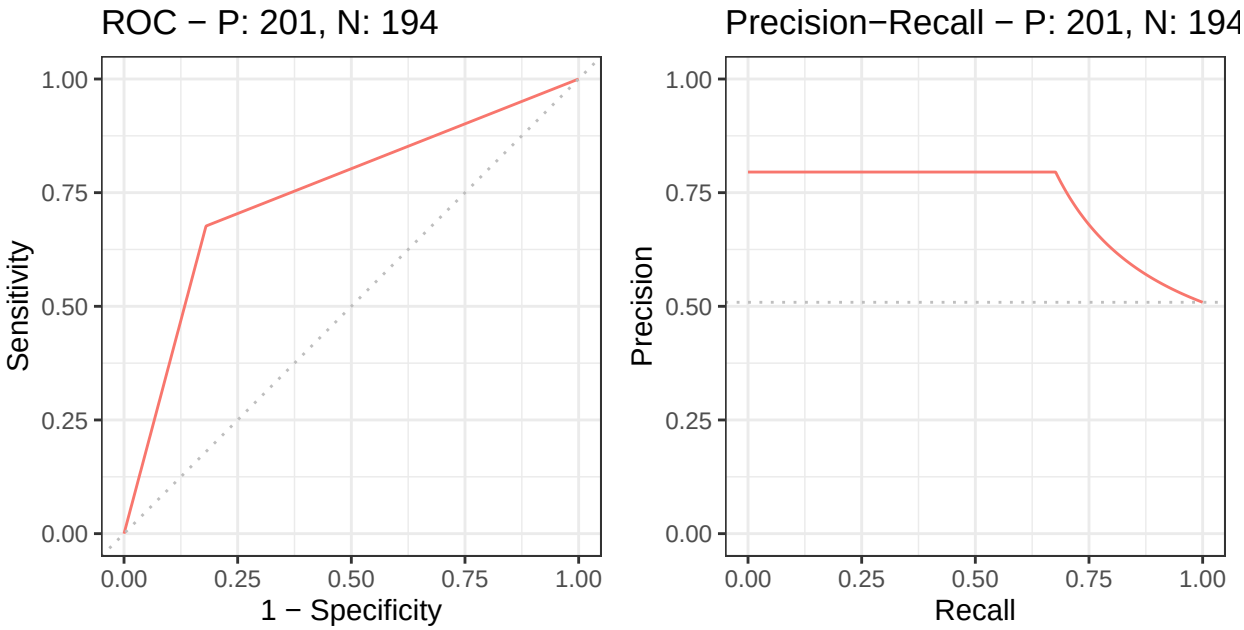
Una vez realizado el modelo de árbol de decisión, se procede a estimar el grupo de test, crear la matrix de confusión y ver el *accuracy* obtenido:

```
# Predict the test data class
predicted_class <-predict(fit, data_test, type='class')

# Create the confusion matrix
matrix<-confusionMatrix(as.factor(data_test$`Competitive?`), predicted_class)
dt_accuracy <- matrix$overall["Accuracy"]

# Print confusion matrix
matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 159  35
##          1  65 136
##
##                Accuracy : 0.7468
##                  95% CI : (0.7009, 0.789)
##     No Information Rate : 0.5671
##     P-Value [Acc > NIR] : 8.699e-14
##
##                   Kappa : 0.4949
##
##  Mcnemar's Test P-Value : 0.003732
##
##             Sensitivity : 0.7098
##             Specificity : 0.7953
##          Pos Pred Value : 0.8196
##          Neg Pred Value : 0.6766
##              Prevalence : 0.5671
##          Detection Rate : 0.4025
##    Detection Prevalence : 0.4911
##       Balanced Accuracy : 0.7526
##
##        'Positive' Class : 0
##
```

A continuación se visualiza la curva ROC asociada al modelo de árbol de decisión:

```
# We evaluate the ROC curve
roc_curve <- evalmod(scores = as.numeric(predicted_class), labels = data_test$`Competitive?`)
autoplot(roc_curve)
```



Random Forest

El siguiente modelo a observar se trata de Random Forest, el cual dado un conjunto de datos, divide este por características de forma que se tome una decisión o su opuesta, para clasificar, formando varios árboles de decisión con decisiones diferentes. Tras crear una seria de árboles de decisión, el algoritmo se encarga de realizar una votación entre los distintos árboles para obtener un modelo final. Para ello se ha utilizado la librería randomForest, utilizando para ello la función randomForest.

Para la realización del modelo se escoge un total de 100 árboles de decisión (*ntree*), y se escoge 2 como número de variables seleccionadas aleatoriamente para la división (*mtry*), tras estudiar y probar diferentes configuraciones del modelo.

Una vez realizado el modelo de Random Forest, se procede a estimar el grupo de test, crear la matrix de confusión y ver el *accuracy* obtenido:

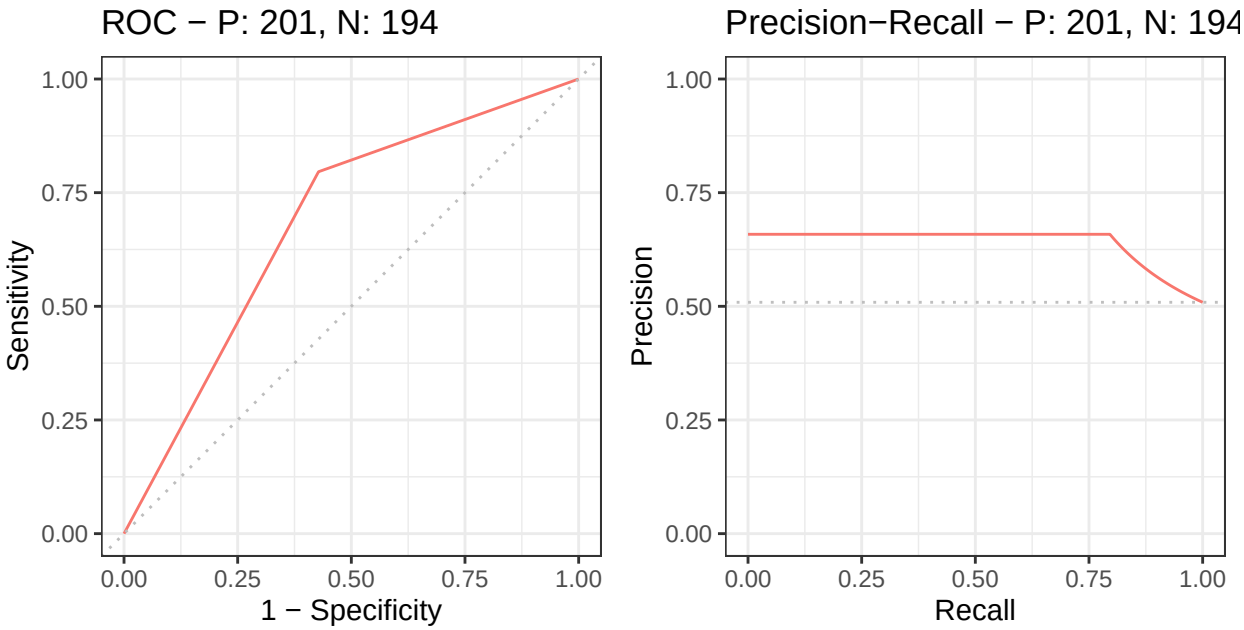
```
# Set train and test data
data_train <- create_train_test(bd_eBay, 0.8, train = TRUE)
data_test <- create_train_test(bd_eBay, 0.8, train = FALSE)


# Adjust the model
fit <- randomForest(as.factor(`Competitive?`)~., data = data_train, ntree=100, mtry=2)

# Predict the test data class
predicted_class <-predict(fit, data_test)
```

```r
# Create the confusion matrix
matrix<-confusionMatrix(as.factor(data_test$`Competitive?`), predicted_class)
rf_accuracy <- matrix$overall["Accuracy"]

# Print confusion matrix
matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 111  83
##          1  41 160
##
##                Accuracy : 0.6861
##                  95% CI : (0.6378, 0.7316)
##     No Information Rate : 0.6152
##     P-Value [Acc > NIR] : 0.0020042
##
##                   Kappa : 0.3696
##
##  Mcnemar's Test P-Value : 0.0002315
##
##             Sensitivity : 0.7303
##             Specificity : 0.6584
##          Pos Pred Value : 0.5722
##          Neg Pred Value : 0.7960
##              Prevalence : 0.3848
##          Detection Rate : 0.2810
##    Detection Prevalence : 0.4911
##       Balanced Accuracy : 0.6943
##
##        'Positive' Class : 0
##
```

A continuación se visualiza la curva ROC asociada al modelo de Random Forest:

```r
# We evaluate the ROC curve
roc_curve <- evalmod(scores = as.numeric(predicted_class), labels = data_test$`Competitive?`)
autoplot(roc_curve)
```



Redes Neuronales

El tercer modelo seleccionado es el de Red Neuronal Articifial, el cual recibe un en las neuronas de entrada un conjunto de datos, que son procesados por las diferentes capas de neuronas ocultas (previamente configuradas) de la red, para obtener en la salida un modelo que a través del aprendizaje puede ajustarse mediante pesos al conjunto de datos.

Para ello se ha utilizado la librería nnet, empleando la función nnet.

Para la realización del modelo se han utilizado una serie de configuraciones distintas, y tras estudiar y analizar los resultados, y analizar el conjunto de datos, se han establecido los siguientes parámetros:

- Número de capas ocultas: 2 (*size*).
- Número máximo de iteraciones: 200 (*maxit*).
- Permitir conexiones entre la entrada y la salida mediante el parámetro *skip*.

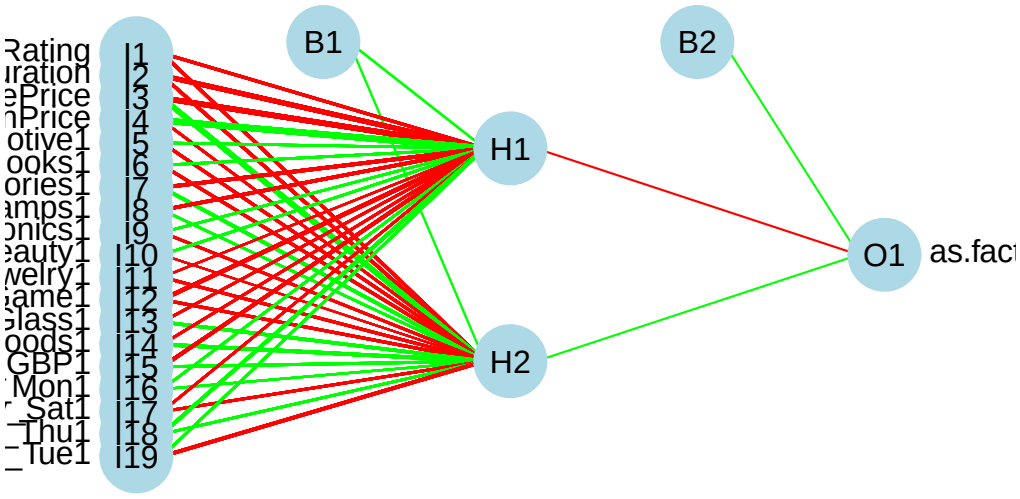
Se han utilizado estos parámetros ya que un ajuste excesivo del modelo es incapaz de predecir correctamente el conjunto de datos y añade una complejidad excesiva, por lo que con una configuración de 2 capas de neuronas ocultas se pueden obtener unos buenos resultados, además de que el modelo llega a converger a las 1000 iteraciones, pero tras estudiarse, con unas 200 iteraciones es más que suficiente para obtener un resultado similar reduciendo en una quinta parte el número de iteraciones del algoritmo.

Para visualizar la red neuronal se utiliza la función plotnet del paquete NeuralNetTools.

```r
# Set train and test data
data_train <- create_train_test(bd_eBay, 0.8, train = TRUE)
data_test <- create_train_test(bd_eBay, 0.8, train = FALSE)

# Adjust the model
nn=nnet(as.factor(`Competitive?`)~ ., data=data_train, size=2,  maxit=200, skip=TRUE)
```

```
## # weights:  62
## initial  value 1177.741645
## iter  10 value 975.086533
## iter  20 value 859.216350
## iter  30 value 788.494142
## iter  40 value 766.016862
## iter  50 value 710.814006
## iter  60 value 680.192126
## iter  70 value 675.952244
## iter  80 value 675.334649
## iter  90 value 673.647206
## iter 100 value 672.259903
## iter 110 value 671.509181
## iter 120 value 666.997113
## iter 130 value 665.926784
## iter 140 value 664.444309
## iter 150 value 663.703657
## iter 160 value 662.905381
## iter 170 value 662.351005
## iter 180 value 661.601985
## iter 190 value 661.301263
## iter 200 value 660.123848
## final  value 660.123848
## stopped after 200 iterations
```

```r
# Plot the model
plotnet(nn, pos_col="green", neg_col="red", max_sp=TRUE)
```



Una vez realizado el modelo de Red Neuronal Artificial, se procede a estimar el grupo de test, crear la matrix de confusión y ver el *accuracy* obtenido:

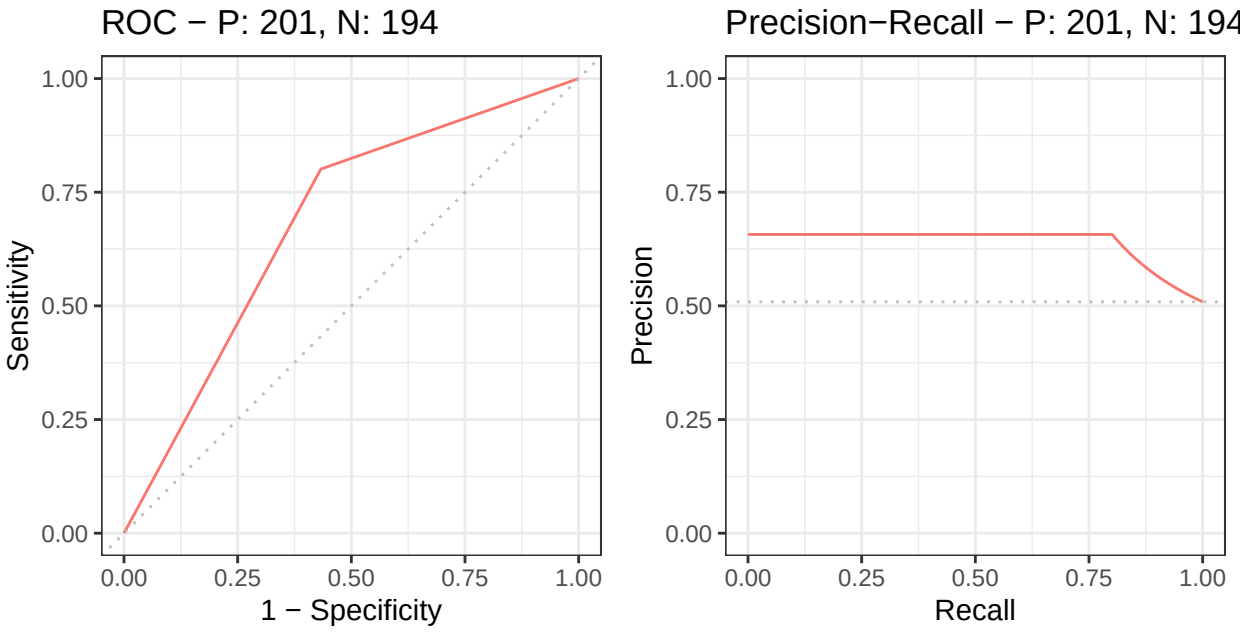
```
# Predict the test data class
predicted_class <-predict(fit, data_test)

# Create the confusion matrix
matrix<-confusionMatrix(as.factor(data_test$`Competitive?`), predicted_class)
nn_accuracy <- matrix$overall["Accuracy"]

# Print confusion matrix
matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 110  84
##          1  40 161
##
##                Accuracy : 0.6861
##                  95% CI : (0.6378, 0.7316)
##     No Information Rate : 0.6203
##     P-Value [Acc > NIR] : 0.0037591
##
##                   Kappa : 0.3695
```

```
##
##  Mcnemar's Test P-Value : 0.0001127
##
##             Sensitivity : 0.7333
##             Specificity : 0.6571
##          Pos Pred Value : 0.5670
##          Neg Pred Value : 0.8010
##              Prevalence : 0.3797
##          Detection Rate : 0.2785
##    Detection Prevalence : 0.4911
##       Balanced Accuracy : 0.6952
##
##        'Positive' Class : 0
##
```

A continuación se visualiza la curva ROC asociada al modelo de Red Neuronal Artificial:

```
# We evaluate the ROC curve
roc_curve <- evalmod(scores = as.numeric(predicted_class), labels = data_test$`Competitive?`)
autoplot(roc_curve)
```



Naive Bayes

El cuarto modelo seleccionado es el de Naive Bayes, el cual asume que la presencia de una característica no es dependiente de la existencia de otra, permitiendo un entrenamiento del modelo mediante las frecuencias

relativas de las características del conjunto de entrenamiento. Para ello crea un modelo que mezcla probabilidad y frecuencia, definiendo así las probabilidades de que se de una determinada clase a partir de sus características de forma independiente.

Para ello se ha utilizado la librería e1071, utilizando la función naiveBayes.

A continuación se pueden observar las diferentes probabilidades individuales de cada característica:

```
# Set train and test data
data_train <- create_train_test(bd_eBay, 0.8, train = TRUE)
data_test <- create_train_test(bd_eBay, 0.8, train = FALSE)

# Adjust the model
fit=naiveBayes(as.factor(`Competitive?`)~ ., data=data_train)

# Print the model
fit
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##         0         1
## 0.4514902 0.5485098
##
## Conditional probabilities:
##    sellerRating
## Y         [,1]      [,2]
##   0 0.10154848 0.1546383
##   1 0.09155428 0.1657191
##
##    Duration
## Y        [,1]      [,2]
##   0 0.6540262 0.2209600
##   1 0.6186256 0.2256137
##
##    ClosePrice
## Y          [,1]       [,2]
##   0 0.007186205 0.00648029
##   1 0.028960695 0.06801027
##
##    OpenPrice
## Y          [,1]        [,2]
##   0 0.006062421 0.004029300
##   1 0.003107341 0.003408884
##
##    Category_Automotive
## Y            0          1
##   0 0.95505618 0.04494382
##   1 0.96878613 0.03121387
##
```

```
##    Category_Books
## Y            0          1
##   0 0.96207865 0.03792135
##   1 0.97572254 0.02427746
##
##    Category_ClothingAccessories
## Y            0          1
##   0 0.93258427 0.06741573
##   1 0.94913295 0.05086705
##
##    Category_CoinsStamps
## Y            0          1
##   0 0.97752809 0.02247191
##   1 0.98843931 0.01156069
##
##    Category_Electronics
## Y             0           1
##   0 0.991573034 0.008426966
##   1 0.965317919 0.034682081
##
##    Category_HealthBeauty
## Y            0          1
##   0 0.93258427 0.06741573
##   1 0.98728324 0.01271676
##
##    Category_Jewelry
## Y            0          1
##   0 0.94522472 0.05477528
##   1 0.96878613 0.03121387
##
##    Category_MusicMovieGame
## Y           0         1
##   0 0.7879213 0.2120787
##   1 0.7387283 0.2612717
##
##    Category_PotteryGlass
## Y             0           1
##   0 0.991573034 0.008426966
##   1 0.994219653 0.005780347
##
##    Category_SportingGoods
## Y            0          1
##   0 0.95926966 0.04073034
##   1 0.91676301 0.08323699
##
##    currency_GBP
## Y            0          1
##   0 0.93679775 0.06320225
##   1 0.88901734 0.11098266
##
##    endDay_Mon
## Y           0         1
##   0 0.7963483 0.2036517
##   1 0.6369942 0.3630058
```

```
##
##    endDay_Sat
## Y           0         1
##   0 0.7724719 0.2275281
##   1 0.8624277 0.1375723
##
##    endDay_Thu
## Y           0         1
##   0 0.8932584 0.1067416
##   1 0.8647399 0.1352601
##
##    endDay_Tue
## Y           0          1
##   0 0.91994382 0.08005618
##   1 0.91560694 0.08439306
```

Una vez realizado el modelo de Naive Bayes, se procede a estimar el grupo de test, crear la matrix de confusión y ver el *accuracy* obtenido:

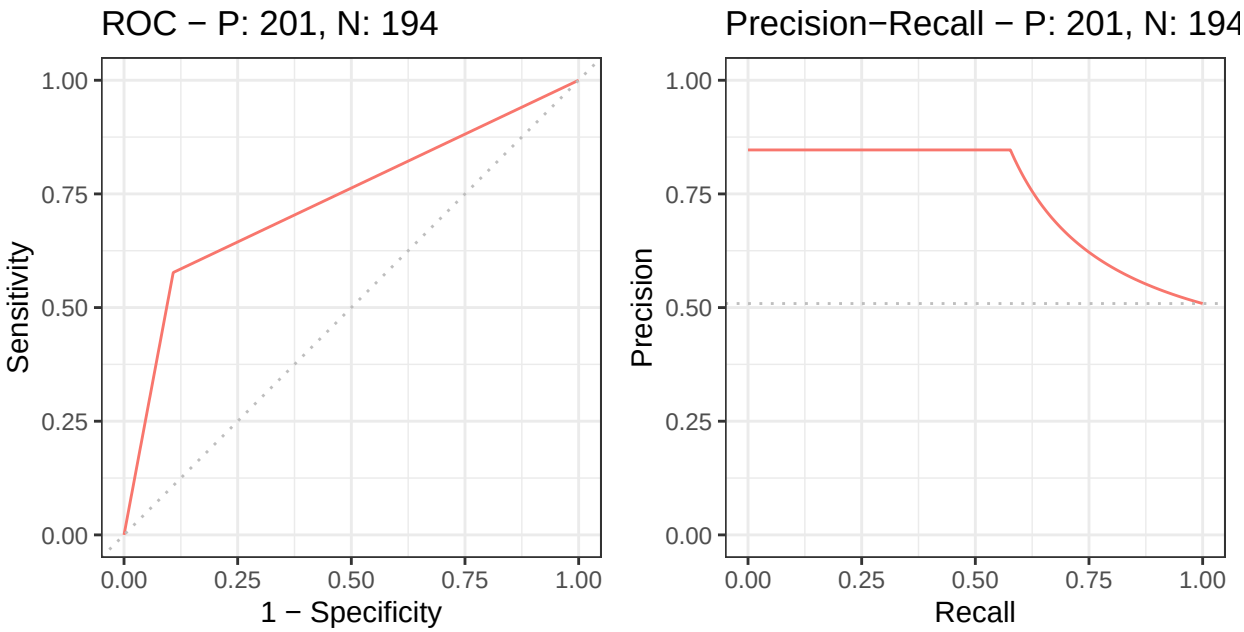
```
# Predict the test data class
predicted_class <-predict(fit, data_test)

# Create the confusion matrix
matrix<-confusionMatrix(as.factor(data_test$`Competitive?`), predicted_class)
nb_accuracy <- matrix$overall["Accuracy"]

# Print confusion matrix
matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 173  21
##          1  85 116
##
##                Accuracy : 0.7316
##                  95% CI : (0.6851, 0.7747)
##     No Information Rate : 0.6532
##     P-Value [Acc > NIR] : 0.0005104
##
##                   Kappa : 0.4662
##
##  Mcnemar's Test P-Value : 9.41e-10
##
##             Sensitivity : 0.6705
##             Specificity : 0.8467
##          Pos Pred Value : 0.8918
##          Neg Pred Value : 0.5771
##              Prevalence : 0.6532
##          Detection Rate : 0.4380
##    Detection Prevalence : 0.4911
##       Balanced Accuracy : 0.7586
##
```

```
##         'Positive' Class : 0
##
```

A continuación se visualiza la curva ROC asociada al modelo de Naive Bayes:

```
# We evaluate the ROC curve
roc_curve <- evalmod(scores = as.numeric(predicted_class), labels = data_test$`Competitive?`)
autoplot(roc_curve)
```



KNN

El último modelo seleccionado es el de K-Nearest Neighbors, el cual dado un conjunto de entrada con diferentes clases, analiza si un elemento posee dentro de su rango más vecinos de una clase u otra, y clasifica en aquella clase en la que posea más vecinos dentro de un rango con k vecinos. Este modelo sigue una distribución espacial de los elementos.

Para la realización de este modelo se ha hecho uso de la librería class, concretamente de la función knn.

Tras probar diferentes parámetros, y observar el tamaño del conjunto de datos, se ha tomado como valor `k=20`, es decir, los 20 vecinos más cercanos.

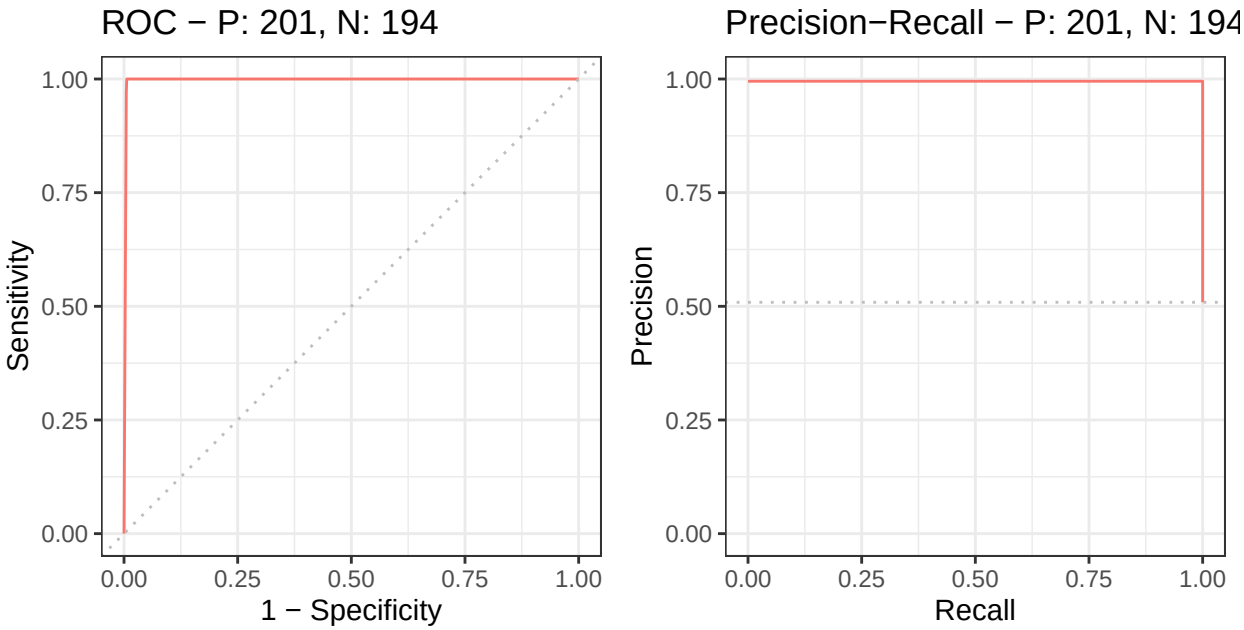
Una vez realizado el modelo de KNN, se procede a estimar el grupo de test, crear la matrix de confusión y ver el *accuracy* obtenido:

```r
# Extract train and test labels
train.label <- data_train[, "Competitive?"]
test.label <- data_test[, "Competitive?"]

# Adjust the model
predicted_class <- knn(data_train,data_test,cl=train.label,k=20)

# Create the confusion matrix
matrix<-confusionMatrix(as.factor(test.label), predicted_class)
knn_accuracy <- matrix$overall["Accuracy"]

# Print confusion matrix
matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 193   1
##          1   0 201
##
##                Accuracy : 0.9975
##                  95% CI : (0.986, 0.9999)
##     No Information Rate : 0.5114
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9949
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 1.0000
##             Specificity : 0.9950
##          Pos Pred Value : 0.9948
##          Neg Pred Value : 1.0000
##              Prevalence : 0.4886
##          Detection Rate : 0.4886
##    Detection Prevalence : 0.4911
##       Balanced Accuracy : 0.9975
##
##        'Positive' Class : 0
##
```

A continuación se visualiza la curva ROC asociada al modelo de KNN:

```r
# We evaluate the ROC curve
roc_curve <- evalmod(scores = as.numeric(predicted_class), labels = test.label)
autoplot(roc_curve)
```



Comparativa Accuracy

Tras realizar los diferentes modelos, se procede a visualizar una tabla comparativa del *accuracy* obtenido por los diferentes modelos:

```
models_name <- c('Árbol de decisión', 'Random Forest', 'Red Neuronal Artificial', 'Naive Bayes', 'KNN')
models_acc <- c(dt_accuracy, rf_accuracy, nn_accuracy, nb_accuracy, knn_accuracy)
acc_table <- data.frame(models_name, models_acc)
names(acc_table) <- c("Modelos", "Accuracy")

acc_table
```

```
##                   Modelos  Accuracy
## 1       Árbol de decisión 0.7468354
## 2           Random Forest 0.6860759
## 3 Red Neuronal Artificial 0.6860759
## 4             Naive Bayes 0.7316456
## 5                     KNN 0.9974684
```

Por último, antes de presentar las conclusiones, se observa un resumen de la información final del conjunto tras los diferentes tratamientos para ambas clases:

```r
# Get both classes from the dataset
for(i in 1:length(bd_eBay)) non_competitive_class <- bd_eBay[bd_eBay$`Competitive?` == 0, ]
for(i in 1:length(bd_eBay)) competitive_class <- bd_eBay[bd_eBay$`Competitive?` == 1, ]

summary(non_competitive_class)
```

```
##   sellerRating          Duration        ClosePrice         OpenPrice
##  Min.   :0.0001325   Min.   :0.0000   Min.   :0.000000   Min.   :0.000000
##  1st Qu.:0.0222917   1st Qu.:0.4444   1st Qu.:0.003594   1st Qu.:0.003594
##  Median :0.0574655   Median :0.6667   Median :0.007378   Median :0.006987
##  Mean   :0.1001984   Mean   :0.6161   Mean   :0.019139   Mean   :0.017006
##  3rd Qu.:0.1163623   3rd Qu.:0.6667   3rd Qu.:0.018008   3rd Qu.:0.012302
##  Max.   :1.0000000   Max.   :1.0000   Max.   :1.000000   Max.   :1.000000
##   Competitive? Category_Automotive Category_Books Category_ClothingAccessories
##  Min.   :0     0:791               0:879          0:847
##  1st Qu.:0     1:115               1: 27          1: 59
##  Median :0
##  Mean   :0
##  3rd Qu.:0
##  Max.   :0
##  Category_CoinsStamps Category_Electronics Category_HealthBeauty
##  0:880                0:895                0:853
##  1: 26                1: 11                1: 53
##
##
##
##
##  Category_Jewelry Category_MusicMovieGame Category_PotteryGlass
##  0:854            0:746                   0:893
##  1: 52            1:160                   1: 13
##
##
##
##
##  Category_SportingGoods currency_GBP endDay_Mon endDay_Sat endDay_Thu
##  0:872                  0:860        0:727      0:705      0:826
##  1: 34                  1: 46        1:179      1:201      1: 80
##
##
##
##
##  endDay_Tue
##  0:826
##  1: 80
##
##
##
##
```

```r
summary(competitive_class)
```

```
##   sellerRating        Duration        ClosePrice         OpenPrice
```

```
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.000000   Min.   :0.000000
##  1st Qu.:0.01026   1st Qu.:0.4444   1st Qu.:0.006146   1st Qu.:0.000991
##  Median :0.04102   Median :0.6667   Median :0.014505   Median :0.002440
##  Mean   :0.08941   Mean   :0.6040   Mean   :0.051211   Mean   :0.009464
##  3rd Qu.:0.08612   3rd Qu.:0.6667   3rd Qu.:0.042498   3rd Qu.:0.008999
##  Max.   :1.00000   Max.   :1.0000   Max.   :0.971972   Max.   :0.650647
##   Competitive? Category_Automotive Category_Books Category_ClothingAccessories
##  Min.   :1     0:1003              0:1039         0:1006
##  1st Qu.:1     1:  63              1:  27         1:  60
##  Median :1
##  Mean   :1
##  3rd Qu.:1
##  Max.   :1
##  Category_CoinsStamps Category_Electronics Category_HealthBeauty
##  0:1055               0:1022               0:1055
##  1:  11               1:  44               1:  11
##
##
##
##
##  Category_Jewelry Category_MusicMovieGame Category_PotteryGlass
##  0:1036           0:823                   0:1059
##  1:  30           1:243                   1:   7
##
##
##
##
##  Category_SportingGoods currency_GBP endDay_Mon endDay_Sat endDay_Thu
##  0:976                  0:965        0:697      0:916      0:944
##  1: 90                  1:101        1:369      1:150      1:122
##
##
##
##
##  endDay_Tue
##  0:975
##  1: 91
##
##
##
##
```

Conclusiones

Tras realizar el preprocesamiento, análisis del conjunto de datos, clasificación en distintos modelos, comparativa de modelos y análisis de resultados, se han llegado a las siguientes conclusiones:

- Uno de los puntos complicados que se trató en la realización de la práctica fue el de selección de características, ya que al tratarse de variables categóricas, muchos modelos no podían realizarse o perdían bastante efectividad debido a estas variables, aunque se planteó una discretización, tras realizar estudio sobre diferentes opciones como normalización entre otras, se decidió utilizar una conversión en **Dummy variables** porque se considera que es la mejor forma de que la información que se representa siga manteniendo su significado sin que este se vea modificado. Tras este procedimiento se han realizado

diversos modelos para elegir que características aportaban con mayor o menor importancia al modelo, y se ha decidido eliminar aquellas que si bien influyen, lo hacen en una medida ínfima, y generan más desconocimiento de lo que aportan realmente al problema.

- Otra de las tareas más complicadas ha sido la realización de una selección de instancias, ya que en este problema una selección aleatoria no tiene sentido, ya que se pueden no considerar determinados casos de determinadas categorías de los productos o días de la semana. Se han realizado otros modelos y otros procedimientos de selección de instancias como FRIS, pero en este caso, aunque se obtenían instancias con mucha representatividad, se reducía demasiado el conjunto de datos, como para considerarse una selección representativa de todo el conjunto o fiable del mismo. Se obtenían mejores resultados, pero al extrapolar estos resultados al resto del conjunto inicial, se obtenía un empeoramiento, por lo que se ha descartado esta técnica para este problema.

- Tras analizar los distintos modelos, se puede ver que realmente es un problema complejo de decidir realmente que características son las más importantes, aunque se puede observar que en los diferentes modelos se hace notable que las variables **closePrice**, **openPrice** y **sellerRating** son las más representativas del conjunto, y realizando un análisis del conjunto de datos, es un planteamiento coherente.

- No es de extrañar que el algoritmo con mejores resultados sea **KNN**, ya que se basa en una distribución espacial de características, por lo que subastas con elementos similares realmente serán clasificadas de un mismo modo. A pesar de que existe una considerable diferencia entre este modelo y los demás, los distintos modelos se han desenvuelto bien obteniendo una serie de resultados aceptables, donde se puede observar que para este problema los modelos que hacen uso de la aleatoriedad y estadística, como son en este caso **Random Forest** y **Red Neuronal Artificial** (recordemos que se ha diseñado una red sencilla sin pesos ajustados, ya que conlleva un mayor desarrollo del problema). Por último los modelos que hacen uso o bien de la probabilidad estadística o de la estadística meramente obtienen unos buenos resultados.

Preguntas y respuestas

- **¿Qué se recomendaría a un vendedor para hacer que sus subastas tengan más probabilidad de ser competitivas?**

Realmente lo ideal sería ver ejemplos dentro de su mismo sector y realizar subastas similares, ya que como se puede observar con el algoritmo **KNN**, subastas con características similares obtendrán resultados similares, por lo que dentro de un margen de adaptación del vendedor, lo ideal sería realizar comparativa de subastas previas.

Por otro lado, se puede observar claramente que las subastas competitivas son aquellas que se cierran con un mayor precio (lo cual es lógico y no aporta información adicional), pero también lo son aquellas que empiezan con un precio de apertura inferior, probablemente por la accesibilidad que estas permiten.

Por último, cabría destacar que los días donde se obtienen mejores resultados son los Lunes, seguido de los fines de semana, y los peores los Martes y Miércoles, por lo que lo ideal es hacer ofertas que acaben en fin de semana, preferiblemente Lunes.

- **En función del conocimiento obtenido, ¿qué estrategias de negocio podría adoptar la empresa eBay para mejorar el resultado de las subastas?**

Tras el análisis se puede observar que por norma general, los mejores vendedores no son los que obtienen los mejores resultados, sino aquellos vendedores que siendo buenos, no son los más destacados, ya que por norma general, un vendedor muy destacado tiene a ofrecer subastas con precios mayores, lo cual las hace menos competitivas. *eBay* como empresa debería potenciar este sector de forma que se alimente la competencia entre los grandes vendedores y vendedores no tan conocidos, obteniendo así una mayor competitivad de precios e igualando los diferentes sectores.

Por otro lado, *eBay* debería favorecer de alguna forma el inicio de las subastas por precios reducidos, de forma que estas sean más llamativas para el cliente, y a su vez estas acaben por norma general en los días donde se realiza un mayor consumo, en el intervalo de días descrito en la anterior pregunta (fin de semana con preferencia en Lunes).