Machine Learning I

KNN y Naives Bayes

Dante Conti, Sergi Ramirez, (c) IDEAI

2025-10-12

Table of contents

0.1 Definición del problema

0.1.1 Contexto

Una tienda está planteando la venta final del año. Queremos lanzar una oferta. Será válido sólo para los clientes existentes y la campaña a través de las llamadas telefónicas que se está planificando actualmente para ellos. La dirección considera que la mejor manera de reducir el coste de la campaña es hacer un modelo predictivo que clasifique a los clientes que puedan comprar la oferta.

Las variables que contiene la base de datos son:

- **Response (target)**: 1 si el cliente aceptó la oferta en la última campaña, 0 en caso contrario
- **ID**: ID único de cada cliente

- **Year_Birth** - Edad del cliente
- **Complain** - 1 si el cliente presentó una queja en los últimos 2 años
- **Dt_Customer** - Fecha de alta del cliente en la empresa
- **Education** - Nivel de estudios del cliente
- **Marital** - Estado civil del cliente
- **Kidhome** - Número de niños pequeños en el hogar del cliente
- **Teenhome** - Número de adolescentes en el hogar del cliente
- **Income** - Ingresos anuales del hogar del cliente
- **MntFishProducts** - Cantidad gastada en productos de pescado en los últimos 2 años
- **MntMeatProducts** - Cantidad gastada en productos cárnicos en los últimos 2 años
- **MntFruits** - Cantidad gastada en frutas en los últimos 2 años
- **MntSweetProducts** - cantidad gastada en productos dulces en los últimos 2 años
- **MntWines** - cantidad gastada en productos de vino en los últimos 2 años
- **MntGoldProds** - cantidad gastada en productos de oro en los últimos 2 años
- **NumDealsPurchases** - número de compras realizadas con descuento
- **NumCatalogPurchases** - número de compras realizadas por catálogo (comprando productos con envío por correo)
- **NumStorePurchases** - número de compras realizadas directamente en tiendas
- **NumWebPurchases** - número de compras realizadas a través del sitio web de la empresa
- **NumWebVisitsMonth** - número de visitas al sitio web de la empresa en el último mes
- **Recency** - número de días desde la última compra

0.1.2 Objetivo

Ls supertienda quiere predecir la probabilidad que el cliente de una respuesta positiva y identificar los diferentes factores que afectan la respuesta del cliente.

Podéis encontrar la base de datos en la siguiente web

0.2 KNN

0.2.1 Distancias

0.2.2 KNN Classifier

0.2.2.1 R Base

```
set.seed(1994)

ind_col <- c(16)

default_idx = sample(nrow(datos), nrow(datos)*0.7)
```

```
train <- datos[default_idx, ]
X_train <- train[, -ind_col]
y_train <- train[, 16]
test <- datos[-default_idx, ]
X_test <- test[, -ind_col]
y_test <- test[, 16]

X_train <- data.frame(lapply(X_train, as.numeric))
X_test <- data.frame(lapply(X_test, as.numeric))
```

```
prediccion <- knn(train = X_train, test  = X_test,
                  cl = y_train, k = 3)
head(prediccion)
```

```
[1] No  Yes No  No  No  No
Levels: No Yes
```

```
calc_class_err = function(actual, predicted) {
  mean(actual != predicted)
}
```

```
calc_class_err(actual    = y_test,
               predicted = knn(train = X_train,
                               test  = X_test,
                               cl    = y_train,
                               k     = 5))
```

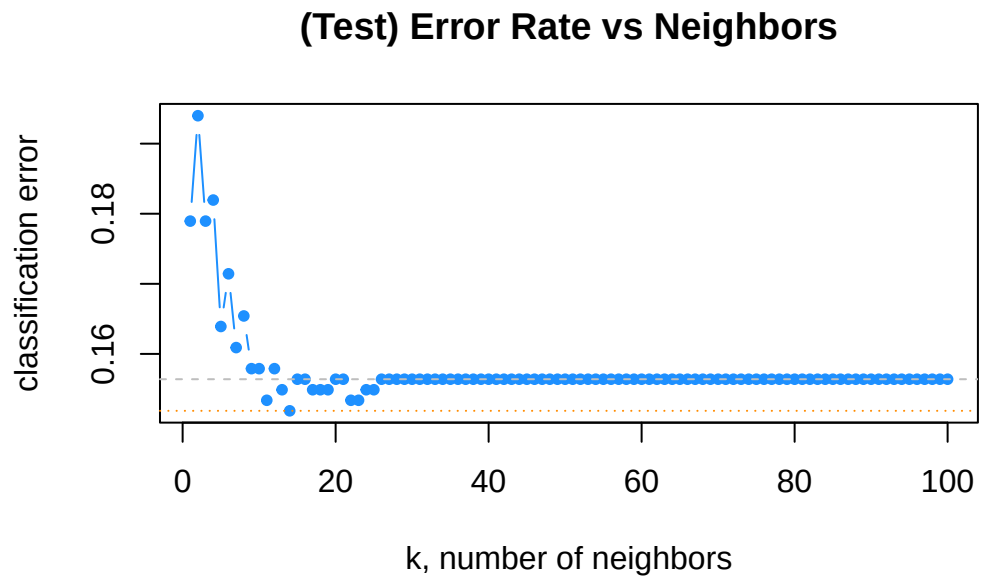
```
[1] 0.156391
```

```
calc_class_err(actual    = y_test,
               predicted = knn(train = scale(X_train),
                               test  = scale(X_test),
                               cl    = y_train,
                               k     = 5))
```

```
[1] 0.1639098
```

```r
set.seed(42)
k_to_try = 1:100
err_k = rep(x = 0, times = length(k_to_try))

for (i in seq_along(k_to_try)) {
  pred = knn(train = scale(X_train),
             test  = scale(X_test),
             cl    = y_train,
             k     = k_to_try[i],
             prob = T)
  err_k[i] = calc_class_err(y_test, pred)
}
```

```r
# plot error vs choice of k
plot(err_k, type = "b", col = "dodgerblue", cex = 1, pch = 20,
     xlab = "k, number of neighbors", ylab = "classification error",
     main = "(Test) Error Rate vs Neighbors")
# add line for min error seen
abline(h = min(err_k), col = "darkorange", lty = 3)
# add line for minority prevalence in test set
abline(h = mean(y_test == "Yes"), col = "grey", lty = 2)
```



```r
min(err_k)
```

```
[1] 0.1518797
```

```r
which(err_k == min(err_k))
```

```
[1] 14
```

```r
max(which(err_k == min(err_k)))
```

```
[1] 14
```

0.2.2.2 packages `Caret`

```r
set.seed(1994)
default_idx = createDataPartition(datos$Response, p = 0.7, list = FALSE)
train_caret = datos[default_idx, ]
test_caret = datos[-default_idx, ]
```

```r
modelLookup("knn")
```

```
  model parameter      label forReg forClass probModel
1   knn         k #Neighbors   TRUE     TRUE      TRUE
```

```r
entrenamiento <- train(Response ~ .,
  data = train_caret, method = "knn",
  trControl = trainControl(method = "cv", number = 5),
  # preProcess = c("center", "scale"),
  tuneGrid = expand.grid(k = seq(1, 31, by = 2)))

entrenamiento
```

```
k-Nearest Neighbors

1553 samples
  39 predictor
   2 classes: 'No', 'Yes'
```

```
No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 1242, 1242, 1244, 1242, 1242
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   1  0.7978043  0.19091718
   3  0.8435384  0.24664163
   5  0.8473845  0.24471015
   7  0.8460941  0.18202413
   9  0.8480317  0.16525357
  11  0.8467497  0.15970141
  13  0.8428870  0.13700638
  15  0.8415967  0.11938922
  17  0.8415967  0.12496952
  19  0.8435259  0.11085824
  21  0.8454510  0.12326182
  23  0.8454510  0.13354390
  25  0.8460983  0.12624563
  27  0.8461025  0.12667406
  29  0.8435259  0.09764204
  31  0.8454552  0.10762181

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 9.
```

```r
entrenamiento$modelType
```

```
[1] "Classification"
```

```r
get_best_result = function(caret_fit) {
  best = which(rownames(caret_fit$results) == rownames(caret_fit$bestTune))
  best_result = caret_fit$results[best, ]
  rownames(best_result) = NULL
  best_result
}
```

```r
head(entrenamiento$results, 5)
```

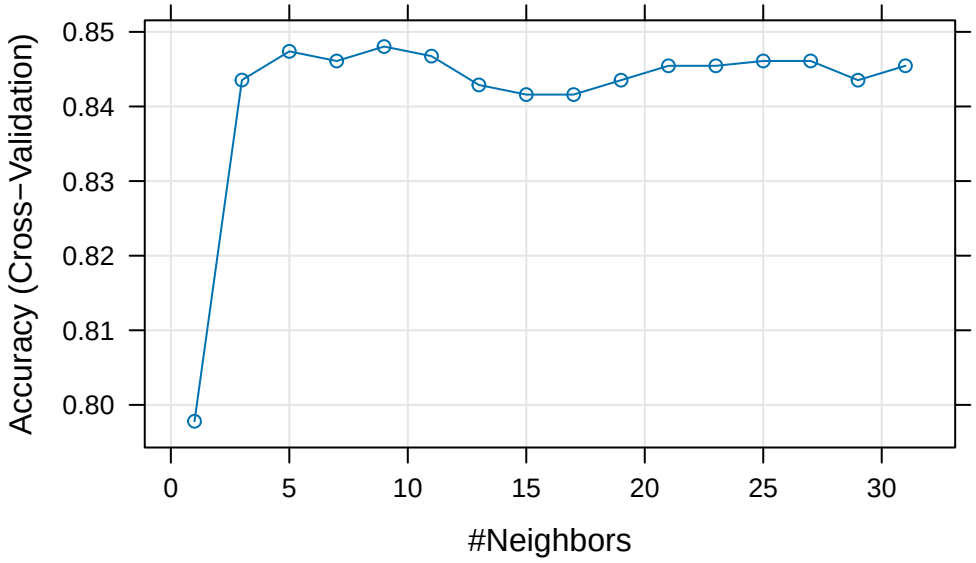
```
  k  Accuracy     Kappa  AccuracySD    KappaSD
1 1 0.7978043 0.1909172 0.016856624 0.02845537
```

```
2 3 0.8435384 0.2466416 0.013230515 0.06417137
3 5 0.8473845 0.2447102 0.004033850 0.03460935
4 7 0.8460941 0.1820241 0.009138246 0.02529977
5 9 0.8480317 0.1652536 0.009005269 0.04459688
```

```
get_best_result(entrenamiento)
```

```
  k  Accuracy     Kappa  AccuracySD    KappaSD
1 9 0.8480317 0.1652536 0.009005269 0.04459688
```

```
plot(entrenamiento)
```




```
entrenamiento$finalModel
```

```
9-nearest neighbor model
Training set outcome distribution:

  No  Yes
1319  234
```

```r
head(predict(entrenamiento, newdata = test_caret, type = "prob"), n = 10)
```

```
          No       Yes
1  0.8888889 0.1111111
2  0.7000000 0.3000000
3  1.0000000 0.0000000
4  1.0000000 0.0000000
5  1.0000000 0.0000000
6  0.7777778 0.2222222
7  0.8888889 0.1111111
8  0.7777778 0.2222222
9  0.8888889 0.1111111
10 1.0000000 0.0000000
```

```r
caret::confusionMatrix(predict(entrenamiento, newdata = test_caret), test_caret$Response)
```

0.2.2.3 Python

```python
cols = X_train.columns

from sklearn.preprocessing import StandardScaler
import pandas as pd

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_train = pd.DataFrame(X_train, columns=[cols])
X_test = pd.DataFrame(X_test, columns=[cols])
```

0.2.2.3.1 Fit K Neighbours Classifier to the training eet

```python
# import KNeighbors ClaSSifier from sklearn
from sklearn.neighbors import KNeighborsClassifier


# instantiate the model
knn = KNeighborsClassifier(n_neighbors=3)
```

```
# fit the model to the training set
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(n_neighbors=3)
```

0.2.2.3.2 Predict test-set results

```
y_pred = knn.predict(X_test)

y_pred
```

```
array(['No', 'No', 'No', 'No', 'No', 'Yes', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'Yes', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'Yes', 'No', 'No', 'No', 'No', 'Yes', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'Yes', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'Yes', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'Yes',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'Yes', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'Yes', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'Yes', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'Yes', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'Yes', 'No', 'No', 'No', 'No', 'Yes', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'Yes', 'No', 'No', 'No',
```

```
       'No', 'No', 'Yes', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'Yes', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'Yes', 'No', 'No', 'No', 'No', 'No', 'Yes',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'Yes', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'Yes', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'Yes', 'No', 'Yes', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'Yes', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'Yes',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'Yes', 'No', 'No', 'No', 'No', 'No', 'Yes', 'No', 'No',
       'Yes', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'Yes',
       'No', 'Yes', 'No', 'No', 'No', 'Yes', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'No', 'No', 'No', 'No', 'No', 'No'], dtype=object)
```

predict_proba method¶

```
# probability of getting output as 2 - benign cancer

knn.predict_proba(X_test)
```

```
array([[1.        , 0.        ],
       [0.66666667, 0.33333333],
       [1.        , 0.        ],
```

```
       ...,
       [1.        , 0.        ],
       [1.        , 0.        ],
       [1.        , 0.        ]], shape=(665, 2))
```

0.2.2.3.3 Check accuracy score

```
from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score: 0.8346
```

0.2.2.3.4 Comnpare train - test

```
y_pred_train = knn.predict(X_train)
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))
```

```
Training-set accuracy score: 0.8878
```

0.2.2.3.5 Check for overfitting and underfitting¶

```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(knn.score(X_train, y_train)))
```

```
Training set score: 0.8878
```

```
print('Test set score: {:.4f}'.format(knn.score(X_test, y_test)))
```

```
Test set score: 0.8346
```

0.2.2.3.6 Rebuild kNN Classification model using different values of k

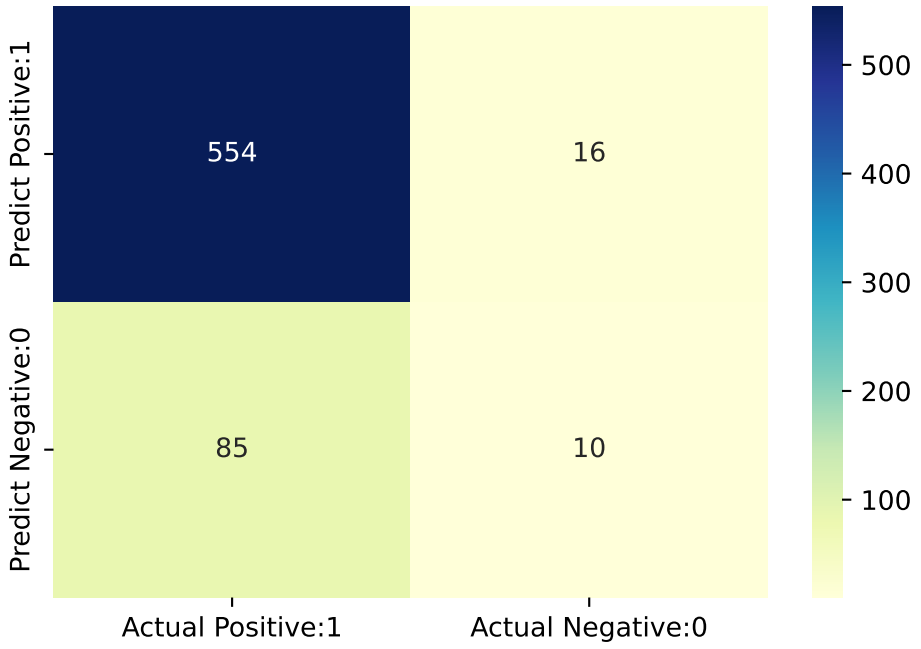
```
# instantiate the model with k=5
knn_5 = KNeighborsClassifier(n_neighbors=5)


# fit the model to the training set
knn_5.fit(X_train, y_train)
```

```
KNeighborsClassifier()
```

```
# predict on the test-set
y_pred_5 = knn_5.predict(X_test)


print('Model accuracy score with k=5 : {0:0.4f}'. format(accuracy_score(y_test, y_pred_5)))
```

```
Model accuracy score with k=5 : 0.8481
```

```
import matplotlib.pyplot as plt # for data visualization purposes
import seaborn as sns # for data visualization
from sklearn.metrics import confusion_matrix

# visualize confusion matrix with seaborn heatmap

plt.figure(figsize=(6,4))

cm_7 = confusion_matrix(y_test, y_pred_5)

cm_matrix = pd.DataFrame(data=cm_7, columns=['Actual Positive:1', 'Actual Negative:0'], inde

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```




```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

          No       0.87      0.96      0.91       570
         Yes       0.29      0.11      0.15        95

    accuracy                           0.83       665
   macro avg       0.58      0.53      0.53       665
weighted avg       0.78      0.83      0.80       665
```

0.2.3 KNN Regresor

```
library(FNN)
library(MASS)
data(Boston)
```

```r
set.seed(42)
boston_idx = sample(1:nrow(Boston), size = 250)
trn_boston = Boston[boston_idx, ]
tst_boston  = Boston[-boston_idx, ]
```

```r
X_trn_boston = trn_boston[-ncol(trn_boston)]
X_tst_boston = tst_boston[-ncol(trn_boston)]
y_trn_boston = trn_boston["medv"]
y_tst_boston = tst_boston["medv"]
```

```r
pred_001 = knn.reg(train = X_trn_boston, test = X_tst_boston, y = y_trn_boston, k = 1)
pred_005 = knn.reg(train = X_trn_boston, test = X_tst_boston, y = y_trn_boston, k = 5)
pred_010 = knn.reg(train = X_trn_boston, test = X_tst_boston, y = y_trn_boston, k = 10)
pred_050 = knn.reg(train = X_trn_boston, test = X_tst_boston, y = y_trn_boston, k = 50)
pred_100 = knn.reg(train = X_trn_boston, test = X_tst_boston, y = y_trn_boston, k = 100)
pred_250 = knn.reg(train = X_trn_boston, test = X_tst_boston, y = y_trn_boston, k = 250)
```

```r
rmse = function(actual, predicted) {
  sqrt(mean((actual - predicted) ^ 2))
}
```

```r
# define helper function for getting knn.reg predictions
# note: this function is highly specific to this situation and dataset
make_knn_pred = function(k = 1, training, predicting) {
  pred = FNN::knn.reg(train = training["lstat"],
                      test = predicting["lstat"],
                      y = training$medv, k = k)$pred
  act  = predicting$medv
  rmse(predicted = pred, actual = act)
}
```

```r
# define values of k to evaluate
k = c(1, 5, 10, 25, 50, 250)

# get requested train RMSEs
knn_trn_rmse = sapply(k, make_knn_pred,
                      training = trn_boston,
                      predicting = trn_boston)
```

```
# get requested test RMSEs
knn_tst_rmse = sapply(k, make_knn_pred,
                      training = trn_boston,
                      predicting = tst_boston)
```

```
# determine "best" k
best_k = k[which.min(knn_tst_rmse)]
```

```
# find overfitting, underfitting, and "best"" k
fit_status = ifelse(k < best_k, "Over", ifelse(k == best_k, "Best", "Under"))
```

```
# summarize results
knn_results = data.frame(
  k,
  round(knn_trn_rmse, 2),
  round(knn_tst_rmse, 2),
  fit_status
)
colnames(knn_results) = c("k", "Train RMSE", "Test RMSE", "Fit?")

# display results
knitr::kable(knn_results, escape = FALSE, booktabs = TRUE)
```

k	Train RMSE	Test RMSE	Fit?
1	1.65	8.32	Over
5	4.98	5.83	Over
10	5.26	5.05	Over
25	5.51	4.79	Best
50	5.94	5.05	Under
250	9.61	8.75	Under

0.2.3.1 packages `Caret`

```
caret::modelLookup("knn")
```

```
  model parameter      label forReg forClass probModel
1   knn         k #Neighbors   TRUE     TRUE      TRUE
```

```r
library("CDR")
library("class")
library("caret")
library("reshape")
library("ggplot2")
```

```r
data(dp_entr_NUM)
head(dp_entr_NUM)
```

```
  ind_pro11 ind_pro12 ind_pro14 ind_pro15 ind_pro16 ind_pro17
1         1         0         1         1         1         0
2         0         0         1         0         1         0
3         0         0         1         1         1         1
4         0         1         1         0         0         0
5         0         1         1         0         1         0
6         1         0         1         0         0         0
  des_nivel_edu.ALTO des_nivel_edu.BASICO des_nivel_edu.MEDIO importe_pro11
1                  0                    0                   1           157
2                  0                    0                   1             0
3                  0                    1                   0             0
4                  0                    0                   1             0
5                  0                    1                   0             0
6                  1                    0                   0           115
  importe_pro12 importe_pro14 importe_pro15 importe_pro16 importe_pro17 edad
1             0            40           200           180             0   49
2             0           240             0           180             0   38
3             0           425           200           180           300   61
4           120            60             0             0             0   47
5           120           133             0           180             0   34
6             0           220             0             0             0   43
  tamano_fam anos_exp ingresos_ano CLS_PRO_pro13
1          4       24        30000             S
2          2       12        53000             N
3          4       37       172000             S
4          3       21        38000             N
5          1       10        38000             N
6          2       18        60000             N
```

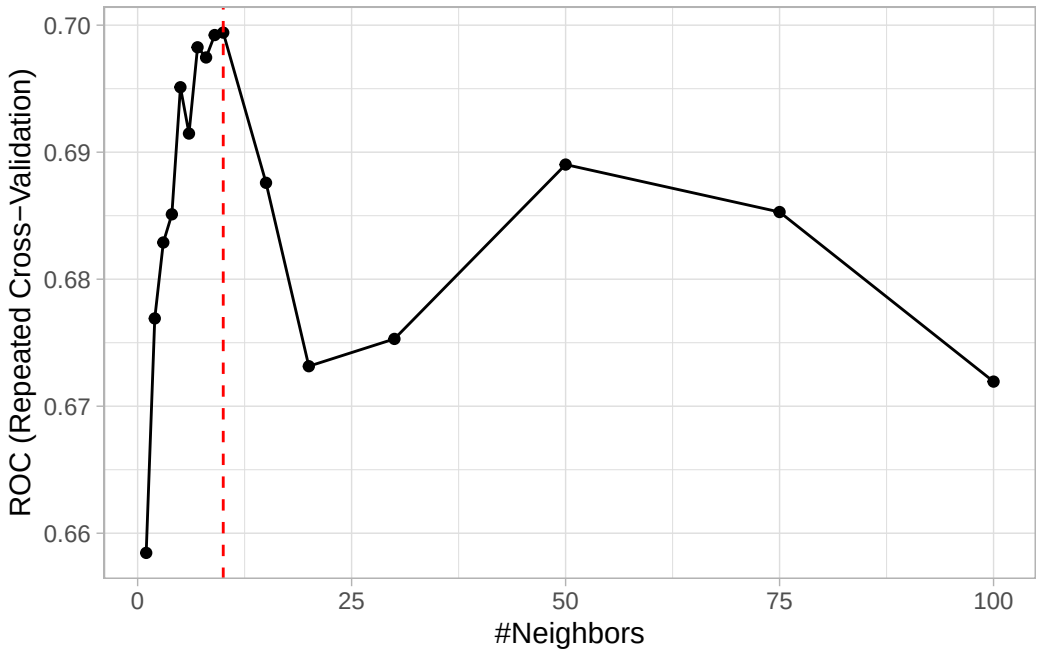
```r
# Definimos un método de remuestreo
cv <- trainControl(
  method = "repeatedcv",
```

```
  number = 10,
  repeats = 5,
  classProbs = TRUE,
  preProcOptions = list("center"),
  summaryFunction = twoClassSummary
)
```

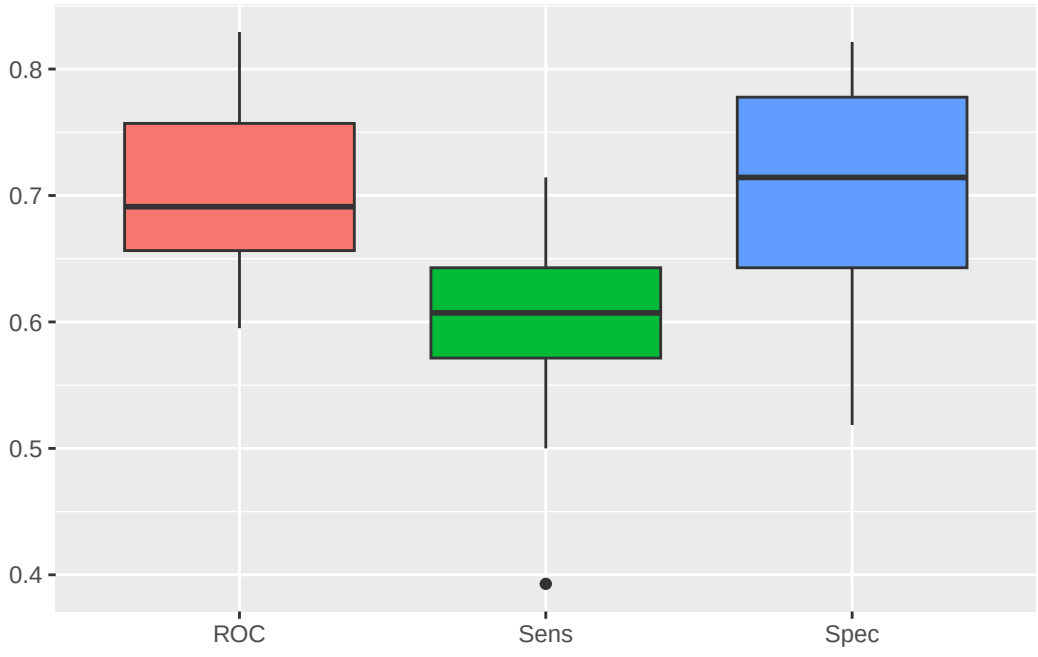
```
# Definimos la red de posibles valores del hiperparámetro
hyper_grid <- expand.grid(k = c(1:10,15,20,30,50,75,100))
```

```
set.seed(101)
# Se entrena el modelo ajustando el hiperparámetro óptimo
model <- train(
  CLS_PRO_pro13 ~ .,
  data = dp_entr_NUM,
  method = "knn",
  trControl = cv,
  tuneGrid = hyper_grid,
  metric = "ROC"
)
```

```
ggplot(model) +
  geom_vline(xintercept = unlist(model$bestTune),col="red",linetype="dashed") +
  theme_light()
```




```
ggplot(melt(model$resample[,-4]), aes(x = variable, y = value, fill=variable)) +
   geom_boxplot(show.legend=FALSE) +
  xlab(NULL) + ylab(NULL)
```



```r
set.seed(101)
confusionMatrix(predict(model), dp_entr_NUM$CLS_PRO_pro13)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   S   N
         S 186  65
         N  93 214

               Accuracy : 0.7168
                 95% CI : (0.6775, 0.7539)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.4337

 Mcnemar's Test P-Value : 0.03171

            Sensitivity : 0.6667
            Specificity : 0.7670
         Pos Pred Value : 0.7410
         Neg Pred Value : 0.6971
             Prevalence : 0.5000
         Detection Rate : 0.3333
   Detection Prevalence : 0.4498
      Balanced Accuracy : 0.7168

       'Positive' Class : S
```

0.3 Naives Bayes

0.3.1 NB Classifier

```r
set.seed(430)
iris_obs = nrow(iris)
iris_idx = sample(iris_obs, size = trunc(0.50 * iris_obs))
# iris_index = sample(iris_obs, size = trunc(0.10 * iris_obs))
```

```r
iris_trn = iris[iris_idx, ]
iris_tst = iris[-iris_idx, ]
```

```r
library(e1071)

iris_nb = naiveBayes(Species ~ ., data = iris_trn)
iris_nb
```

```
Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
    setosa versicolor  virginica
 0.3866667  0.2933333  0.3200000

Conditional probabilities:
            Sepal.Length
Y                [,1]      [,2]
  setosa     4.958621 0.3212890
  versicolor 6.063636 0.5636154
  virginica  6.479167 0.5484993

            Sepal.Width
Y                [,1]      [,2]
  setosa     3.420690 0.4012296
  versicolor 2.845455 0.3262007
  virginica  2.937500 0.3267927

            Petal.Length
Y                [,1]      [,2]
  setosa     1.458621 0.1880677
  versicolor 4.318182 0.5543219
  virginica  5.479167 0.4995469

            Petal.Width
Y                [,1]       [,2]
  setosa     0.237931 0.09788402
  versicolor 1.354545 0.21979920
```

```
  virginica  2.045833 0.29039578
```

```
head(predict(iris_nb, iris_trn))
```

```
[1] setosa     versicolor versicolor setosa     virginica  versicolor
Levels: setosa versicolor virginica
```

```
head(predict(iris_nb, iris_trn, type = "class"))
```

```
[1] setosa     versicolor versicolor setosa     virginica  versicolor
Levels: setosa versicolor virginica
```

```
head(predict(iris_nb, iris_trn, type = "raw"))
```

```
            setosa   versicolor    virginica
[1,]  1.000000e+00 3.096444e-15 5.172277e-27
[2,]  1.079241e-93 9.833098e-01 1.669021e-02
[3,] 6.378471e-106 9.210439e-01 7.895614e-02
[4,]  1.000000e+00 1.691578e-16 2.882941e-28
[5,] 1.791407e-209 3.462703e-04 9.996537e-01
[6,]  4.538228e-59 9.999316e-01 6.835677e-05
```

```
iris_nb_trn_pred = predict(iris_nb, iris_trn)
iris_nb_tst_pred = predict(iris_nb, iris_tst)
```

```
calc_class_err(predicted = iris_nb_trn_pred, actual = iris_trn$Species)
```

```
[1] 0.05333333
```

```
calc_class_err(predicted = iris_nb_tst_pred, actual = iris_tst$Species)
```

```
[1] 0.02666667
```

```
table(predicted = iris_nb_tst_pred, actual = iris_tst$Species)
```

```
            actual
predicted    setosa versicolor virginica
  setosa         21          0         0
  versicolor      0         28         2
  virginica       0          0        24
```

0.3.2 Packages `caret`

```
library("caret")
library("naivebayes")
library("reshape")
library("ggplot2")
library("CDR")

data("dp_entr")
```

```
# se fija la semilla aleatoria
set.seed(101)

# se entrena el modelo
model <- train(CLS_PRO_pro13 ~ .,
            data=dp_entr,
            method="nb",
            metric="Accuracy",
            trControl=trainControl(classProbs = TRUE,
                                   method = "cv",
                                   number = 10))
```

```
# se muestra la salida del modelo
model
```

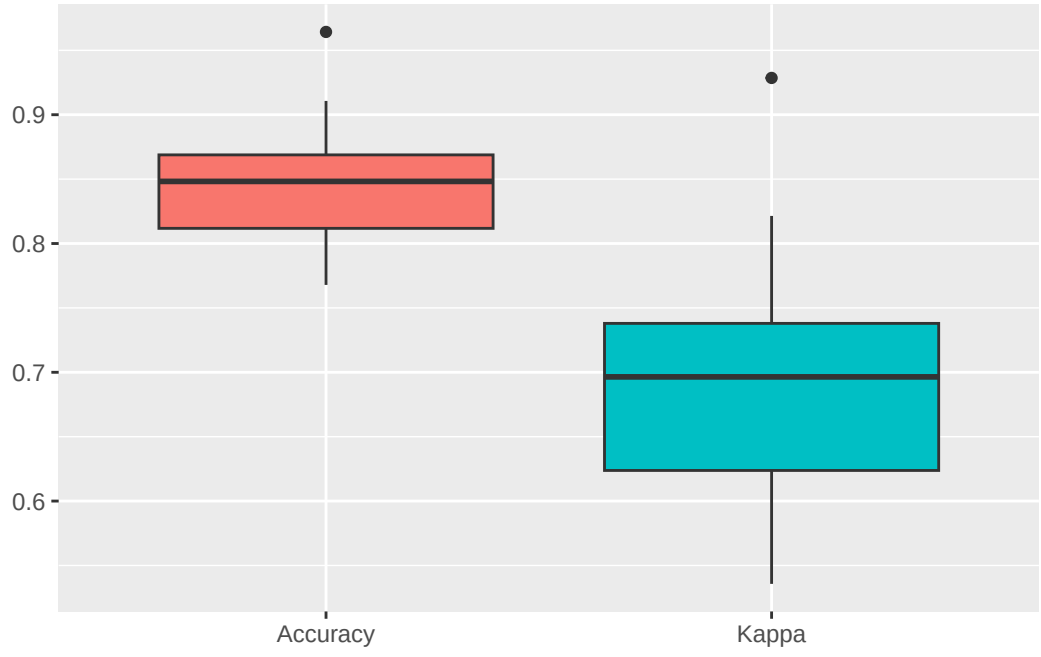
```
Naive Bayes

558 samples
 17 predictor
  2 classes: 'S', 'N'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 502, 502, 502, 503, 503, 502, ...
Resampling results across tuning parameters:

  usekernel  Accuracy   Kappa
  FALSE      0.8512662  0.7026716
   TRUE      0.8512338  0.7025165

Tuning parameter 'fL' was held constant at a value of 0
```

```
Tuning
 parameter 'adjust' was held constant at a value of 1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were fL = 0, usekernel = FALSE and adjust
 = 1.
```

```
confusionMatrix(model)
```

```
Cross-Validated (10 fold) Confusion Matrix

(entries are percentual average cell counts across resamples)

          Reference
Prediction    S    N
         S 41.8  6.6
         N  8.2 43.4

 Accuracy (average) : 0.8513
```

```
ggplot(melt(model$resample[,-4]), aes(x = variable, y = value, fill=variable)) +
  geom_boxplot(show.legend=FALSE) +
  xlab(NULL) + ylab(NULL)
```



0.4 Bibliografia

- https://daviddalpiaz.github.io/r
-