Entrega B: R - Evaluación de Modelos

Andrés García-Serra Romero Ciencia de Datos

## Introducción

En este trabajo aplicaremos los conocimientos aprendidos en evaluación de diferentes tipos de clasificadores de datos en la tarea de predicción de si un indivíduo ha ganado o no una partida de tres en raya, utilizando como datos de input la distribución del tablero de la partida. Para realizar la práctica hemos utilizado el software RStudio, creando un proyecto del tipo Markdown notebook. Utilizando el paquete *caret* de *R* entrenaremos cuatro modelos: Un Naive Bayes, un modelo de Árboles de Decisión, una Red Neuronal, uno de Nearest Neighbours y finalmente un SVM con kernel lineal. Estos modelos serán entenados todos usando Cross-Validation y los mismos sets de entrenamiento y evaluación, divididiendo el set total de datos al 70/30 respectivamente.

Explicaremos en primer lugar cómo hemos llevado a cabo la división del set global de datos manteniendo la proporción de clases y comprobando que todos los datos sean correctos. Seguidamente entrenaremos los modelos, mostrando las métricas de entrenamiento de cada modelo y evaluaremos los modelos usando el set de evaulación, para lo cual calcularemos distintas métricas (incluyendo el AUC) para comprobar la validez de cada uno de los clasificadores según sus curvas ROC. Finalmente, responderemos las preguntas planteadas en el enunciado de la tarea.

## 1. Importar Datos

En primer lugar importaremos el fichero *tic-tac-toe.txt* con los datos en un dataframe, del que cambiaremos los nombres de las columnas para comprender mejor los datos. Para esto tendremos en cuenta de que se trata de un tablero 3x3 en el que las posiciones van desde arriba a la izquierda (*pos1*) hasta debajo a la derecha (*pos9*). Llamaremos *win* al último elemento de cada fila, que recoge la victoria o derrota del jugador "x". Aquí podemos ver un ejemplo para entender mejor la distribución de los datos, siendo un ejemplo aleatorio, no real.

Para el caso real, podemos generar una tabla que recoge las primeras 6 filas de nuestro dataframe ya modificado.

```
tictac <- data.frame(read.csv("tic-tac-toe.txt"))
header <- c("pos1","pos2","pos3","pos4","pos5","pos6","pos7","pos8","pos9","win")
colnames(tictac) <- header
head(tictac)
```

```
##   pos1 pos2 pos3 pos4 pos5 pos6 pos7 pos8 pos9      win
## 1    x    x    x    x    o    o    o    x    o positive
## 2    x    x    x    x    o    o    o    o    x positive
## 3    x    x    x    x    o    o    o    b    b positive
## 4    x    x    x    x    o    o    b    o    b positive
## 5    x    x    x    x    o    o    b    b    o positive
## 6    x    x    x    x    o    b    o    o    b positive
```

Como último paso dentro del input de datos, analizamos en busca de valores vacíos, NaNs o valores que no sean los deseados, es decir: "x", "o" o "b" entre los componentes de posición y "positive" o "negative" en el último elemento de cada fila.

```r
any(is.na(tictac))
values <- c("x", "o", "b","positive", "negative")
any(!as.matrix(tictac[, 1:9]) %in% values[1:3])
any(!tictac[, 10] %in% values[4:5])
```

```
## [1] FALSE
## [1] FALSE
## [1] FALSE
```

Podemos ver que el output de estas tres comprobaciones nos da un valor booleano "FALSE", indicando que no existen valores NaN, vacíos o diferentes a los esperados.

2. Data Splitting

Usando el paquete *caret*, dividimos el dataset en 70% training y 30% test.

```r
set.seed(100)
train_pos <- createDataPartition(tictac$win, p = .7,
                                 list = FALSE,
                                 times = 1)
tictac_train <- tictac[ train_pos,]
tictac_test  <- tictac[-train_pos,]
```

Además, comprobamos dos cosas, la primera es que realmente se hayan dividido correctamente y el cociente entre la cantidad de datos para entrenamiento y validación sea igual o muy cercano a 7/3. La segunda es que la proporción de clases ("positive" y "negative") para los nuevos sets de datos sigan el mismo ratio que en el dataset original.

```r
# Ratio entre el Set de Entrenamiento y el de Evaluación respecto a 7/3
test_train_ratio <- data.frame(
  Name = c("Sets Size Ratio to 7/3"),
  value = dim(tictac_train)[1]/dim(tictac_test)[1] / (7/3)
)
kable(test_train_ratio, format = "html") %>%
  kable_styling(full_width = FALSE, bootstrap_options = c("striped", "hover"))

# Ratios de Positivos/Negativos para cada set
sets <- data.frame(
  Set = c("Pos/Neg Ratio"),
  Full_Dataset = c(sum(tictac[,10]=="positive")/sum(tictac[,10]=="negative")),
  Training_Set = c(sum(tictac_train[,10]=="positive")/sum(tictac_train[,10]=="negative")),
  Validation_Set = c(sum(tictac_test[,10]=="positive")/sum(tictac_test[,10]=="negative"))
)
kable(sets, col.names = c("", "Full Dataset", "Training Set", "Validation Set"), format = "html") %>%
  kable_styling(full_width = FALSE, bootstrap_options = c("striped", "hover"))
```

Name

value

Sets Size Ratio to 7/3

1.005494

Full Dataset

Training Set

Validation Set

Pos/Neg Ratio

1.88253

1.879828

1.888889

Como podemos ver, tanto la proporción de datos entre los datasets como el ratio de clases cumplen nuestras espectativas, siendo esta segunda comprobación casi idéntica para los tres datasets, el global y los dos nuevos de entrenamiento y validación.

3. Generación de Modelos

Utilizando nuestro nuevo dataset de entrenamiento podemos generar los 5 modelos deseados, que tras una búsqueda en la documentación del paquete caret* serán el *'nb'* (Naive Bayes), *'C5.0'* (Decision Tree), *'nnet'* (Neural Network), *'knn'* (Nearest Neighbour) y por último el *'svmLinear2'* (SVM con kernel lineal).

Nos aseguramos que los entrenamientos se llevan a cabo utilizando validación cruzada de 10 folds y que nuestros clasificadores ya entrenados también sean evaluados como soft en este mismo proceso de entrenamiento, para más tarde utilizar esta propiedad para poder representar las curvas ROC y hallar la AUC de cada clasificador.

```
fitControl <- trainControl(
  method = "cv",      # validacion cruzada
  number = 10,        # numero de folds
  classProbs = TRUE
)

set.seed(100)
models <- list(
  nb_train <- train(win ~ ., data = tictac_train, method = "nb", trControl = fitControl),
  dt_train <- train(win ~ ., data = tictac_train, method = "C5.0", trControl = fitControl),
  nn_train <- train(win ~ ., data = tictac_train, method = "nnet", trControl = fitControl),
  knn_train <- train(win ~ ., data = tictac_train, method = "knn", trControl = fitControl),
  SVM_train <- train(win ~ ., data = tictac_train, method = "svmLinear2", trControl = fitControl)
)

results <- bind_rows(lapply(models, function(model) {
  data.frame(
    Accuracy = max(model$results$Accuracy),  # Mejor accuracy
    Kappa = max(model$results$Kappa)         # Mejor kappa
  )
}))

Models <- c('Naive Bayes','Decision Tree','Neural Network','Nearest Neighbour','SVM')
results <- cbind(Models, results)
```

En la siguiente tabla representamos los valores de Accuracy y Kappa del entrenamiento de cada uno de los modelos. Vemos como para el dataset de entrenamiento el modelo que mejores valores de estas métricas da es el Decision Tree, seguido de cerca por el de Red Neuronal y SVM. Estos últimos dos modelos tienen un valor de accuracy idéntico, dando a entender que debido a la simplicidad de los datos y que estamos usando un modelo simple de red neuronal, ambos modelos habrán encontrado una solución muy similar para dividir las dos clases.

Por último queda destacar el mal desempeño del modelo de Naive Bayes, cuyas métricas están muy por debajo del resto de modelos. Esto puede deberse a varias causas, pero la más probable es la correlación de las variables, las cuales el modelo Naive Bayes asume independientes por naturaleza.

Models

Accuracy

Kappa

Naive Bayes

0.6900351

0.3008525

Decision Tree

0.9851844

0.9664239

Neural Network

0.9820669

0.9595686

Nearest Neighbour

0.9343024

0.8482188

SVM

0.9820669

0.9594050

4. Rendimiento de modelos

Una vez tenemos los modelos correctamente entrenados, podemos aplicarlos a nuestro dataset de validación para ver cuál es su desempeño. Para esto utilizaremos la rutina *predict*. En primer lugar, como ejemplo podemos observar la matriz de confusión del modelo de Red Neuronal, en el que podemos ver que acierta todos los valores negativos, que son menoría, fallando únicamente 3 clases que predice como positivas y son realmente negativas. Este es un muy bien desempeño, como también podemos ver en las diferentes métricas.

```
nn_pred <- predict(nn_train,tictac_test)
nn_matrix <- confusionMatrix(nn_pred,factor(tictac_test$win))
nn_eval <- postResample(nn_pred,factor(tictac_test$win))
nn_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction negative positive
##   negative       96        0
##   positive        3      187
##
##                Accuracy : 0.9895
##                  95% CI : (0.9697, 0.9978)
```

```
##     No Information Rate : 0.6538
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9767
##
##  Mcnemar's Test P-Value : 0.2482
##
##             Sensitivity : 0.9697
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9842
##              Prevalence : 0.3462
##          Detection Rate : 0.3357
##    Detection Prevalence : 0.3357
##       Balanced Accuracy : 0.9848
##
##        'Positive' Class : negative
##
```

Podemos mostrar también la matriz de confusión del resto de modelos. Para hacer este proceso más visual nos apoyamos en *gpt-4o* y obtenemos una visual más clara de todas las matrices.

Podemos ver cómo los modelos de Decision Tree Neural Network y SVM funcionan prácticamente idénticos, dejando entre 3 y 4 valores clasificados como positivos pero que realmente eran negativos. Esto pude deberse a 4 outliers que no siguen la tendencia general de los datos, por ello se repiten en los 3 modelos, podrían ser distintas predicciones en cada caso pero se trataría de una coincidencia demasiado oportuna.

También podemos ver cómo el modelo de Naive Bayes predice la gran mayoría de datos (exceptuando 5) en la clase positiva, fallando así todas las clases que realmente eran negativas.

El modelo de Nearest Neighbour sigue la tendencia de estos tres pero con menos porcentaje de aciertos.

Podemos ahora extraer una tabla de accuracy y kappa de cada modelo aplicado a los datos de evaluación y utilizando el

```
## Setting levels: control = negative, case = positive
```

```
## Setting direction: controls < cases
```

```
## Setting levels: control = negative, case = positive
```

```
## Setting direction: controls < cases
```

```
## Setting levels: control = negative, case = positive
```

```
## Setting direction: controls < cases
```

```
## Setting levels: control = negative, case = positive
```

```
## Setting direction: controls < cases
```

```
## Setting levels: control = negative, case = positive
```

```
## Setting direction: controls < cases
```

Models

Accuracy

Kappa

AUC

Naive Bayes

0.6713287

0.0650344

0.7447739

Decision Tree

0.9895105

0.9766607

0.9960568

Neural Network

0.9895105

0.9766607

0.9863339

Nearest Neighbour

0.9580420

0.9055118

0.9975153

SVM

0.9860140

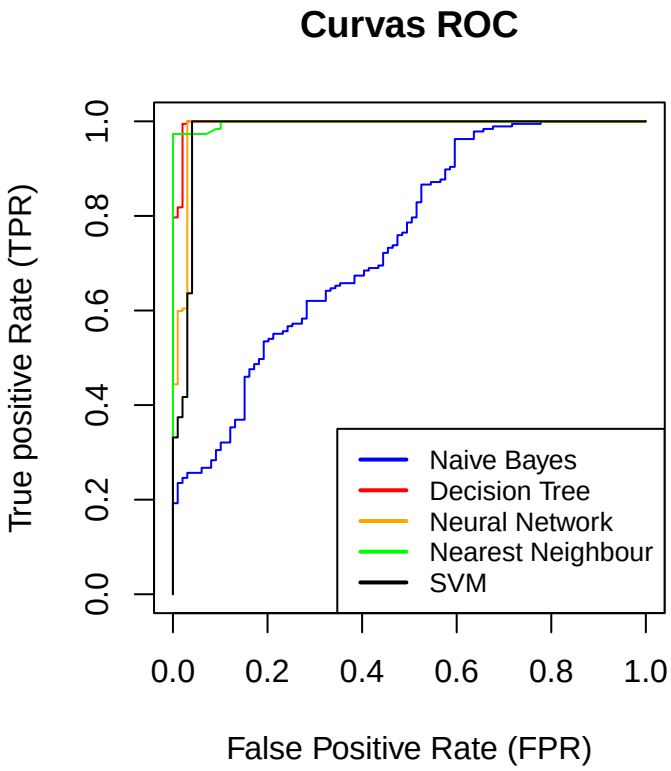
0.9688062

0.9773673

```
set.seed(100)

nb_pred <- prediction(nb_prob[,2], tictac_test$win)
dt_pred <- prediction(dt_prob[,2], tictac_test$win)
nn_pred <- prediction(nn_prob[,2], tictac_test$win)
knn_pred <- prediction(knn_prob[,2], tictac_test$win)
SVM_pred <- prediction(SVM_prob[,2], tictac_test$win)

nb_perf <- performance(nb_pred, measure = "tpr", x.measure = "fpr")
par(pty = "s")
plot(nb_perf, col = "blue", main = "Curvas ROC", xlab = "False Positive Rate (FPR)",
     ylab = "True positive Rate (TPR)", xlim = c(0, 1), ylim = c(0, 1), asp=1)
plot(performance(dt_pred, measure = "tpr", x.measure = "fpr"), col = "red", add = TRUE)
plot(performance(nn_pred, measure = "tpr", x.measure = "fpr"), col = "orange", add = TRUE)
plot(performance(knn_pred, measure = "tpr", x.measure = "fpr"), col = "green", add = TRUE)
plot(performance(SVM_pred, measure = "tpr", x.measure = "fpr"), col = "black", add = TRUE)
legend("bottomright", legend = Models, col = c("blue", "red","orange","green","black"), lwd = 2, cex =
```


Curvas ROC
True positive Rate (TPR)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (FPR)
Naive Bayes
Decision Tree
Neural Network
Nearest Neighbour
SVM