Treball_Bio1

Victor Casals, Ivan Castillo, Sergi Esturi, Ferran Garcia i Blanca Rodríguez

2024-03-21

Contents


```
library(corrplot)
library(knitr)
library(ROCR)
library(class)
library(kernlab)
library(caret)
```

Objectius de l'estudi

L'objectiu de l'estudi és implementar diversos algorismes de Machine Learning per predir la localització subcel · lular de proteïnes. A més a més, un cop implementats els algorismes en qüestió, es compararan els seus rendiments per escollir-ne el de major rendiment. És a dir, es buscarà quin és l'algorisme que millor prediu la localització subcel · lular d'una proteina.

Mètodes utilitzats

Es desenvoluparan dos algoritmes de Machine Learning: el K-NN (K-Nearest Neighbour) i l'SVM (Support Vector Machine). Utilitzant 3 fold-cross validation s'exploraran el k-NN amb els valors per al nombre de veïns k = 1, 11, 21, 31, i per al SVM s'exploraran les funcions kernel lineal i rbf.

El mètode de 3 fold cross-validation consisteix en entrenar l'algoritme fent servir tres submostres de les dades d'entrenament o training, i finalment evaluar-ho amb la totalitat de les dades. Amb aquest procediment ajudem al model a tenir una bona capacitat de generalització i que funcioni de manera més acurada amb dades noves.

K-Nearest Neighbour

En aquest mètode, l'objectiu és classificar les dades segons la categoria de les k observacions més properes. L'algoritme k-NN suposa que hi ha observacions similars a prop. Observeu a la imatge de dalt que la majoria de les vegades, punts de dades similars estan a prop els uns dels altres. L'algorisme k-NN depèn que aquesta hipòtesi és prou certa perquè l'algorisme sigui útil. K-NN captura la idea de similitud a través del càlcul de la distància entre els punts. Aquesta distància sovint es mesura amb la distància euclidiana, tot i que també hi ha altres mesures de distància com la distància de Manhattan.

Support Vector Machine

Support Vector Machines (SVM) és un algorisme d'aprenentatge supervisat que s'utilitza per a problemes de classificació i regressió. L'objectiu del SVM és trobar la millor línia o hiperplà que divideixi les dades en classes diferents.

En altres paraules, l'algorisme busca una frontera de decisió que separi les dades en classes distintes, de manera que les dades de cada classe es trobin a un costat o l'altre de la frontera. Els vectors de suport són els punts que estan més propers a la frontera i ajuden a definir-la. El SVM és útil per a problemes de classificació on les dades no són linealment separables. En aquests casos, l'algorisme pot utilitzar una funció kernel per a transformar les dades en un espai de característiques de dimensió superior, on les dades poden ser separades per una frontera de decisió lineal.

El SVM és una eina potent per a l'aprenentatge automàtic, ja que pot treballar amb dades de dimensions altes i és capaç d'aprendre patrons complexos en les dades. També és robust a l'overfitting i pot ser utilitzat per a problemes de classificació binària i multiclasse.

La principal diferència entre el kernel lineal i el kernel RBF és el tipus de frontera de decisió que s'obté en cada cas. El kernel lineal genera una frontera de decisió lineal, mentre que el kernel RBF genera una frontera de decisió no lineal. En altres paraules, el kernel lineal assumeix que les dades són separables per una línia recta, mentre que el kernel RBF permet que la frontera de decisió sigui una superfície complexa i no lineal.

Això fa que el kernel RBF sigui més adequat per a problemes on les dades no són linealment separables. No obstant això, és important tenir en compte que el kernel RBF pot ser més propens a l'overfitting que el kernel lineal, especialment si s'usa amb un paràmetre gamma molt gran.

En resum, la principal diferència entre el kernel lineal i el kernel RBF és la naturalesa de la frontera de decisió que generen, sent el kernel lineal més simple però limitat i el kernel RBF més complex però flexible. La elecció del kernel depèn del problema en qüestió i la naturalesa de les dades.

```r
data <- read.table(paste0(wd, "DATA/", "yeast.data"),
                   quote="\"", comment.char = "")
colnames(data) <- c("nom_seq", "mcg", "gvh", "alm", "mit",
                    "erl", "pox", "vac", "nuc", "location")
```

```r
# Crear un vector amb els valors a considerar
valors_considerats <- c("CYT", "ME1", "ME2", "ME3", "MIT", "NUC")

# Seleccionar les files amb valors en la columna "MIT" que estàn a la llista de valors a considerar
data_filtrat <- subset(data, location %in% valors_considerats)
```

```r
# Reasignar els valors de "ME1", "ME2" y "ME3" com "MEM"
data_filtrat[["location"]] <- ifelse(
  data_filtrat[["location"]] %in% c("ME1", "ME2", "ME3"),
  "MEM", data_filtrat[["location"]]
  ) |> as.factor()
```

Descriptiva

```r
head(data_filtrat)
```

```
##      nom_seq  mcg  gvh  alm  mit erl pox  vac  nuc location
## 1 ADT1_YEAST 0.58 0.61 0.47 0.13 0.5 0.0 0.48 0.22      MIT
## 2 ADT2_YEAST 0.43 0.67 0.48 0.27 0.5 0.0 0.53 0.22      MIT
## 3 ADT3_YEAST 0.64 0.62 0.49 0.15 0.5 0.0 0.53 0.22      MIT
## 4 AAR2_YEAST 0.58 0.44 0.57 0.13 0.5 0.0 0.54 0.22      NUC
## 5 AATM_YEAST 0.42 0.44 0.48 0.54 0.5 0.0 0.48 0.22      MIT
## 6 AATC_YEAST 0.51 0.40 0.56 0.17 0.5 0.5 0.49 0.22      CYT
```

```r
str(data_filtrat)
```

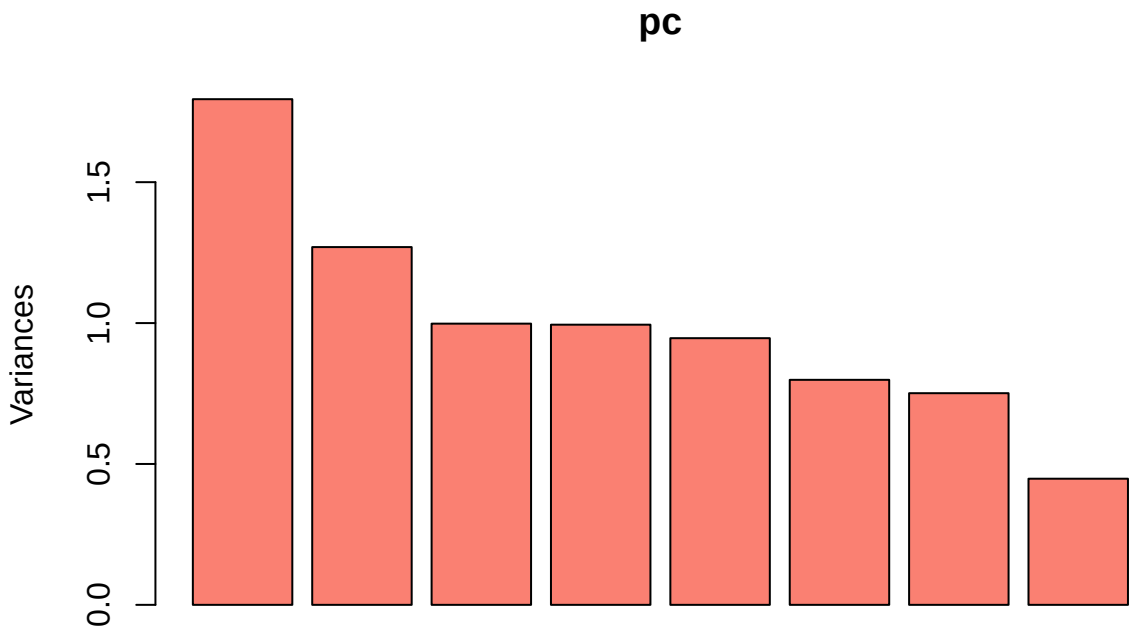
```
## 'data.frame':    1394 obs. of  10 variables:
##  $ nom_seq : chr  "ADT1_YEAST" "ADT2_YEAST" "ADT3_YEAST" "AAR2_YEAST" ...
##  $ mcg     : num  0.58 0.43 0.64 0.58 0.42 0.51 0.5 0.48 0.55 0.4 ...
##  $ gvh     : num  0.61 0.67 0.62 0.44 0.44 0.4 0.54 0.45 0.5 0.39 ...
##  $ alm     : num  0.47 0.48 0.49 0.57 0.48 0.56 0.48 0.59 0.66 0.6 ...
##  $ mit     : num  0.13 0.27 0.15 0.13 0.54 0.17 0.65 0.2 0.36 0.15 ...
##  $ erl     : num  0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
##  $ pox     : num  0 0 0 0 0 0.5 0 0 0 0 ...
##  $ vac     : num  0.48 0.53 0.53 0.54 0.48 0.49 0.53 0.58 0.49 0.58 ...
##  $ nuc     : num  0.22 0.22 0.22 0.22 0.22 0.22 0.22 0.34 0.22 0.3 ...
##  $ location: Factor w/ 4 levels "CYT","MEM","MIT",..: 3 3 3 4 3 1 3 4 3 1 ...
```

```r
summary(data_filtrat)
```

```
##    nom_seq               mcg              gvh              alm
##  Length:1394        Min.   :0.1100   Min.   :0.1300   Min.   :0.2100
##  Class :character   1st Qu.:0.4000   1st Qu.:0.4200   1st Qu.:0.4600
##  Mode  :character   Median :0.4800   Median :0.4800   Median :0.5100
##                     Mean   :0.4918   Mean   :0.4928   Mean   :0.5009
##                     3rd Qu.:0.5700   3rd Qu.:0.5600   3rd Qu.:0.5600
##                     Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       mit              erl              pox               vac
##  Min.   :0.0000   Min.   :0.5000   Min.   :0.000000   Min.   :0.0000
##  1st Qu.:0.1700   1st Qu.:0.5000   1st Qu.:0.000000   1st Qu.:0.4800
```

```
##  Median :0.2200   Median :0.5000   Median :0.000000   Median :0.5100
##  Mean   :0.2616   Mean   :0.5032   Mean   :0.001908   Mean   :0.5002
##  3rd Qu.:0.3200   3rd Qu.:0.5000   3rd Qu.:0.000000   3rd Qu.:0.5300
##  Max.   :1.0000   Max.   :1.0000   Max.   :0.830000   Max.   :0.7300
##       nuc          location
##  Min.   :0.0000   CYT:463
##  1st Qu.:0.2200   MEM:258
##  Median :0.2200   MIT:244
##  Mean   :0.2787   NUC:429
##  3rd Qu.:0.3100
##  Max.   :1.0000
```

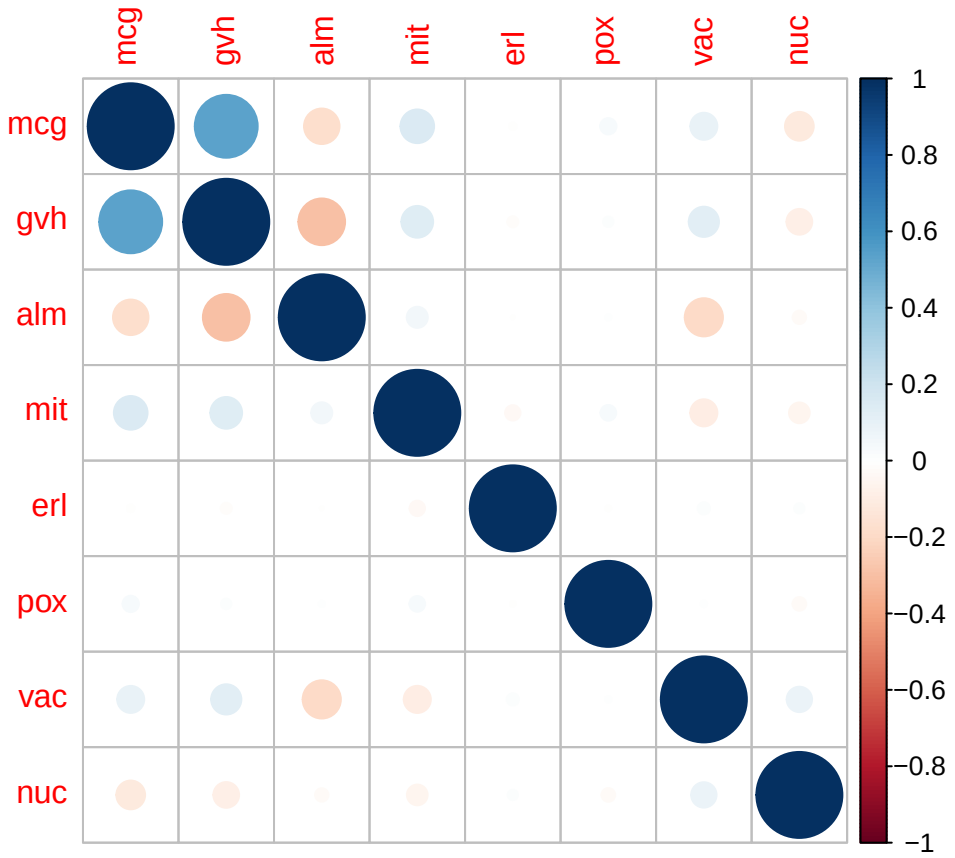
```
pc <- prcomp(data_filtrat[,-c(1,10)], scale = TRUE)
plot(pc, col="salmon")
```




```
summary(pc)
```

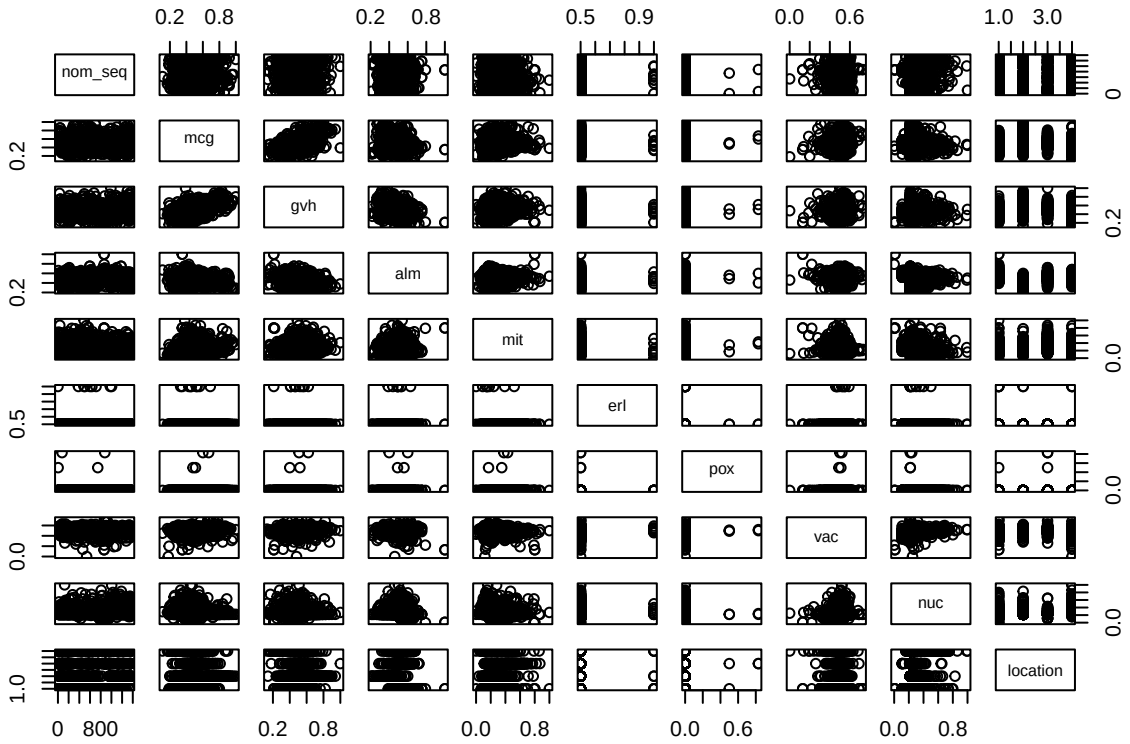
```
## Importance of components:
##                           PC1    PC2    PC3    PC4    PC5     PC6     PC7
## Standard deviation     1.3396 1.1268 0.9990 0.9971 0.9728 0.89362 0.86666
## Proportion of Variance 0.2243 0.1587 0.1247 0.1243 0.1183 0.09982 0.09389
## Cumulative Proportion  0.2243 0.3830 0.5078 0.6320 0.7503 0.85015 0.94404
##                           PC8
## Standard deviation     0.66907
## Proportion of Variance 0.05596
## Cumulative Proportion  1.00000
```

```
cor(data_filtrat[, -c(1,10)]) |> corrplot()
```

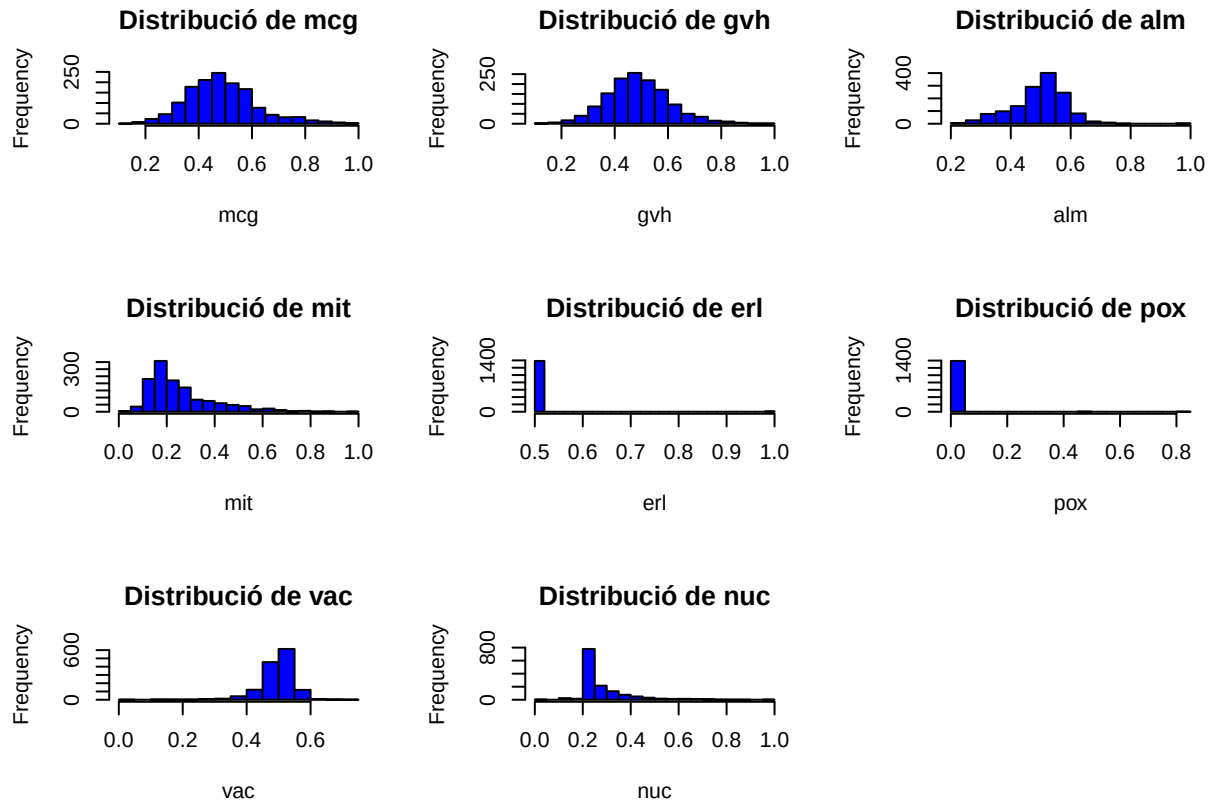


S'observa com no val la pena reduir la dimensionalitat perquè totes les variables expliquen bona part de la variabilitat. L'últim variable no s'ha pogut incloure perquè és de tipus string.

```
plot(data_filtrat)
```




```
vars <- names(data_filtrat)[2:9]
par(mfrow=c(3, 3))
for (var in vars) {
  hist(data_filtrat[[var]], col = "blue", breaks = 20,
       main = paste("Distribució de", var), xlab = var)
}
```



Train i Test

Tal com demana l'enunciat fixem la llavor aleatòria (123) i fixem que el 67% de les dades seran utilitzades com a training i l'altre 33% com a test.

```
p <- 2/3
indexs <- sample(nrow(data_filtrat), p * nrow(data_filtrat))
data_train <- data_filtrat[indexs, ]
data_test <- data_filtrat[-indexs,]
```

Algorismes

Utilitzant 3 fold-cross validation amb el paquet caret, explorar al k-NN els valors per al nombre de veïns k = 1, 11,21,31 i per al SVM explorar les funcions kernel lineal i rbf.

K-Nearest Neighbours (KNN)

S'observen els veïns d'acord amb la distància euclídea

```
# la variable 1 és el nom de la seq. i la 10 la resposta
knn_model_1 <- knn(
  train = data_train[,-c(1,10)],
  test = data_test[,-c(1,10)],
  cl = data_train[, 10],
  k = 3
  )

confusionMatrix(knn_model_1, data_test[, 10])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CYT MEM MIT NUC
##        CYT  98   6  22  48
##        MEM   8  55   8  10
##        MIT  18   5  30   9
##        NUC  50   6  12  80
##
## Overall Statistics
##
##                Accuracy : 0.5656
##                  95% CI : (0.5192, 0.6112)
##     No Information Rate : 0.3742
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.3897
##
##  Mcnemar's Test P-Value : 0.8277
##
## Statistics by Class:
##
##                      Class: CYT Class: MEM Class: MIT Class: NUC
## Sensitivity              0.5632     0.7639    0.41667     0.5442
## Specificity              0.7388     0.9338    0.91858     0.7862
## Pos Pred Value           0.5632     0.6790    0.48387     0.5405
## Neg Pred Value           0.7388     0.9557    0.89578     0.7886
## Prevalence               0.3742     0.1548    0.15484     0.3161
## Detection Rate           0.2108     0.1183    0.06452     0.1720
## Detection Prevalence     0.3742     0.1742    0.13333     0.3183
## Balanced Accuracy        0.6510     0.8489    0.66762     0.6652
```

Només un 59% d'accuracy, es pot millorar molt

Utilitzar les dades de training afegeix molt biaix, per això es fa una 3-fold cross validation:

```
control <- trainControl(method = "cv",    # Método de validación cruzada
                        number = 3,       # Número de pliegues
                        verboseIter = TRUE)

knn_model_cv <- train(location ~ .,     # Fórmula de la variable dependiente en función de las variable
                      data = data_train,              # Datos de entrenamiento
                      method = "knn",                 # Algoritmo k-NN
                      trControl = control,               # Control de entrenamiento con validación cruza
                      tuneGrid = expand.grid(k = c(1, 11, 21, 31)))
```

```
## + Fold1: k= 1
## - Fold1: k= 1
## + Fold1: k=11
## - Fold1: k=11
## + Fold1: k=21
## - Fold1: k=21
## + Fold1: k=31
## - Fold1: k=31
## + Fold2: k= 1
## - Fold2: k= 1
```

```
## + Fold2: k=11
## - Fold2: k=11
## + Fold2: k=21
## - Fold2: k=21
## + Fold2: k=31
## - Fold2: k=31
## + Fold3: k= 1
## - Fold3: k= 1
## + Fold3: k=11
## - Fold3: k=11
## + Fold3: k=21
## - Fold3: k=21
## + Fold3: k=31
## - Fold3: k=31
## Aggregating results
## Selecting tuning parameters
## Fitting k = 31 on full training set
```

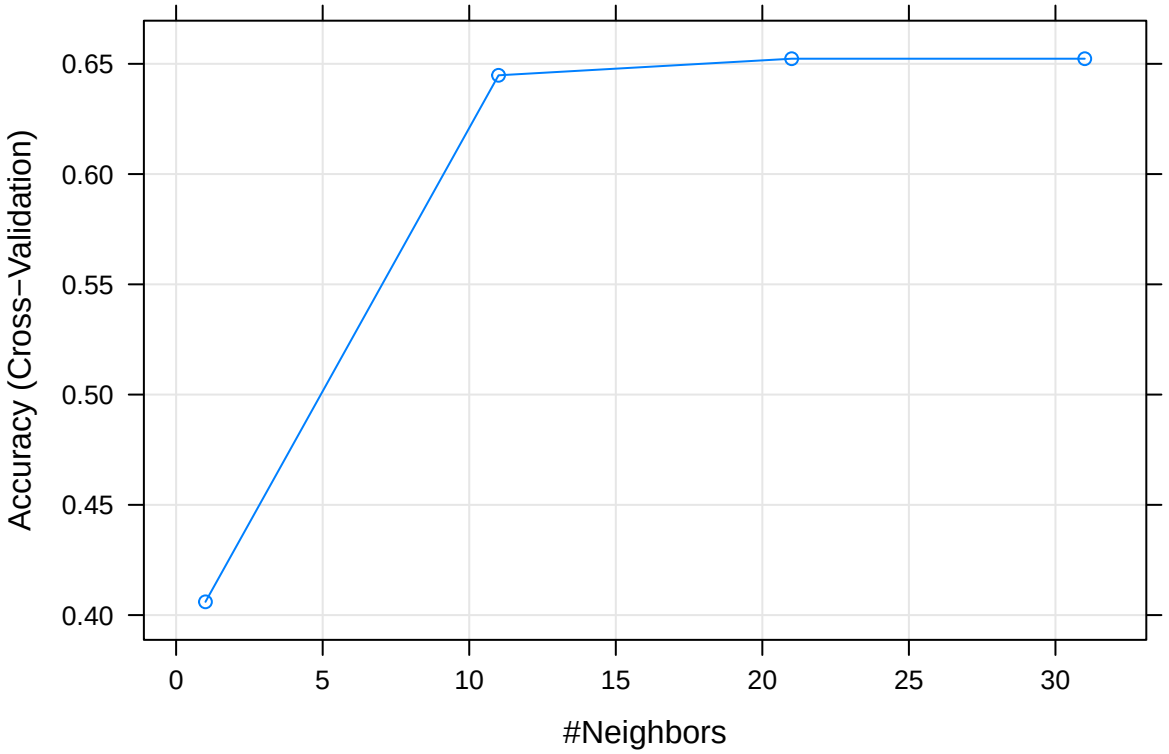
```
print(knn_model_cv)
```

```
## k-Nearest Neighbors
##
## 929 samples
##   9 predictor
##   4 classes: 'CYT', 'MEM', 'MIT', 'NUC'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 619, 619, 620
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.4060132  0.1491126
##   11  0.6447924  0.5146101
##   21  0.6523158  0.5251824
##   31  0.6523228  0.5249164
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 31.
```

```
print(knn_model_cv[["results"]])
```

```
##    k  Accuracy     Kappa  AccuracySD     KappaSD
## 1  1 0.4060132 0.1491126 0.161303385 0.254176559
## 2 11 0.6447924 0.5146101 0.014305663 0.019704557
## 3 21 0.6523158 0.5251824 0.006565488 0.009572118
## 4 31 0.6523228 0.5249164 0.009385752 0.013818020
```

```
plot(knn_model_cv)
```




```
# la k òptima és 21
```

Support Vector Machine (SVM)

```
svm_lineal <- ksvm(location ~ .,
                   data = data_train[, -1],
                   kernel = "vanilladot")
```

```
##  Setting default kernel parameters
```

```
svm_lineal_predict <- predict(svm_lineal, data_test)
conf_mat.lineal <- confusionMatrix(svm_lineal_predict, data_test[, 10])
conf_mat.lineal
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CYT MEM MIT NUC
##        CYT 119   5  25  66
##        MEM   3  61  10   9
##        MIT  18   1  37   7
##        NUC  34   5   0  65
##
## Overall Statistics
##
##                Accuracy : 0.6065
##                  95% CI : (0.5604, 0.6511)
##     No Information Rate : 0.3742
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                   Kappa : 0.4439
##
##  Mcnemar's Test P-Value : 0.0001226
##
## Statistics by Class:
##
##                      Class: CYT Class: MEM Class: MIT Class: NUC
## Sensitivity              0.6839     0.8472    0.51389     0.4422
## Specificity              0.6701     0.9440    0.93384     0.8774
## Pos Pred Value           0.5535     0.7349    0.58730     0.6250
## Neg Pred Value           0.7800     0.9712    0.91294     0.7729
## Prevalence               0.3742     0.1548    0.15484     0.3161
## Detection Rate           0.2559     0.1312    0.07957     0.1398
## Detection Prevalence     0.4624     0.1785    0.13548     0.2237
## Balanced Accuracy        0.6770     0.8956    0.72387     0.6598
```

```
svm_rbf <- ksvm(location ~ .,
                data = data_train[, -1], kernel = "rbf")
```

```
svm_rbf_predict <- predict(svm_rbf, data_test)
conf_mat.rbf <- confusionMatrix(svm_rbf_predict, data_test[,10])
conf_mat.rbf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CYT MEM MIT NUC
##        CYT 106   7  26  50
##        MEM   2  58   8  11
##        MIT  12   1  34   3
##        NUC  54   6   4  83
##
## Overall Statistics
##
##                Accuracy : 0.6043
##                  95% CI : (0.5582, 0.649)
##     No Information Rate : 0.3742
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4387
##
##  Mcnemar's Test P-Value : 0.01914
##
## Statistics by Class:
##
##                      Class: CYT Class: MEM Class: MIT Class: NUC
## Sensitivity              0.6092     0.8056    0.47222     0.5646
## Specificity              0.7148     0.9466    0.95929     0.7987
## Pos Pred Value           0.5608     0.7342    0.68000     0.5646
## Neg Pred Value           0.7536     0.9637    0.90843     0.7987
## Prevalence               0.3742     0.1548    0.15484     0.3161
## Detection Rate           0.2280     0.1247    0.07312     0.1785
## Detection Prevalence     0.4065     0.1699    0.10753     0.3161
## Balanced Accuracy        0.6620     0.8761    0.71575     0.6817
```

```
mysigma <- c(.0001, 0.001, 0.005, .01, .05, .1)
myC <- 1:5

param_space <- expand.grid(sigma = mysigma, C = myC)
model_sr <- caret::train(location ~ .,
                         data = data_train[, -1],
                         method='svmRadial',
                         trControl = trainControl(method = 'cv', number = 3),
                         tuneGrid = param_space,
                         trace = FALSE)
plot(model_sr)
```