KNN: K-Nearest Neighbors o K vecinos más cercanos

Packages base de datos

```
library(dummies)

## dummies-1.5.6 provided by Decision Patterns

library(FNN) #ara aplicar KNN
library(scales)#Para escalados
library(caret) #Para las particiones

## Loading required package: ggplot2
## Loading required package: lattice

setwd("C:\\Users\\81799\\OneDrive\\Documentos\\ESFM_CLASES\\Servicio Social ARTF\\Machine Learni
edu <- read.csv("education.csv")
```

Observación a la tabla edu

```
head(edu, 5)

##   state region urban income under18 expense
## 1    ME      1   508   3944     325     235
## 2    NH      1   564   4578     323     231
## 3    VT      1   322   4011     328     270
## 4    MA      1   846   5233     305     261
## 5    RI      1   871   4780     303     300
```

Las columnas son las siguientes:

- state: Estados del gobierno.
- region: Cada número significa la región en la que pertence el estado (Variable categórica)
- urban: Es el número de residentes por cada $1,000$ en los distritors urbanos medidos en el año 1970
- income: Es la renta per cápita de 1973 que ganaban en promedio.
- under18: Número de residentes por cada $1,000$ que tienen menos de 18 años.
- expense: El gasto que se preveía llevar a cabo en educación por cada estado en el año siguiente-

Se puede elaborar el modelo KNN para prever el gasto basado en el resto de predictores, es decir, si ahora tenemos otro estado con unas ciertas condiciones o cambian las condiciones urbanas de ingresos o de habitantes por debajo de 18 años de cada uno de los estados o regiones. ¿ Cuanto se cree que se va a gastar en Educación el Gobierno?

Generamos variables dummies para la variable categórica region"

```
dms <- dummy(edu$region, sep = "_")#Variable categóricas

## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts = FALSE): non-list
contrasts argument ignored

edu <- cbind(edu, dms) #Unimos las varables dummies a los datos de edu
head(edu,5)
```

```
##   state region urban income under18 expense KNN_R.Rnw_1 KNN_R.Rnw_2 KNN_R.Rnw_3
## 1    ME      1   508   3944     325     235           1           0           0
## 2    NH      1   564   4578     323     231           1           0           0
## 3    VT      1   322   4011     328     270           1           0           0
## 4    MA      1   846   5233     305     261           1           0           0
## 5    RI      1   871   4780     303     300           1           0           0
##   KNN_R.Rnw_4
## 1           0
## 2           0
## 3           0
## 4           0
## 5           0
```

Normalizamos los datos

```
edu$urban.s <- rescale(edu$urban) #Normalizamos la columna urban
edu$income.s <- rescale(edu$income) #Normalizamos la columna income
edu$under18.s <- rescale(edu$under18) #Normalizamos la columna under18
```

```
set.seed(2018) #Generación de semilla
t.id <- createDataPartition(edu$expense, p=0.6, list = F)#Conjunto de entrenamiento
tr <- edu[t.id, ] #Datos de entrenamiento
temp <- edu[-t.id, ]#Datos que no pertecen a los de entrenamiento
v.id <- createDataPartition(temp$expense, p=0.5, list = F)#Partición al 50%
val <- temp[v.id,]#Datos de validación
test <- temp[-v.id,] #Datos de testing
```

Función para calcularla raíz del error cuadrático medio

```
rmse <- function(prediccion, original){
  rmse <- sqrt(mean((prediccion-original)^2))
  return(rmse)
}
```

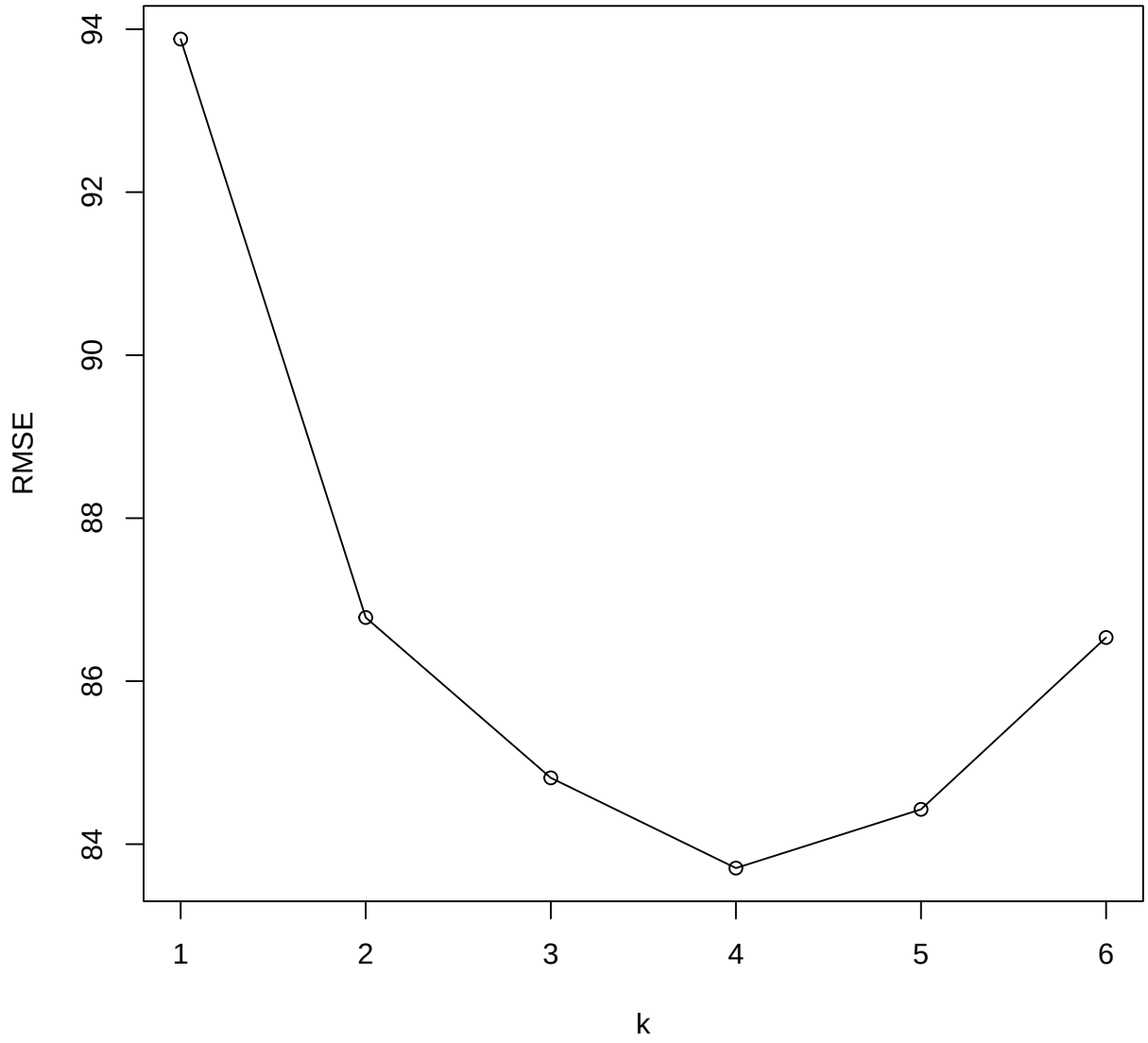
Modelos KNN

```
# Modelo con K=1
reg1 <- knn.reg(tr[,7:12], val[,7:12], tr$expense, k=1,
                algorithm = "brute") #Calculo del modelo
rmse1 <- sqrt(mean((reg1$pred-val$expense)^2))
rmse1 #Raíz del error cuadratico delo modelo K=1

## [1] 93.87865

# Modelo con K=2
reg2 <- knn.reg(tr[,7:12], val[,7:12], tr$expense, k=2,
                algorithm = "brute") #Calculo del modelo
rmse2 <- rmse(val$expense, reg2$pred)
rmse2 #Raíz del error cuadratico delo modelo K=2

## [1] 86.78105

# Modelo con K=3
reg3 <- knn.reg(tr[,7:12], val[,7:12], tr$expense, k=3,
                algorithm = "brute")#Calculo del modelo
rmse3 <- rmse(val$expense, reg3$pred)
rmse3 #Raíz del error cuadratico delo modelo K=3
```

```
## [1] 84.81359

# Modelo con K=4
reg4 <- knn.reg(tr[,7:12], val[,7:12], tr$expense, k=4,
                algorithm = "brute") #Calculo del modelo
rmse4 <- rmse(val$expense, reg4$pred)
rmse4 #Raíz del error cuadratico delo modelo K=4

## [1] 83.70633

#Modelo con K=5
reg5 <- knn.reg(tr[,7:12], val[,7:12], tr$expense, k=5,
                algorithm = "brute")
rmse5 <- rmse(val$expense, reg5$pred)
rmse5

## [1] 84.42684

#Modelo con K=6
reg6 <- knn.reg(tr[,7:12], val[,7:12], tr$expense, k=6,
                algorithm = "brute")
rmse6 <- rmse(val$expense, reg6$pred)
rmse6

## [1] 86.53572
```

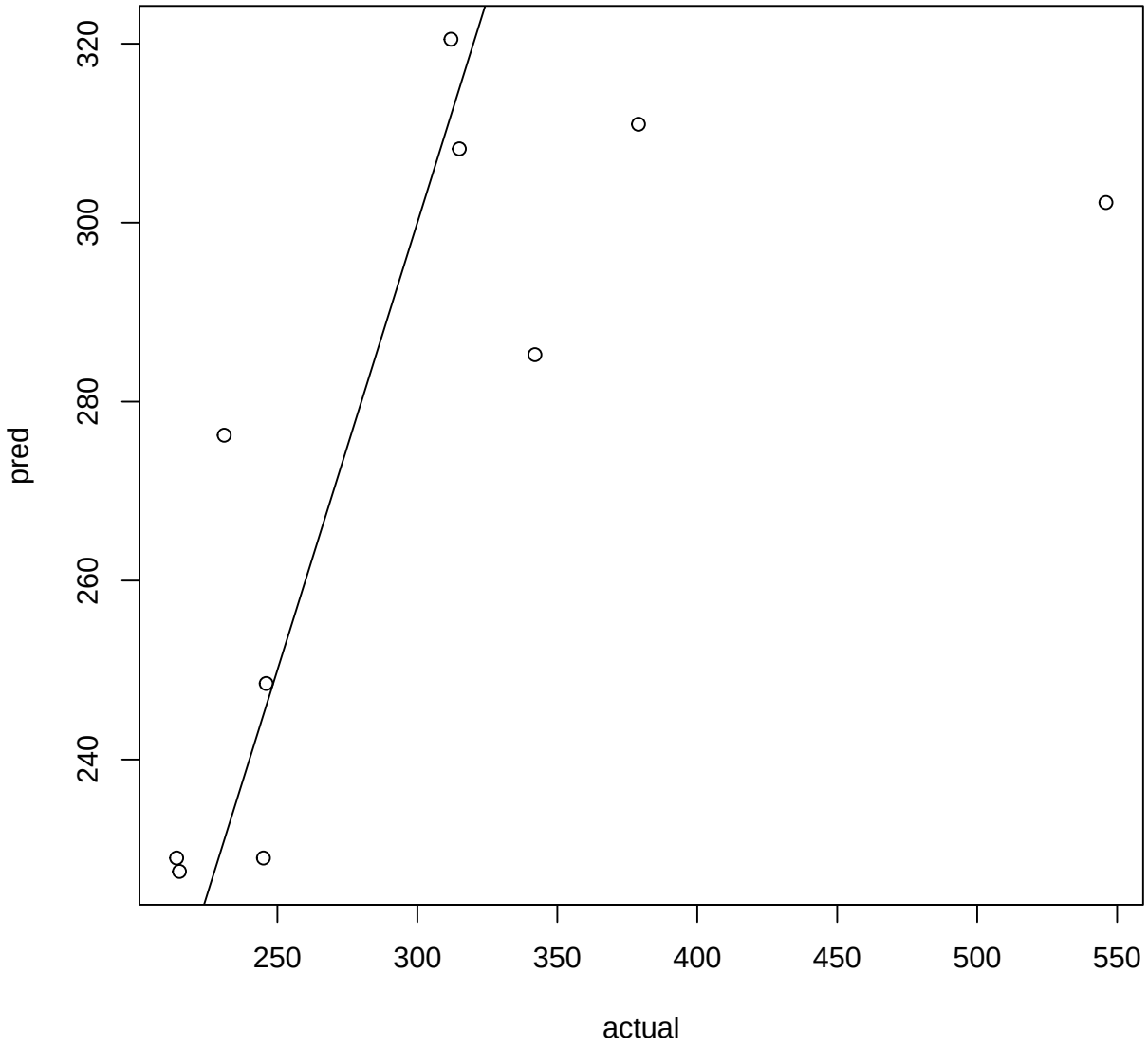
Graficando las raíces de los errores medios de cada modelo

```
#Vector con las raíces de los erroes medios de los modelos
errors = c(rmse1, rmse2, rmse3, rmse4,rmse5, rmse6)
plot(errors, type = 'o', xlab = "k", ylab = "RMSE")
```



Concluimos que el mínimo se alcanza con 4 vecinos cercanos para tomar la decisión. La gráfica respecto a los datos originlaes queda de la siguiente manera:

```
df = data.frame(actual = val$expense, pred = reg4$pred)
plot(df)
abline(0,1)
```

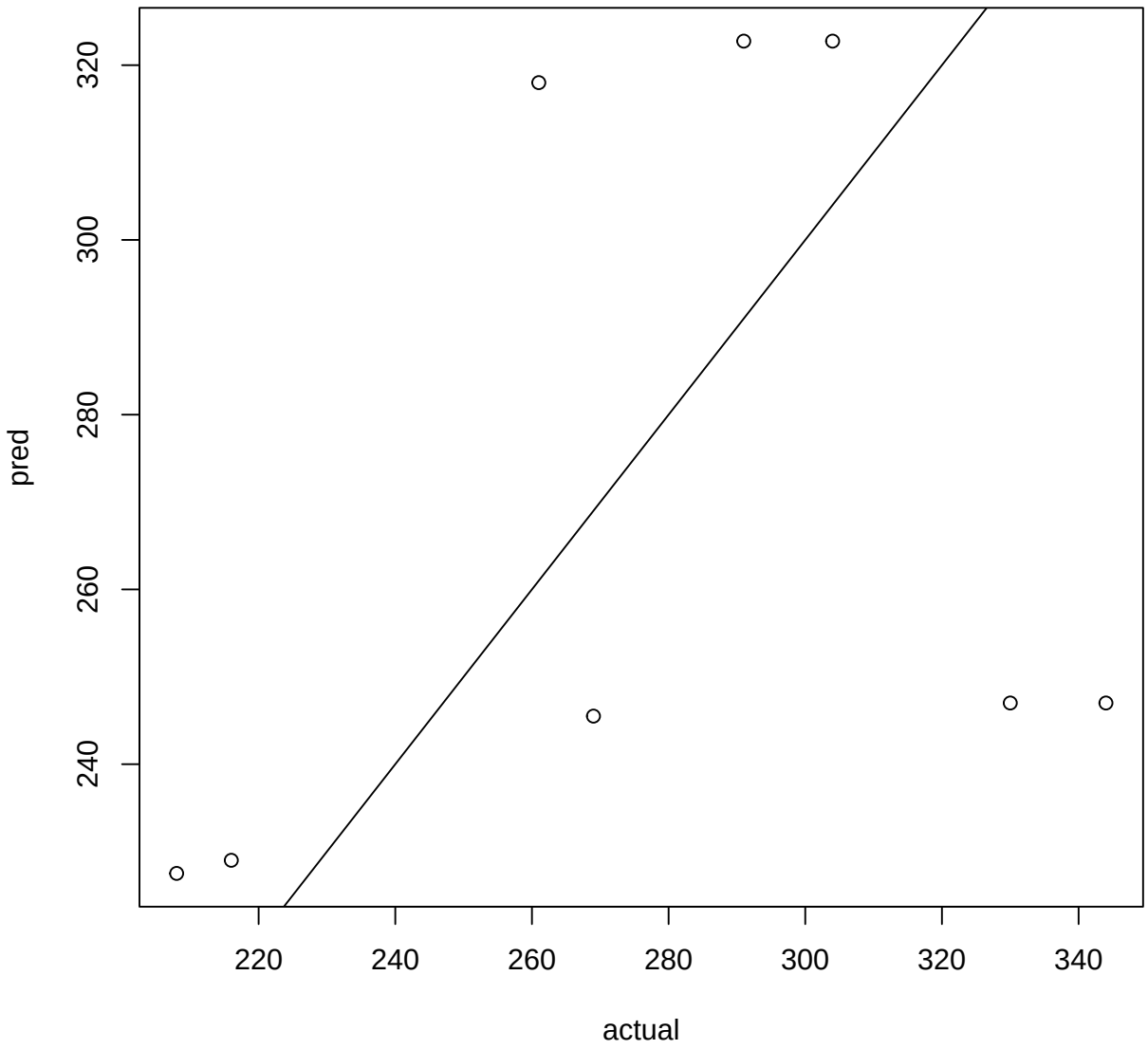


Como se determino que un $K = 4$ era un modelo adecuado, entonces ahora aplicamos la regresión al conjunto de testing con $K = 4$

```
reg.test <- knn.reg(tr[,7:12], test[,7:12], tr$expense, k=4,
                algorithm = "brute")
rmse.test <- rmse(test$expense, reg.test$pred)
rmse.test #Ráiz del error cuadrático del conjunto testing

## [1] 52.45013
```

Su gráfica respecto a los datos originales queda de la siguiente manera:

```
df = data.frame(actual = test$expense, pred = reg.test$pred)
plot(df)
abline(0,1)
```



KNN sin partición de validación.

```
reg <- knn.reg(tr[,7:12], test = NULL, y = tr$expense,
               k = 4, algorithm = "brute")
rmse.reg <- sqrt(mean(reg$residuals^2))
rmse.reg

## [1] 41.36466
```

Función para automatizar KNN

```
##Función para automatizar KNN
rdacb.knn.reg <- function(tr_predictor, val_predictors,
                          tr_target, val_target, k){
  library(FNN)
  res <- knn.reg(tr_predictor, val_predictors,
```

```
                tr_target, k, algorithm = "brute")
  rmserror <- sqrt(mean((val_target - res$pred)^2))
  cat(paste("RMSE para k = ", toString(k), ": ", rmserror,"\n", sep = ""))
  rmserror
}
```

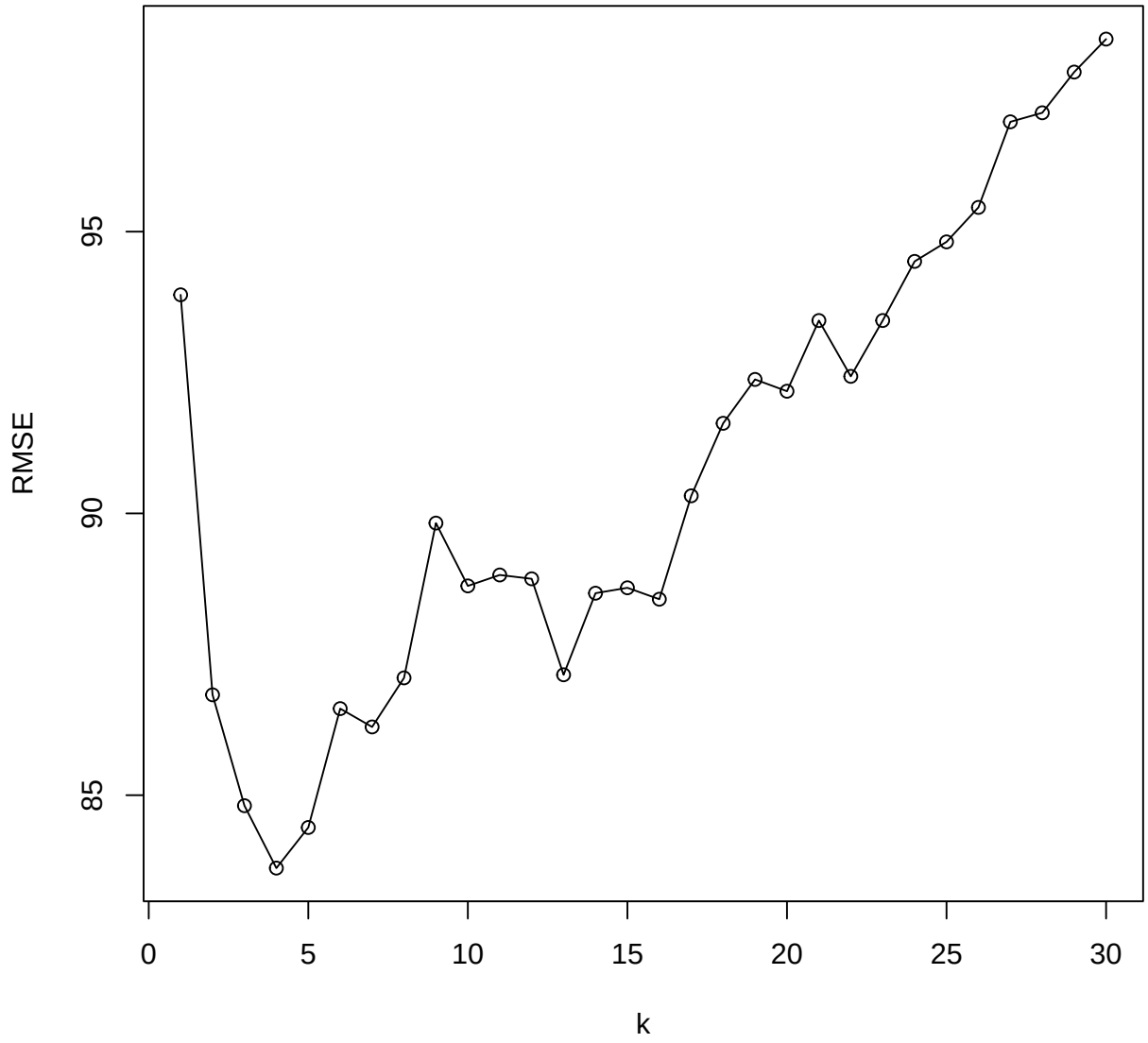
Usando la función anterior, podremos optimizar tiempo para calcular la KNN

```
rdacb.knn.reg(tr[,7:12], val[,7:12], tr$expense,val$expense, k=1)

## RMSE para k = 1: 93.8786450690465
## [1] 93.87865

rdacb.knn.reg(tr[,7:12], val[,7:12], tr$expense,val$expense, k=2)

## RMSE para k = 2: 86.78104631773
## [1] 86.78105

rdacb.knn.reg(tr[,7:12], val[,7:12], tr$expense,val$expense, k=3)

## RMSE para k = 3: 84.8135864378134
## [1] 84.81359

rdacb.knn.reg(tr[,7:12], val[,7:12], tr$expense,val$expense, k=4)

## RMSE para k = 4: 83.7063318990864
## [1] 83.70633

rdacb.knn.reg(tr[,7:12], val[,7:12], tr$expense,val$expense, k=5)

## RMSE para k = 5: 84.4268440722499
## [1] 84.42684

rdacb.knn.reg(tr[,7:12], val[,7:12], tr$expense,val$expense, k=6)

## RMSE para k = 6: 86.5357183800744
## [1] 86.53572
```

Función para realizar múltiples KNN

```
rda.knn.reg.multi <- function(tr_predictors, val_predictors,
                              tr_target, val_target, start_k, end_k){
  rms_errors <- vector()
  for(k in start_k:end_k){
    rms_error <- rdacb.knn.reg(tr_predictors, val_predictors,
                               tr_target, val_target, k)
    rms_errors <- c(rms_errors, rms_error)
  }
  plot(rms_errors, type = 'o', xlab = "k", ylab = "RMSE")
}
rda.knn.reg.multi(tr[,7:12], val[,7:12],
                  tr$expense, val$expense, 1,30)

## RMSE para k = 1: 93.8786450690465
## RMSE para k = 2: 86.78104631773
## RMSE para k = 3: 84.8135864378134
```

```
## RMSE para k = 4: 83.7063318990864
## RMSE para k = 5: 84.4268440722499
## RMSE para k = 6: 86.5357183800744
## RMSE para k = 7: 86.210788188022
## RMSE para k = 8: 87.0813735824143
## RMSE para k = 9: 89.8283823276638
## RMSE para k = 10: 88.7133980861966
## RMSE para k = 11: 88.9073758897613
## RMSE para k = 12: 88.8395166703547
## RMSE para k = 13: 87.1364637428526
## RMSE para k = 14: 88.5840601591828
## RMSE para k = 15: 88.6802445744134
## RMSE para k = 16: 88.4773386417392
## RMSE para k = 17: 90.3124583787215
## RMSE para k = 18: 91.5990912537811
## RMSE para k = 19: 92.3774564481714
## RMSE para k = 20: 92.1681479688075
## RMSE para k = 21: 93.4223251637544
## RMSE para k = 22: 92.4321115626701
## RMSE para k = 23: 93.4230352539889
## RMSE para k = 24: 94.4719406461222
## RMSE para k = 25: 94.8181931909694
## RMSE para k = 26: 95.430227152815
## RMSE para k = 27: 96.9485835191257
## RMSE para k = 28: 97.1091774599022
## RMSE para k = 29: 97.8310617689954
## RMSE para k = 30: 98.4166567880322
```


RMSE
85
90
95
0
5
10
15
20
25
30
k