PEC 4: Predicció de dolències cardíaques a partir dun electrocardiograma – Part en R

Vicent Caselles Ballester

2024-01-29

Contents

Introducció

Aquest informe, en l'intent de ser "dinàmic" com s'ha anat demanant al llarg de tota l'assignatura, està pensat per a què es pugui controlar el *flow* del codi des de aquest fitxer directament. És a dir, no és necessari obrir el fitxer `.ipynb`, ni cap dels fitxers amb codi d'`R`, per a que s'executi correctament[1].

Per exemple, per a fer la divisió dels *train* i *test subsets*, des d'aquest mateix *notebook* es crida la funció localitzada al directori `R_code` que s'encarrega de dur-ho a terme. Això ho dic ja que considero que el nivell d'abstracció és força alt. Tot depèn del detall al que volgueu arribar per a entendre com ho he fet, però si voleu arribar al màxim detall suposo que no us quedarà més remei que inspeccionar els diferents fitxers individualment amb un editor de text com `Rstudio`.

Aquest present fitxer `Markdown` executa de nou el codi escrit en `Python` que entrena les xarxes neuronals **de nou** i genera noves prediccions. Les matrius de confusió que en resulten és el que es guarda, i és després carregat al present fitxer per a analitzar-ho. Això té inconvenients, com la variablitat dels resultats. Tot i això, en les vàries vegades que ho he executat els resultats han sigut semblants, així que no hi hauria d'haver cap problema.

El *output* que es genera és el següent:

[1] Això no és del tot així. Preparar el fitxer `.ipynb` per a que pugui córrer a qualsevol ordinador no és un tema trivial. El problema es troba a la metadata de la *notebook*, que requereix que especifiquis el nom del `kernel` de `ipython` que correrà la *notebook*. Adjuntaré els resultats obtinguts amb les xarxes neuronals a l'entrega de la PEC, per a saltar-me aquest problema (el codi detecta els resultats i no corre la *notebook*). Si voleu córrer el fitxer `.ipynb`, heu d'obrir-la i canviar, a la metadata, els següents camps: `name`, `display_name`; allà heu de treure el nom que hi ha (que és el que correspon al meu `venv`), i ficar el nom del `kernel` o `venv` on tingueu instal · lat els *requirements* per a córrer les xarxes neuronals (i.e. `keras`, `numpy`, etc.; ho trobareu a `requirements.txt`).

- Un fitxer `.pdf` i un fitxer `.html` d'acord amb aquest present fitxer `.Rmd`.
- Un fitxer `.pdf` resultat de córrer el *Jupyter Notebook* (`.ipynb`). Aquest realment no cal que s'inspeccioni, ja que es carreguen aquí les matrius de confusió obtingudes a partir de les prediccions amb les dades *test* utilitzant les dues xarxes neuronals entrenades al present document, utilitzant-les per a valorar-ne la *performance*. Tot i això, recomano que s'obri per a entendre el codi `Python`.
- Un fitxer `.csv` amb les mètriques per classe de tots els algorismes. Aquests resultats no es mostren al present document però si que es comenten.

Exploració inicial del conjunt de dades

Anem a explorar una mica les dades. Encara que el conjunt de dades inicial té `NAs`, tot i així per a l'EDA (*Exploratory Data Analysis*) vull treballar amb aquest. Així doncs, veureu segurament variables que després no participaran en els algorismes que implementaré. Sé que això no és gaire lògic, però em sembla que és més enriquidor. Primer de tot, observem quina és la distribució de la variable a predir (Figura 1).

```
data <- read.csv(data, row.names=1)
```

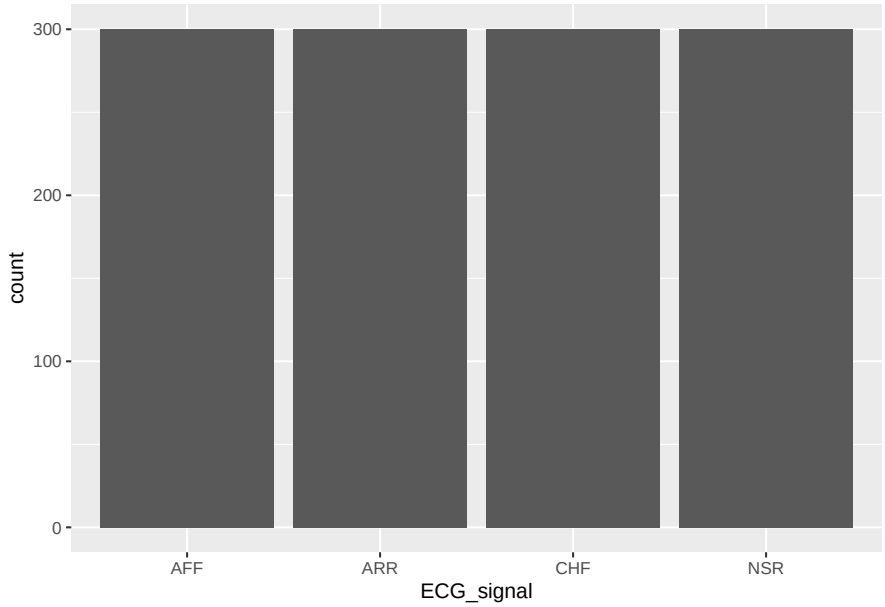

Figure 1: Distribució de la variable a predir (Resultat de l'ECG).

Com podem observar, el conjunt de dades és perfecte. 300 observacions per a cada una de les categories. Això és força extrany a la " " ""vida real"" " ".

Podem obtenir una petita visualització de totes les variables (la distribució que segueixen) amb el paquet `reshape2` i `ggplot2`. Es veu una mica apretat, i les variables amb més *outliers* dificulten la visualització de les altres, però ens permeten tenir una idea de la situació (Figura 2).

Només amb els noms de les variables, veiem que hi ha molta similaritat entre aquests. És a dir, sospito que hi haurà una forta covariança entre moltes de les variables del *dataset*. Possiblement hi hagi alguna variable que és *linear dependent* d'altres; és a dir, una variable y que sigui igual a una altra variable x més una constant (o una altra variable):

$$y = x + a; \forall a \in \mathbb{R}$$

Això ho podem observar en el següent gràfic, en el qual mostro *scatter plots* per a diferents combinacions

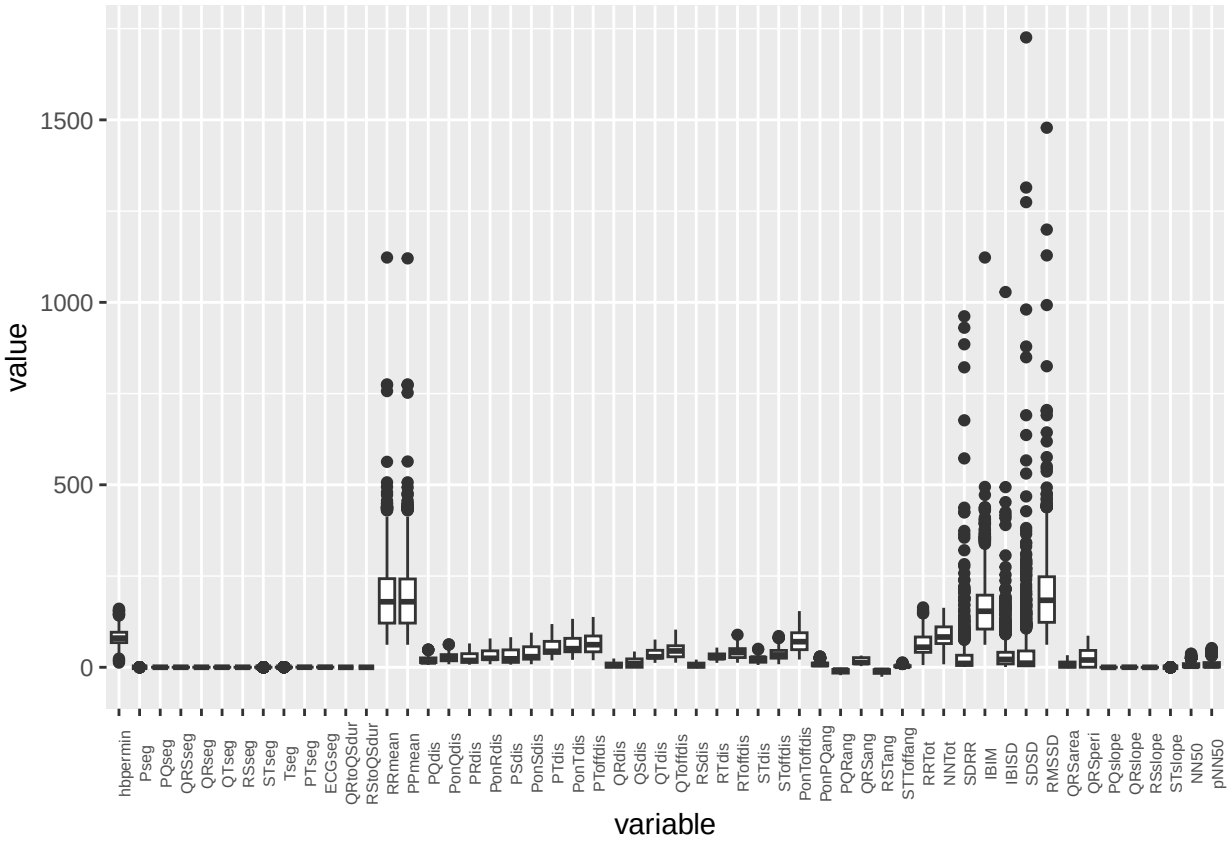

Figure 2: Distribució de les variables contínues que trobades al dataset.

de dos variables triades arbitràriament – més ben dit, amb els noms més o menys similars (sent això un indicador de potencial dependència lineal) (Figura 3).

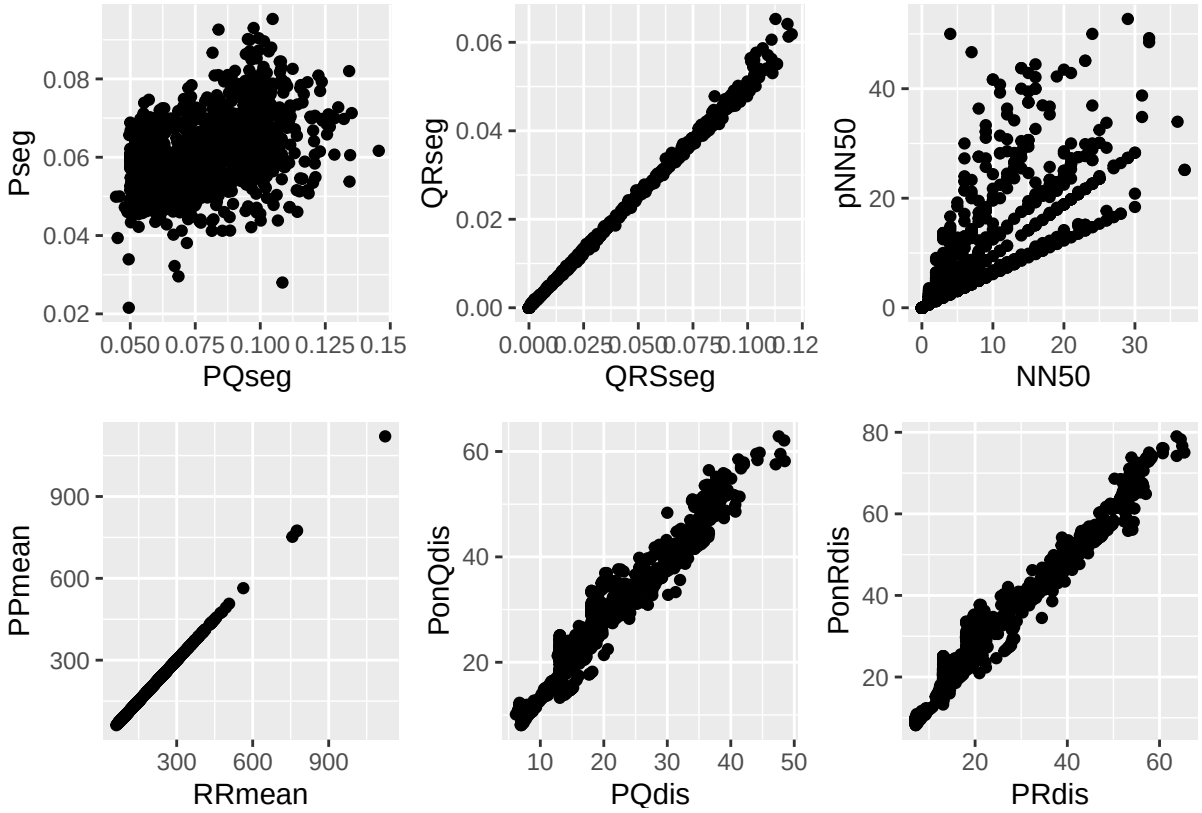

Figure 3: Scatter-plot d'algunes de les variables predictores presents al dataset (sense filtrar NAs).

Això ho podem observar, d'una manera més general i per a totes les variables simultàniament en un únic gràfic, mitjançant el que s'anomena *correlation heatmap* (Figura 4).

Això, per exemple, seria un problema si tinguessim la intenció de generar un model de regressió múltiple amb algunes d'aquestes variables predictores, ja que estaríem amb un cas de multicolinealitat (estudiat a regressió lineal).

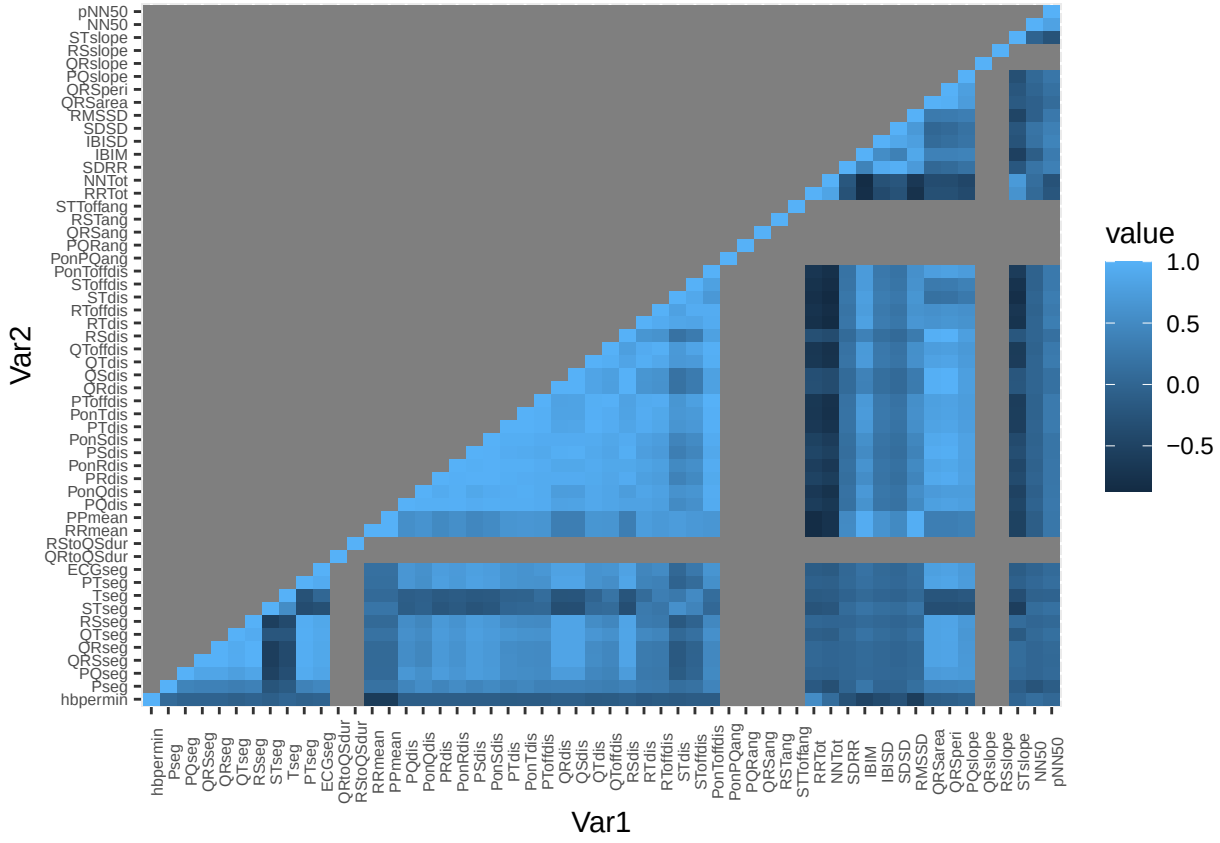

Figure 4: Correlation heatmap de totes les variables contingudes al conjunt de dades inicial (sense filtrar NAs).

Implementació dels diferents algorismes

Primer de tot, carregaré els conjunts de dades *train* i *test*.

```
train_data <- read.csv('train_data.csv', row.names=1)
test_data <- read.csv('test_data.csv', row.names=1)

## això em permetrà fer subsets amb només les dades predictores/classe a predir
whereisclass <- which(colnames(train_data)==class)
```

A continuació creo una llista on guardaré els resultats de tots els models. Els guardaré com a una llista de tipus `named` amb `confusionMatrices` com a elements.

```
cm_all <- list()
```

Algorisme k-NN

Per a la implementació d'aquest algorisme, utilitzaré les funcions que vaig crear per a la PEC 1 d'aquesta mateixa assignatura. Primer de tot preparem les dades en el format que el meu codi permet. El codi no és perfecte al final del dia.

```
## loading some custom functions
source('R_code/knn_implemented_vicent.R')
y_train <- train_data[, whereisclass]
x_train <- train_data[, -whereisclass]
x_test <- test_data[, -whereisclass]
y_test <- test_data[, whereisclass]
```

Itero per tots els valors de l'hierparàmetre `k` que he de provar i en guardo els resultats.

```
require(class)
ks_to_try <- params$ks_to_try
cm_knn <- list()
for (i in 1:length(ks_to_try)){
  k = ks_to_try[i]
  pred_test = predict_all_observations(x_test, x_train, train_labels=y_train,
                                       k=k, prob=F, seed=1234)

  ## aquí comprovo que els resultats són comparables amb els de knn implementat
  ## al paquet class, que em prenc com el ground truth (sé que estan bé)
  pred_class <- knn(train=x_train, test=x_test, cl=y_train,k=k)
  stopifnot(sum(pred_class == pred_test) / nrow(test_data) >= 0.99)

  name_for_list <- paste('knn_with_k', k, sep='_')
  cm_all[[name_for_list]] <- confusionMatrix(factor(pred_test),
                                             factor(y_test))
}
```

Mostro les matrius de confusió i mètriques generades en el següent *chunk*.

```
for (each in names(cm_all)[grep('knn', x=names(cm_all))]){
  cat(paste('Results for', each, '\n'))
  print(cm_all[[each]])
  cat('**************************************\n')
}
```

```
## Results for knn_with_k_1
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction AFF ARR CHF NSR
##        AFF  96   2   5   0
##        ARR   0 112   0   0
##        CHF   7   0  85   0
##        NSR   1   2   0  89
##
## Overall Statistics
##
##                Accuracy : 0.9574
##                  95% CI : (0.9327, 0.975)
##     No Information Rate : 0.2907
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.943
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: AFF Class: ARR Class: CHF Class: NSR
## Sensitivity              0.9231     0.9655     0.9444     1.0000
## Specificity              0.9763     1.0000     0.9773     0.9903
## Pos Pred Value           0.9320     1.0000     0.9239     0.9674
## Neg Pred Value           0.9730     0.9861     0.9837     1.0000
## Prevalence               0.2607     0.2907     0.2256     0.2231
## Detection Rate           0.2406     0.2807     0.2130     0.2231
## Detection Prevalence     0.2581     0.2807     0.2306     0.2306
## Balanced Accuracy        0.9497     0.9828     0.9609     0.9952
## ************************************
## Results for knn_with_k_3
## Confusion Matrix and Statistics
##
##           Reference
## Prediction AFF ARR CHF NSR
##        AFF  92   2   4   0
##        ARR   1 114   0   0
##        CHF  11   0  86   0
##        NSR   0   0   0  89
##
## Overall Statistics
##
##                Accuracy : 0.9549
##                  95% CI : (0.9296, 0.973)
##     No Information Rate : 0.2907
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9397
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
```

```
##                      Class: AFF Class: ARR Class: CHF Class: NSR
## Sensitivity              0.8846     0.9828     0.9556     1.0000
## Specificity              0.9797     0.9965     0.9644     1.0000
## Pos Pred Value           0.9388     0.9913     0.8866     1.0000
## Neg Pred Value           0.9601     0.9930     0.9868     1.0000
## Prevalence               0.2607     0.2907     0.2256     0.2231
## Detection Rate           0.2306     0.2857     0.2155     0.2231
## Detection Prevalence     0.2456     0.2882     0.2431     0.2231
## Balanced Accuracy        0.9321     0.9896     0.9600     1.0000
## ***************************************
## Results for knn_with_k_5
## Confusion Matrix and Statistics
##
##           Reference
## Prediction AFF ARR CHF NSR
##        AFF  87   2   3   0
##        ARR   1 113   0   0
##        CHF  16   1  87   0
##        NSR   0   0   0  89
##
## Overall Statistics
##
##                Accuracy : 0.9424
##                  95% CI : (0.9148, 0.9631)
##     No Information Rate : 0.2907
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.923
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: AFF Class: ARR Class: CHF Class: NSR
## Sensitivity              0.8365     0.9741     0.9667     1.0000
## Specificity              0.9831     0.9965     0.9450     1.0000
## Pos Pred Value           0.9457     0.9912     0.8365     1.0000
## Neg Pred Value           0.9446     0.9895     0.9898     1.0000
## Prevalence               0.2607     0.2907     0.2256     0.2231
## Detection Rate           0.2180     0.2832     0.2180     0.2231
## Detection Prevalence     0.2306     0.2857     0.2607     0.2231
## Balanced Accuracy        0.9098     0.9853     0.9558     1.0000
## ***************************************
## Results for knn_with_k_7
## Confusion Matrix and Statistics
##
##           Reference
## Prediction AFF ARR CHF NSR
##        AFF  89   0   4   0
##        ARR   1 115   0   0
##        CHF  14   1  86   0
##        NSR   0   0   0  89
##
## Overall Statistics
```

```
##
##                Accuracy : 0.9499
##                  95% CI : (0.9236, 0.9691)
##     No Information Rate : 0.2907
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.933
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: AFF Class: ARR Class: CHF Class: NSR
## Sensitivity              0.8558     0.9914     0.9556     1.0000
## Specificity              0.9864     0.9965     0.9515     1.0000
## Pos Pred Value           0.9570     0.9914     0.8515     1.0000
## Neg Pred Value           0.9510     0.9965     0.9866     1.0000
## Prevalence               0.2607     0.2907     0.2256     0.2231
## Detection Rate           0.2231     0.2882     0.2155     0.2231
## Detection Prevalence     0.2331     0.2907     0.2531     0.2231
## Balanced Accuracy        0.9211     0.9939     0.9535     1.0000
## ************************************
```

Veiem que generalment la *performance* de l'algorisme k-NN és bona. Tots els models prediuen perfectament la classe `NSR` (*Normal Synus Rhythm*). La classe que més li costa predir correctament és la classe `AFF` (en prediu algunes instàncies com a `CHF`). Curiosament, el valor de k que resulta en una millor *performance* global és 1 (és a dir, que només està jugant un paper el *nearest neighbor*). Aquest valor de `k` és el que porta a una millor predicció de les observacions que són realment `AFF`, però per contra és el que prediu pitjor `CHF`.

Naive Bayes

Implemento l'algoritme *Naive Bayes* utilitzant el paquet `e1071`. Tal i com s'indica a l'enunciat de la PEC, provem aplicant la transformació de Laplace i sense. Al final, la probabilitat de que demà no surti el sol mai és 0.

```
require(e1071)
naive_0 <- naiveBayes(x_train, y_train, laplace=0)
naive_1 <- naiveBayes(x_train, y_train, laplace=1)
```

Duc a terme les prediccions i guardo les `cm` a la llista global.

```
pred_naive_0 <- predict(naive_0, x_test)
pred_naive_1 <- predict(naive_1, x_test)
cm_all[['naive_bayes_no_laplace']] <- confusionMatrix(pred_naive_0, factor(y_test))
cm_all[['naive_bayes_w_laplace']] <- confusionMatrix(pred_naive_1, factor(y_test))
```

Mostro els resultats.

```
for (each in names(cm_all)[grep('naive_bayes', x=names(cm_all))]){
  cat(paste('Results for', each, '\n'))
  print(cm_all[[each]])
  cat('************************************\n')
}
```

```
## Results for naive_bayes_no_laplace
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction AFF ARR CHF NSR
##        AFF  89  10  16   0
##        ARR   5 106   0   0
##        CHF  10   0  74   0
##        NSR   0   0   0  89
##
## Overall Statistics
##
##                Accuracy : 0.8972
##                  95% CI : (0.8632, 0.9252)
##     No Information Rate : 0.2907
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8624
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: AFF Class: ARR Class: CHF Class: NSR
## Sensitivity              0.8558     0.9138     0.8222     1.0000
## Specificity              0.9119     0.9823     0.9676     1.0000
## Pos Pred Value           0.7739     0.9550     0.8810     1.0000
## Neg Pred Value           0.9472     0.9653     0.9492     1.0000
## Prevalence               0.2607     0.2907     0.2256     0.2231
## Detection Rate           0.2231     0.2657     0.1855     0.2231
## Detection Prevalence     0.2882     0.2782     0.2105     0.2231
## Balanced Accuracy        0.8838     0.9481     0.8949     1.0000
## ************************************
## Results for naive_bayes_w_laplace
## Confusion Matrix and Statistics
##
##           Reference
## Prediction AFF ARR CHF NSR
##        AFF  89  10  16   0
##        ARR   5 106   0   0
##        CHF  10   0  74   0
##        NSR   0   0   0  89
##
## Overall Statistics
##
##                Accuracy : 0.8972
##                  95% CI : (0.8632, 0.9252)
##     No Information Rate : 0.2907
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8624
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: AFF Class: ARR Class: CHF Class: NSR
```

```
## Sensitivity              0.8558     0.9138     0.8222     1.0000
## Specificity              0.9119     0.9823     0.9676     1.0000
## Pos Pred Value           0.7739     0.9550     0.8810     1.0000
## Neg Pred Value           0.9472     0.9653     0.9492     1.0000
## Prevalence               0.2607     0.2907     0.2256     0.2231
## Detection Rate           0.2231     0.2657     0.1855     0.2231
## Detection Prevalence     0.2882     0.2782     0.2105     0.2231
## Balanced Accuracy        0.8838     0.9481     0.8949     1.0000
## ************************************
```

Com podem veure, els models de tipus *Naive Bayes* mostren una pitjor *performance*. Això era esperable, ja que aquest tipus d'algorismes no estan pensats precisament per a predir classes amb conjunts de variables íntegrament contínues (ha de discretitzar-les – fer-ne *bins* – per a poder utilitzar-les).

Algorisme SVM

Per a construir els models SVM, torno a juntar en un `dataframe` les dades x (predictores) amb les dades y (classe a predir).

```
require(kernlab)
data_train_full <- data.frame(x_train, y_train)
data_train_full$y_train <- factor(data_train_full$y_train)
kllineal <- ksvm(y_train ~ ., data=data_train_full, kernel='vanilladot')
```

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

```
klrbf <- ksvm(y_train~., data=data_train_full, kernel='rbfdot')
```

Again, construeixo les prediccions i guardo les `confusionMatrices`.

```
pred_lineal <- predict(kllineal, x_test)
pred_rbf <- predict(klrbf, x_test)
cm_all[['svm_lineal']] <- confusionMatrix(pred_lineal, factor(y_test))
cm_all[['svm_rbf']] <- confusionMatrix(pred_rbf, factor(y_test))
```

Printejo les `confusions matrices`

```
for (each in names(cm_all)[grep('svm', x=names(cm_all))]){
  cat(paste('Results for', each, '\n'))
  print(cm_all[[each]])
  cat('************************************\n')
}
```

```
## Results for svm_lineal
## Confusion Matrix and Statistics
##
##           Reference
## Prediction AFF ARR CHF NSR
##        AFF  95   0   4   0
##        ARR   1 116   0   0
##        CHF   8   0  86   0
##        NSR   0   0   0  89
##
## Overall Statistics
##
##                Accuracy : 0.9674
##                  95% CI : (0.9449, 0.9825)
##     No Information Rate : 0.2907
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9564
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: AFF Class: ARR Class: CHF Class: NSR
## Sensitivity              0.9135     1.0000     0.9556     1.0000
## Specificity              0.9864     0.9965     0.9741     1.0000
## Pos Pred Value           0.9596     0.9915     0.9149     1.0000
## Neg Pred Value           0.9700     1.0000     0.9869     1.0000
## Prevalence               0.2607     0.2907     0.2256     0.2231
## Detection Rate           0.2381     0.2907     0.2155     0.2231
## Detection Prevalence     0.2481     0.2932     0.2356     0.2231
## Balanced Accuracy        0.9500     0.9982     0.9648     1.0000
## ***************************************
## Results for svm_rbf
## Confusion Matrix and Statistics
##
##           Reference
## Prediction AFF ARR CHF NSR
##        AFF  92   0   6   0
##        ARR   1 116   0   0
##        CHF  11   0  84   0
##        NSR   0   0   0  89
##
## Overall Statistics
##
##                Accuracy : 0.9549
##                  95% CI : (0.9296, 0.973)
##     No Information Rate : 0.2907
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9396
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: AFF Class: ARR Class: CHF Class: NSR
## Sensitivity              0.8846     1.0000     0.9333     1.0000
## Specificity              0.9797     0.9965     0.9644     1.0000
## Pos Pred Value           0.9388     0.9915     0.8842     1.0000
## Neg Pred Value           0.9601     1.0000     0.9803     1.0000
## Prevalence               0.2607     0.2907     0.2256     0.2231
## Detection Rate           0.2306     0.2907     0.2105     0.2231
## Detection Prevalence     0.2456     0.2932     0.2381     0.2231
## Balanced Accuracy        0.9321     0.9982     0.9489     1.0000
## ***************************************
```

Veiem que, en quant als algorismes SVM, el que dóna millors resultats és el lineal.

Decision tree

Per a dur a terme la creació del model d'arbre de decisió, utilitzo el paquet `C50`.

```
require(C50)

tree_no_boost <- C5.0(x_train, factor(y_train))
tree_boost <- C5.0(x_train, factor(y_train), trials=10)

pred_no_boost <- predict(tree_no_boost, x_test)
pred_boost <- predict(tree_boost, x_test)

cm_all[['tree_no_boost']] <- confusionMatrix(pred_no_boost, factor(y_test))
cm_all[['tree_boost']] <- confusionMatrix(pred_boost, factor(y_test))
```

```
for (each in names(cm_all)[grep('tree', x=names(cm_all))]){
  cat(paste('Results for', each, '\n'))
  print(cm_all[[each]])
  cat('**************************************\n')
}
```

```
## Results for tree_no_boost
## Confusion Matrix and Statistics
##
##           Reference
## Prediction AFF ARR CHF NSR
##        AFF  95   0   2   1
##        ARR   0 115   1   0
##        CHF   9   1  87   0
##        NSR   0   0   0  88
##
## Overall Statistics
##
##                Accuracy : 0.9649
##                  95% CI : (0.9418, 0.9807)
##     No Information Rate : 0.2907
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9531
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: AFF Class: ARR Class: CHF Class: NSR
## Sensitivity              0.9135     0.9914     0.9667     0.9888
## Specificity              0.9898     0.9965     0.9676     1.0000
## Pos Pred Value           0.9694     0.9914     0.8969     1.0000
## Neg Pred Value           0.9701     0.9965     0.9901     0.9968
## Prevalence               0.2607     0.2907     0.2256     0.2231
## Detection Rate           0.2381     0.2882     0.2180     0.2206
## Detection Prevalence     0.2456     0.2907     0.2431     0.2206
## Balanced Accuracy        0.9516     0.9939     0.9672     0.9944
## **************************************
## Results for tree_boost
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction AFF ARR CHF NSR
##        AFF  92   0   4   1
##        ARR   5 116   0   0
##        CHF   7   0  86   0
##        NSR   0   0   0  88
##
## Overall Statistics
##
##                Accuracy : 0.9574
##                  95% CI : (0.9327, 0.975)
##     No Information Rate : 0.2907
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9429
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: AFF Class: ARR Class: CHF Class: NSR
## Sensitivity              0.8846     1.0000     0.9556     0.9888
## Specificity              0.9831     0.9823     0.9773     1.0000
## Pos Pred Value           0.9485     0.9587     0.9247     1.0000
## Neg Pred Value           0.9603     1.0000     0.9869     0.9968
## Prevalence               0.2607     0.2907     0.2256     0.2231
## Detection Rate           0.2306     0.2907     0.2155     0.2206
## Detection Prevalence     0.2431     0.3033     0.2331     0.2206
## Balanced Accuracy        0.9338     0.9912     0.9665     0.9944
## ***************************************
```

Curiosament, el model sense *boosting* té una millor *accuracy* general.

Random Forest

```
require(randomForest)
rdm_for_100 <- randomForest(x_train, factor(y_train), ntree=100, mtry=sqrt(ncol(x_train)))

rdm_for_200 <- randomForest(x_train, factor(y_train), ntree=200, mtry=sqrt(ncol(x_train)))

pred_for_100 <- predict(rdm_for_100, x_test)
pred_for_200 <- predict(rdm_for_200, x_test)

cm_all[['random_forest_100']] <- confusionMatrix(pred_for_100, factor(y_test))
cm_all[['random_forest_200']] <- confusionMatrix(pred_for_200, factor(y_test))
```

```
for (each in names(cm_all)[grep('random_forest', x=names(cm_all))]){
  cat(paste('Results for', each, '\n'))
  print(cm_all[[each]])
  cat('***************************************\n')
}
```

```
## Results for random_forest_100
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction AFF ARR CHF NSR
##        AFF  97   0   1   0
##        ARR   0 116   0   0
##        CHF   7   0  89   0
##        NSR   0   0   0  89
##
## Overall Statistics
##
##                Accuracy : 0.9799
##                  95% CI : (0.9609, 0.9913)
##     No Information Rate : 0.2907
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9732
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: AFF Class: ARR Class: CHF Class: NSR
## Sensitivity              0.9327     1.0000     0.9889     1.0000
## Specificity              0.9966     1.0000     0.9773     1.0000
## Pos Pred Value           0.9898     1.0000     0.9271     1.0000
## Neg Pred Value           0.9767     1.0000     0.9967     1.0000
## Prevalence               0.2607     0.2907     0.2256     0.2231
## Detection Rate           0.2431     0.2907     0.2231     0.2231
## Detection Prevalence     0.2456     0.2907     0.2406     0.2231
## Balanced Accuracy        0.9647     1.0000     0.9831     1.0000
## ***************************************
## Results for random_forest_200
## Confusion Matrix and Statistics
##
##           Reference
## Prediction AFF ARR CHF NSR
##        AFF  98   0   1   0
##        ARR   0 116   0   0
##        CHF   6   0  89   0
##        NSR   0   0   0  89
##
## Overall Statistics
##
##                Accuracy : 0.9825
##                  95% CI : (0.9642, 0.9929)
##     No Information Rate : 0.2907
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9765
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
```

```
##
##                      Class: AFF Class: ARR Class: CHF Class: NSR
## Sensitivity              0.9423     1.0000     0.9889     1.0000
## Specificity              0.9966     1.0000     0.9806     1.0000
## Pos Pred Value           0.9899     1.0000     0.9368     1.0000
## Neg Pred Value           0.9800     1.0000     0.9967     1.0000
## Prevalence               0.2607     0.2907     0.2256     0.2231
## Detection Rate           0.2456     0.2907     0.2231     0.2231
## Detection Prevalence     0.2481     0.2907     0.2381     0.2231
## Balanced Accuracy        0.9695     1.0000     0.9847     1.0000
## ***************************************
```

Podem observar que els *random forests* (ambdós) prediuen les classes `ARR` i `NSR` de manera perfecta. Tenen potencial per a ser els millors models de tots.

Neural Network

Els models de *deep learning* generats amb `Python` i `Keras`, els avaluaré juntament amb els altres en aquest document. Per a fer això, el fitxer `.ipynb` que conté el codi `Python`, que entrena + prediu el conjunt *test*, guarda les matrius de confusió com a resum de la seva performance, que és el que carrego a continuació i analitzo aquí.

```
source('R_code/parse_cm.R')
cm_all[["neural_net_1_hid"]] <- parse_from_csv('confusion_matrices/cm_nn1.csv')
cm_all[["neural_net_2_hid"]] <- parse_from_csv('confusion_matrices/cm_nn2.csv')
```

Mostro els seus resultats.

```
for (each in names(cm_all)[grep('neural', x=names(cm_all))]){
  cat(paste('Results for', each, '\n'))
  print(cm_all[[each]])
  cat('***************************************\n')
}
```

```
## Results for neural_net_1_hid
## Confusion Matrix and Statistics
##
##     AFF ARR CHF NSR
## AFF  96   0   5   0
## ARR   0 116   0   0
## CHF   8   0  85   0
## NSR   0   0   0  89
##
## Overall Statistics
##
##                Accuracy : 0.9674
##                  95% CI : (0.9449, 0.9825)
##     No Information Rate : 0.2907
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9564
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
```

```
##                      Class: AFF Class: ARR Class: CHF Class: NSR
## Sensitivity              0.9231     1.0000     0.9444     1.0000
## Specificity              0.9831     1.0000     0.9741     1.0000
## Pos Pred Value           0.9505     1.0000     0.9140     1.0000
## Neg Pred Value           0.9732     1.0000     0.9837     1.0000
## Prevalence               0.2607     0.2907     0.2256     0.2231
## Detection Rate           0.2406     0.2907     0.2130     0.2231
## Detection Prevalence     0.2531     0.2907     0.2331     0.2231
## Balanced Accuracy        0.9531     1.0000     0.9593     1.0000
## ************************************
## Results for neural_net_2_hid
## Confusion Matrix and Statistics
##
##     AFF ARR CHF NSR
## AFF  95   0   2   0
## ARR   1 116   0   0
## CHF   8   0  88   0
## NSR   0   0   0  89
##
## Overall Statistics
##
##                Accuracy : 0.9724
##                  95% CI : (0.9512, 0.9862)
##     No Information Rate : 0.2907
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9631
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: AFF Class: ARR Class: CHF Class: NSR
## Sensitivity              0.9135     1.0000     0.9778     1.0000
## Specificity              0.9932     0.9965     0.9741     1.0000
## Pos Pred Value           0.9794     0.9915     0.9167     1.0000
## Neg Pred Value           0.9702     1.0000     0.9934     1.0000
## Prevalence               0.2607     0.2907     0.2256     0.2231
## Detection Rate           0.2381     0.2907     0.2206     0.2231
## Detection Prevalence     0.2431     0.2932     0.2406     0.2231
## Balanced Accuracy        0.9533     0.9982     0.9759     1.0000
## ************************************
```

Report final i conclusions

Inicialitzo el dataframe on guardaré totes les mètriques d'interés.

```
source('R_code/metrics.R')

## dataframe amb les mètriques per classe
dataframe_large_byclass = as.data.frame(round(cm_all[['knn_with_k_1']]$byClass, 3))
dataframe_large_byclass$algorithm <- rep(names(cm_all)[1], 4)

## dataframe amb les mètriques globals (1 fila per algoritme)
```

```
dataframe_large_overall = as.data.frame(t(cm_all[['knn_with_k_1']]$overall[1:2]))
dataframe_large_overall$f1score = f1_score(cm_all[['knn_with_k_1']])
dataframe_large_overall$recall = recall(cm_all[['knn_with_k_1']])
dataframe_large_overall$algorithm <- names(cm_all)[1]
```

Ara simplement itero per a totes les `confusionMatrices` que hi ha a la llista `cm_all`, i n'agafo les mètriques d'intrés (per classe i globals).

```
for (each in 2:length(cm_all)){
  we_are_doing = names(cm_all)[each]
  byclass = as.data.frame(round(cm_all[[we_are_doing]]$byClass, 3))
  byclass$algorithm <- rep(we_are_doing, 4)
  overall = as.data.frame(t(cm_all[[we_are_doing]]$overall[1:2]))
  overall$f1score <- f1_score(cm_all[[we_are_doing]])
  overall$recall <- recall(cm_all[[we_are_doing]])
  overall$algorithm <- we_are_doing

  dataframe_large_byclass = rbind(dataframe_large_byclass, byclass)
  dataframe_large_overall = rbind(dataframe_large_overall, overall)
}
```

Mostrem els resultats finals. També guardo els resultats per classe en un fitxer `csv` (a `results/results_by_class.csv`). He pensat maneres per a incloure d'alguna manera una visualització o taula amb aquests resultats, però finalment he decidit obviar-ho. Faré un petit comentari sobre els resultats globals a continuació (subjectes a una mica de variabilitat gràcies a les xarxes neuronals). Generalment, tots els models han mostrat una precisió gairebé perfecta a l'hora de predir la classe `NSR` (que podríem considerar com a la classe negativa, ja que entenc que correspon a no tenir cap aflicció). En canvi, a l'hora de discernir les tres patologies, aquí és on els models pateixen més. Especialment això es veritat per a la classe `AFF` (*Atrial Fibrillation*), seguida per la classe `CHF` (*Congestive heart failure*). La classe `ARR` (*Arrhythmia*) és, de les patologies, la més fàcil de predir correctament, destacant els models SVM i *random forest* que ambdós, en les seves dues configuracions cada un (lineal i amb *kernel* gaussià per als SVM; amb 100 i 200 arbres per als *RF*), prediuen correctament un 100% de les vegades.

Table 1: Resultats globals per als models provats en aquest informe.
F1-score computat com a "macro" average

Accuracy	Kappa	f1score	recall	algorithm
0.957	0.943	0.957	0.956	knn_with_k_1
0.955	0.940	0.954	0.954	knn_with_k_3
0.942	0.923	0.942	0.943	knn_with_k_5
0.950	0.933	0.949	0.950	knn_with_k_7
0.897	0.862	0.899	0.902	naive_bayes_no_laplace
0.897	0.862	0.899	0.902	naive_bayes_w_laplace
0.967	0.956	0.967	0.966	svm_lineal
0.955	0.940	0.954	0.954	svm_rbf
0.965	0.953	0.964	0.964	tree_no_boost
0.957	0.943	0.957	0.958	tree_boost
0.980	0.973	0.979	0.979	random_forest_100
0.982	0.977	0.982	0.982	random_forest_200
0.967	0.956	0.966	0.966	neural_net_1_hid
0.972	0.963	0.972	0.972	neural_net_2_hid

Com podem veure, els models que funcionen millor, en general, són el model *random forest* i les xarxes neuronals. Entre aquests dos, costa decidir quin és millor. En quant a quin és el pitjor, sense cap dubte,

són els models *naive bayes*. Aquests, com he comentat anteriorment, no són els millors per a classificació amb variables exclusivament númeriques i contínues. Jo tinc *bias* cap a les xarxes neuronals, ja que treballo amb elles i al final és el que més ho peta avui dia. Però sent el màxim d'*unbiased* que puc ser, diria que per números globals el millor model és el *random forest* amb 200 arbres.