Informe PEC 2

Predicción de interacción péptido-MHCI con Algoritmos ANN y SVM

José Félix Rojas Cabeza

20 de May de 2020

Contents

1. Explicación del funcionamiento y características de ANN y SVM.

Algoritmo Red Neuronal Artificial (ANN)

Las redes neuronales artificiales están inspiradas en las redes neuronales como las que tiene el cerebro. Las neuronas se sustituyen por nodos que reciben y envian señales (información). Se crea una red con diferentes nodos y capas interconectados para procesar la información. Cada capa esta formada por un grupo de nodos que transmite la información a los otros nodos de las capas siguientes.

Una red neuronal artificial se caracteriza por:

- La topología: Esto corresponde al número de capas y de nodos. Además de la dirección en que se la información pasa de un nodo al siguiente, dentro de capas o entre capas.
- La función de activación: Función que recibe un conjunto de entradas e integra la señales para transmitir la información a otro nodo.
- El algoritmo de entrenamiento: Establece la importancia de cada conexión para transmitir o no la señal a los nodos correspondientes. El más usado es el algoritmo "backpropagation". El nombre indica que para corregir los errores de predicción va hacia atras de la red corrigiendo los pesos de los nodos.

Table 1: Fortalezas y debilidades de algoritmo ANN

Fortalezas	Debilidades
Se puede adaptar a problemas de clasificación o predicción numérica	Extremadamente intenso computacionalmente.Lento para entrenar,particularmente si la topología de la red es compleja
Capaz de modelar patrones más complejos que casi cualquier algoritmo	Muy propenso a sobreajustar los datos de entrenamiento
Hace pocas suposiciones sobre las relaciones subyacentes de los datos	Resulta en un modelo de caja negra compleja que es difícil, si no imposible, de interpretar

Table 2: fortalezas y debilidades de algoritmo SVM

Fortalezas	Debilidades
Uso en predicción y clasificación	Requiere especificar parámetro C y función de kernel (ensayo y error)
Uso bastante extendido	Lento de entrenar, sobre todo a medida que aumenta el número de características
Funciona de forma óptima con ruido	El funcionamiento interno es difícil de interpretar, Como en el caso de ANN
Facilidad de uso, comparado con ANN	Requiere especificar parámetro C y función de kernel (ensayo y error)

Las fortalezas y debilidades de este algoritmo son:

Algoritmo Support Vector Machine (SVM)

Una "máquina de vectores de soporte" (SMV, por sus siglas en inglés), puede imaginarse como una superficie que crea un límite entre puntos de datos multidimensionales que representan ejemplos y los valores de sus características. La meta de un SVM es crear un límite plano, un **hiperplano**, que divida el espacio para crear particiones con bastante homogenidad en cada lado, combinando aspectos de kNN y modelos de regresión lineal. La combinación es muy poderosa, lo que permite a los SVM modelar relaciones muy complejas(Lantz 2013).

Las SVMs pueden adaptarse para ser utilizadas con prácticamente cualquier tipo de tarea de aprendizaje, incluyendo clasificación y predicción numérica. Muchos de los éxitos clave del algoritmo se han conseguido en reconocimiento de patrones. Las aplicaciones más notables incluyen(Lantz 2013):

- Clasificación de los datos de expresión génica de microarrays en el campo de la bioinformática para identificar el cáncer u otras enfermedades genéticas.
- Categorización de texto, como la identificación del idioma utilizado en un documento o la clasificación de documentos por tema.
- La detección de eventos raros pero importantes como fallas en el motor de combustión, brechas de seguridad o terremotos.

2. Leer los datos de peptidos.csv y hacer una breve descripción de ellos.

Incluir en esta descripción el patrón de cada clase de péptido mediante la representación de su secuencia logo (https://en.wikipedia.org/wiki/Sequence_logo). Para realizar esta representación se puede usar el paquete

ggseqlogo descargable desde github.

```
> peptides <- read.csv2(params$file_1)
> dim(peptides)
```

```
[1] 15840     2
```

Se decidió utilizar el esquema de colores “custom”.

```
> library(ggplot2)
> library(ggseqlogo)
>
> # Clases de péptido:
> table(peptides$label)
```

```
  NB   SB
7920 7920
```

```
> indices_n <- which(peptides$label=="NB")
> indices_s <- which(peptides$label=="SB")
>
> nb <- peptides$sequence[indices_n]
> sb <- peptides$sequence[indices_s]
>
> length(nb) # no interaction
```

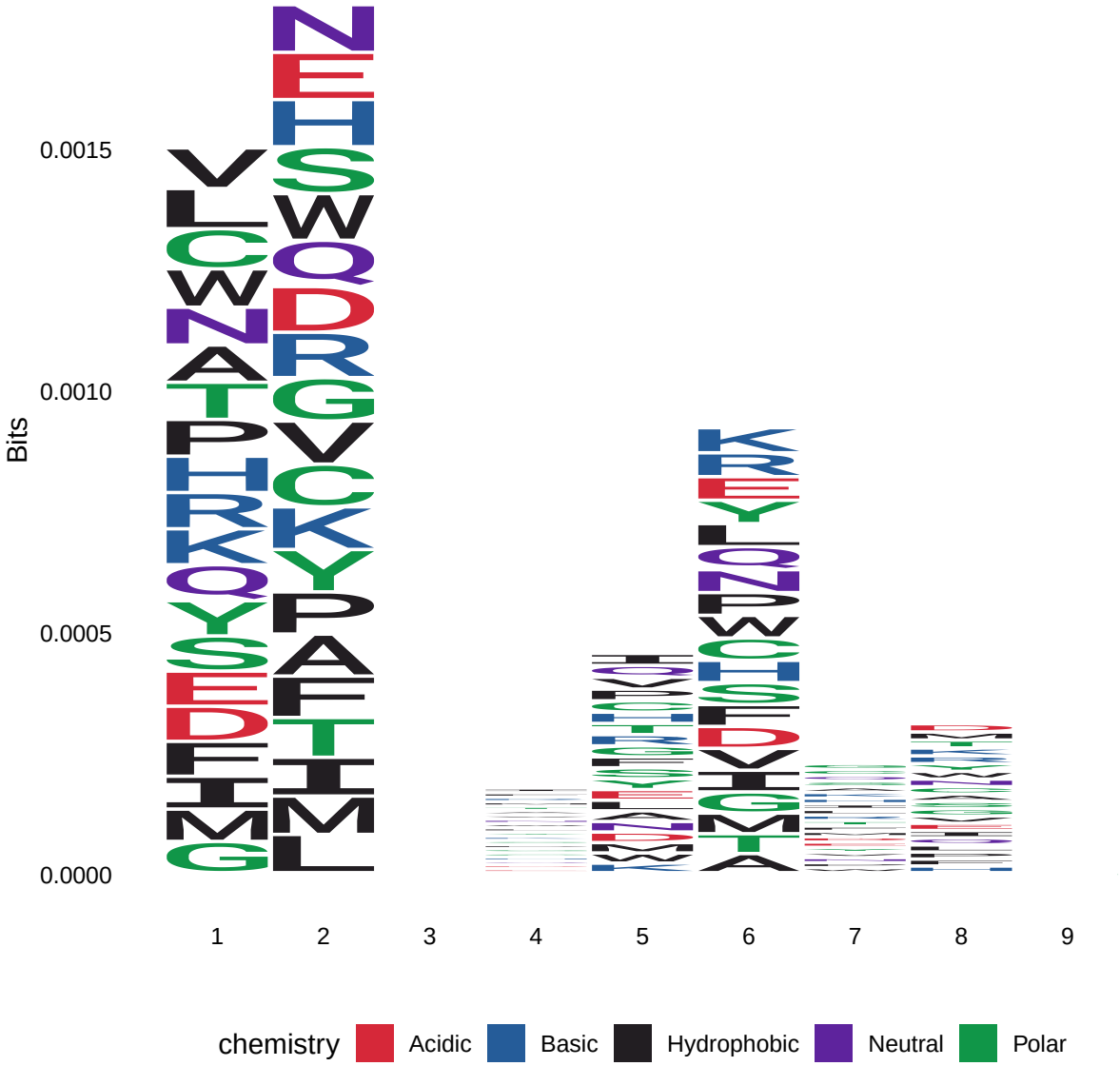
```
[1] 7920
```

```
> length(sb) # interaction
```

```
[1] 7920
```

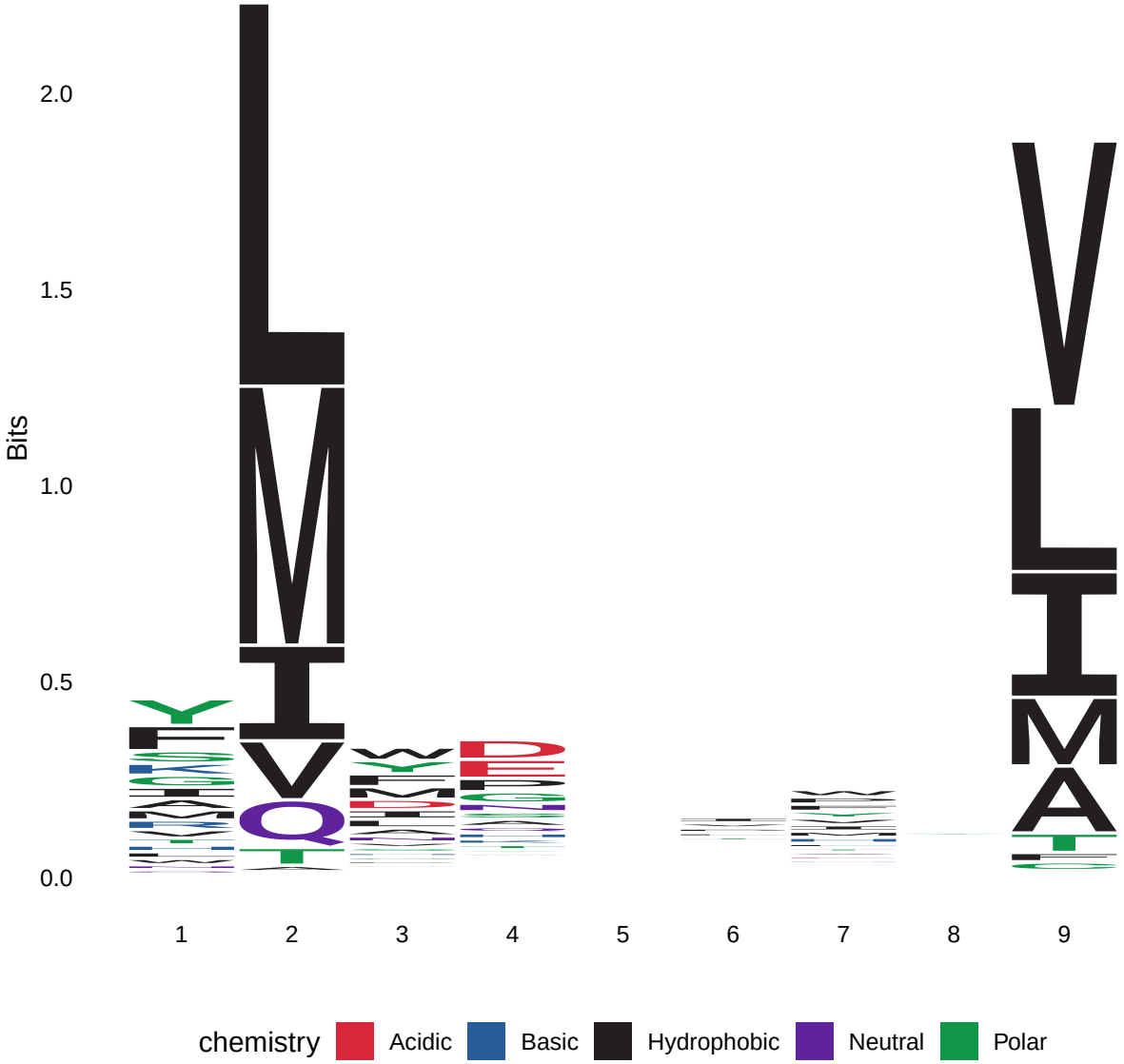
```
> library(ggseqlogo)
> # https://omarwagih.github.io/ggseqlogo/#accepted_input_formats
> g_l <- geom_logo(as.character(nb))
>
> ggplot() + g_l + theme_logo() + ggtitle("Figura 1.- Oligopéptidos tipo NB")
```

Figura 1.− Oligopéptidos tipo NB




```
> library(ggseqlogo)
> g_l_2 <- geom_logo(as.character(sb))
>
> ggplot() + g_l_2 + theme_logo() + ggtitle("Figura 2.- Oligopéptidos tipo SB")
```

Figura 2.− Oligopéptidos tipo SB



Se revsó en la literatura la clasificación de los aminoácidos(Nelson and Cox 2009): y coindcide con la clasificación por naturaleza bioquímica de los residuos. En los residuos tipo NB no se observa una tendencia clara, debido a que la frecuencia cambia en diferentes posiciones, puede ser que el péptido provenga de una región variable o poco seleccionada. En los residuos tipo SB se observa más homogeneidad, la posición 2 y 9 tienen residuos no polares (Leucina, Valina, Metionina, Alanina, Isoleucina) mientras que los otros residuos tienen frecuencias más variables. Llama la atención la presencia de algunos residuos polares en la primera parte del oligo.

3. Desarrollar una función en R que implemente la codificación “one-hot” (one-hot encoding) de las secuencias

```
> p_2_one_hot <- read.csv2(params$file_2)
> dim(p_2_one_hot)
```

```
[1] 15840   180
```

```
> aa_alphabet <- c("A","R","N","D","C","Q","E","G","H","I",
+                  "L","K","M","F","P","S","T","W","Y","V")
>
> one_hot<-function(sq, alphabet){
+ y<-unlist(strsplit(sq,""))
+ sapply(y,function(x){match(alphabet,x,nomatch=0)})
+ }
```

4. Transformar las secuencias de aminoácidos en vectores numéricos usando la función de transformación desarrollada en el punto anterior

```
> # Espacio reservado para hacer la conversión a vector.
>
> head(peptides)
```

```
   sequence label
1 LMAFYLYEV    SB
2 HQRLAPTMP    NB
3 FMNGHTHIA    SB
4 WLLIFHHCP    NB
5 MRYRVSVHP    NB
6 VLNGYSWFA    SB
```

```
> chr_seq <- as.character(peptides$sequence)
> o_h <- t(one_hot(chr_seq, aa_alphabet))
```

```
> # formato vectorial
> o_h[1:9,]
```

```
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
L    0    0    0    0    0    0    0    0    0     0     1     0     0     0
M    0    0    0    0    0    0    0    0    0     0     0     0     1     0
A    1    0    0    0    0    0    0    0    0     0     0     0     0     0
F    0    0    0    0    0    0    0    0    0     0     0     0     0     1
Y    0    0    0    0    0    0    0    0    0     0     0     0     0     0
L    0    0    0    0    0    0    0    0    0     0     1     0     0     0
Y    0    0    0    0    0    0    0    0    0     0     0     0     0     0
E    0    0    0    0    0    0    1    0    0     0     0     0     0     0
V    0    0    0    0    0    0    0    0    0     0     0     0     0     0
  [,15] [,16] [,17] [,18] [,19] [,20]
L     0     0     0     0     0     0
M     0     0     0     0     0     0
A     0     0     0     0     0     0
F     0     0     0     0     0     0
Y     0     0     0     0     1     0
L     0     0     0     0     0     0
Y     0     0     0     0     1     0
```

```
E     0     0     0     0     0     0
V     0     0     0     0     0     1
```

```
> A <- as.data.frame(o_h)
```

```
> o_h <- t(o_h)
> dim (o_h) <- prod(dim(o_h))
> o_h[1:180]
```

```
  [1] 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 [38] 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1
[112] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0
[149] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

```
> M <- matrix(o_h, ncol=180, byrow = T)
```

```
> # igualdad de M y p_2_one_hot
> table(p_2_one_hot == M)
```

```
   TRUE
2851200
```

5. Desarrollar un código en R que implemente un clasificador de red neuronal artificial. El código en R debe:

(a) Leer los datos transformados. En caso de no haber podido realizar la función se dispone del fichero peptidos_transf_one_hot.csv.

(b) Utilizando la semilla aleatoria 123, separar los datos en dos partes, una parte para training (67%) y una parte para test (33%).

(c) Antes de ejecutar cada uno de los modelos de clasificacion que se piden a continuación, poner como semilla generadora el valor 1234567.

(d) Crear dos modelos de red neuronal artificial de una sola capa oculta con 1 nodo y 3 nodos, respectivamente. Aplicar los datos de training para ajustar los modelos y posteriormente, predecir si la secuencia peptídica interacciona o no con MHCI en los datos del test.

(e) Comentar los resultados de la clasificación en función de los valores generales de la clasificación como "accuracy" y otros. Comparar los resultados de clasificación obtenidos para los diferentes valores de nodos usados en la capa oculta.

(f) Usar el paquete caret modelo 'mlp' para implementar la arquitectura de 3 nodos en la capa oculta, usando 5-fold crossvalidation. Comentar los resultados.

Dimensión de los datos

```
> dim(p_2_one_hot)
```

```
[1] 15840   180
```

Preparación de los datos

```
> set.seed(params$seed_train) #123
> # considerando que los datos están entre 0 y 1, inclusive, no deben normalizarse
```

```
> # creación de variables binarias
>
> #int <-  peptides$label=="SB"  # interaction
> #n_int <- peptides$label=="NB" # no interaction
>
> #int_index <- which(peptides$label=="SB")
> #n_int_index <- which(peptides$label=="NB")
>
> # creación de variables binarias
> SB <- peptides$label== "SB"
> NB <- peptides$label== "NB"
>
> data <- cbind(p_2_one_hot, SB, NB)
```

```
> # partición de los datos:
> n <- nrow(data)
> n_train <- 2/3
>
> # Separación de los datos
> train <- sample(n, round((n*n_train),0))
>
> data_train <- data[train,]
> data_test <- data[-train,]
```

Detalle de las últimas 5 columnas de los dos sets

```
> tail(data_train[,178:182],4)
```

```
      V178 V179 V180    SB    NB
1514     0    0    0  TRUE FALSE
8234     0    0    0  TRUE FALSE
14336    0    1    0 FALSE  TRUE
9906     0    0    0  TRUE FALSE
```

```
> tail(data_test[,178:182],4)
```

```
      V178 V179 V180    SB    NB
15828    0    0    0 FALSE  TRUE
15830    0    0    0  TRUE FALSE
15833    0    0    0 FALSE  TRUE
15838    0    0    1  TRUE FALSE
```

Se realiza una extracción de los datos aleatoriamente de 66.67% de todas las observaciones, 10560, para entrenar al modelo y del resto, 5280 para evaluarlo (test).

Entrenamiento del modelo en `data_train`

```
> set.seed(params$seed_nn_svm)
> # En caso de que no se tengan los paquetes, esto los solicita,
> # y si no están los instala
> if(!(require(neuralnet))) install.packages("neuralnet")
> if(!(require(NeuralNetTools))) install.packages("NeuralNetTools")
>
> library(neuralnet)
> library(NeuralNetTools)
>
```

```
> frmla = SB+NB ~ .
>
>  # una sóla neurona escondida
>  data_model_1 <- neuralnet(frmla, data=data_train, hidden=1, linear.output = F)
>
>  # tres neuronas escondidas
>  data_model_3 <- neuralnet(frmla, data=data_train, hidden=3, linear.output = F)
```

Evaluación del modelo

Luego de obtenidos los modelos, evaluamos su rendimiento con `data_test` utilizando la función `compute()`.

Se utilizó neuralnet (Fritsch et al. 2017)

1 nodo:

```
> library(neuralnet)
> model_result_1 <- compute(data_model_1, data_test[,1:180])
> mr_1_nr <- model_result_1$net.result
```

```
> #model_result_1_nr <- model_result_1$net.result
>
> max_idx <- function(X){return(which(X == max(X) ) )}
>
> # index
> idx_1 <- apply(mr_1_nr, 1, max_idx)
> pred_1 <- c("SB", "NB")[idx_1]
> res_1 <- table(pred_1, peptides$label[-train])
```

```
> if(!(require(caret))) install.packages("caret")
> set.seed(params$seed_nn_svm)
>
> library(caret)
> c_mat1 <- caret::confusionMatrix(res_1, positive="NB") # sin interacción
> c_mat1
```

```
Confusion Matrix and Statistics


pred_1   NB   SB
    NB 2654    1
    SB    5 2620

               Accuracy : 0.9989
                 95% CI : (0.9975, 0.9996)
    No Information Rate : 0.5036
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9977

 Mcnemar's Test P-Value : 0.2207

            Sensitivity : 0.9981
            Specificity : 0.9996
         Pos Pred Value : 0.9996
```

```
         Neg Pred Value : 0.9981
             Prevalence : 0.5036
         Detection Rate : 0.5027
   Detection Prevalence : 0.5028
      Balanced Accuracy : 0.9989

       'Positive' Class : NB
```

El modelo con 1 nodo oculto obtiene una precisión de 99.89%, y una sensitividad y especificidad de 99.81% y 99.96% respectivamente.

```
> plot(data_model_1, main= "Modelo con 1 nodo")
```

3 nodos:

```
> library(neuralnet)
> model_result_3 <- compute(data_model_3, data_test[,1:180])
> mr_3_nr <- model_result_3$net.result
```

```
> # index
> idx_3 <- apply(mr_3_nr, 1, max_idx)
> pred_3 <- c("SB", "NB")[idx_1]
> res_3 <- table(pred_3, peptides$label[-train])
```

```
> if(!(require(caret))) install.packages("caret")
> set.seed(params$seed_neuralnet)
> library(caret)
> c_mat3 <- caret::confusionMatrix(res_3, positive="NB") # sin interacción
> c_mat3
```

```
Confusion Matrix and Statistics


pred_3   NB   SB
    NB 2654    1
    SB    5 2620

               Accuracy : 0.9989
                 95% CI : (0.9975, 0.9996)
    No Information Rate : 0.5036
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9977

 Mcnemar's Test P-Value : 0.2207

            Sensitivity : 0.9981
            Specificity : 0.9996
         Pos Pred Value : 0.9996
         Neg Pred Value : 0.9981
             Prevalence : 0.5036
         Detection Rate : 0.5027
   Detection Prevalence : 0.5028
      Balanced Accuracy : 0.9989
```

```
       'Positive' Class : NB
```

El modelo con 3 nodos oculto obtiene una precisión de 99.89%, y una sensitividad y especificidad de 99.81% y 99.96% respectivamente. Es decir, no se observaron diferencias entre los desempeños de 3 nodos y un nodo.

```
> plot(data_model_3)
```

caret - 3 nodos, 5 fold crossvalidation

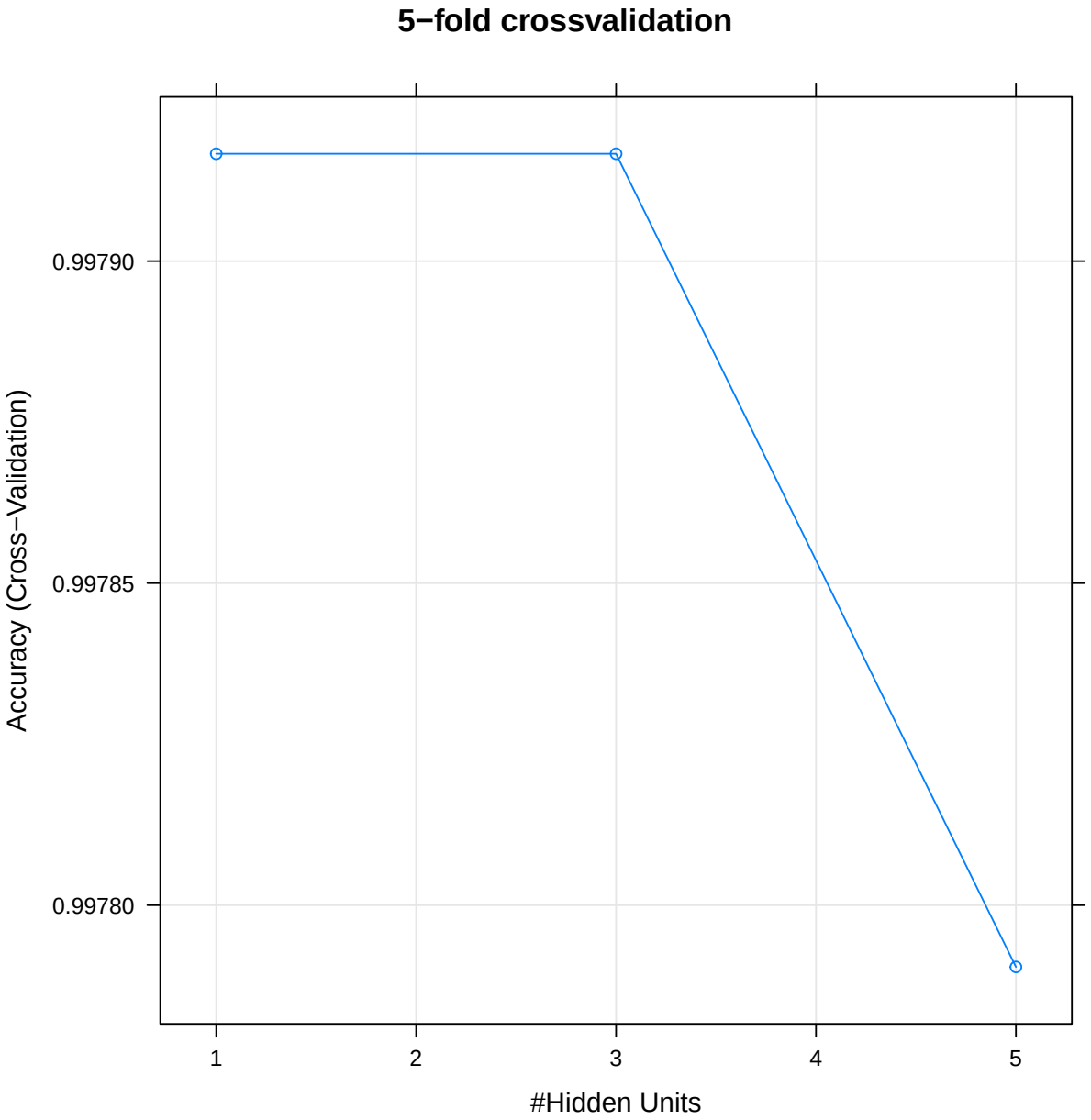
```
> set.seed(params$seed_nn_svm)
> library(caret)
> library(RSNNS)
>
> #p_2_one_hot # el dataset sin la categoría a predecir
> #peptides$label # el outcome recomendado por caret
> label_train <- peptides$label[train]
> label_test <- peptides$label[-train]
>
> sb_nb <- cbind(data$SB, data$NB) # positive class "NB"
> n_sb_nb <- cbind(as.numeric(data$SB), as.numeric(data$NB))
```

Se utilizó caret (“The caret Package” 2020), (Kuhn 2008):

```
> mlp_model <- caret::train(x=p_2_one_hot, y=peptides$label,
+                           method="mlp", size=3,
+                           trControl=trainControl(method="cv", number=5),
+                           trace=F)
```

```
> mlp_model

Multi-Layer Perceptron

15840 samples
  180 predictor
    2 classes: 'NB', 'SB'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 12672, 12672, 12672, 12672, 12672
Resampling results across tuning parameters:

  size  Accuracy   Kappa
  1     0.9979167  0.9958333
  3     0.9979167  0.9958333
  5     0.9977904  0.9955808

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was size = 1.
```

```
> plot(mlp_model, main= "5-fold crossvalidation")
```



En el modelo "mlp" con crossvalidation y 5 nodos, se observa menor exactitud (0.9958333) con 5 nodos.

6. Desarrollar un código en R que implemente un clasificador de SVM. El código en R debe:

(a) Leer los datos transformados. En caso de no haber podido realizar la función se dispone del fichero `peptidos_transf_one_hot.csv`.

(b) Utilizando la semilla aleatoria 123, separar los datos en dos partes, una parte para training (67%) y una parte para test (33%).

(c) Antes de ejecutar cada uno de los modelos de clasificacion que se piden a continuación, poner como semilla generadora el valor 1234567.

(d) Utilizar la función lineal y la RBF para ajustar un modelo de SVM basado en el training para predecir

si la secuencia peptídica interacciona o no con MHCI en los datos del test.

(e) Comentar los resultados de la clasificación en función de los valores generales de la clasificación como "accuracy" y otros. Comparar los resultados de clasificación obtenidos para los diferentes funciones kernel usadas.

(f) Usar el paquete caret modelo svmRBF para aplicar el algoritmo de SVM con 5-fold crossvalidation.

Comentar los resultados.

Datos:

Los datos a utilizar los mismos de la parte anterior:

```
> indices_nb <- which(data$NB)
> indices_sb <- which(data$SB)
>
> # No muestras t
> n_nb <- round(n_train*length(indices_nb),0)
>
> # No muestras t
> n_sb <- round(n_train*length(indices_sb),0)
>
> # aprox 66% de los datos
> set.seed(params$seed_train)
> rnd_nb <- sample(indices_nb, n_nb)
> rnd_sb <- sample(indices_sb, n_sb)
>
> # train
> #rnd_g <- sample(1:(ncol(data)-1), nrow(data))
>
> # Datasets
> data_nb <- data[rnd_nb, ]
> data_sb <- data[rnd_sb, ]
```

```
> label <- peptides$label
> data_2 <- cbind(p_2_one_hot, label)
```

```
> dim(data_2)
```

```
[1] 15840   181
```

Creación de datasets de entrenamiento (train) y prueba (test)

```
> n <- nrow(data_2)
> n_train <- params$ptrain
> set.seed(params$seed_train)
>
> train <- sample(n, floor(n*n_train))
>
> data_train_2 <- data_2[train,]
> data_test_2 <- data_2[-train,]
```

```
> head(data_train_2[,177:181],4)
```

```
      V177 V178 V179 V180 label
2463     1    0    0    0    SB
2511     0    0    0    0    SB
```

```
10419    0    0    0    1    SB
8718     0    0    0    0    NB
```

```
> head(data_test_2[,177:181],4)
```

```
   V177 V178 V179 V180 label
3     0    0    0    0    SB
4     0    0    0    0    NB
8     0    0    0    0    NB
14    0    0    1    0    NB
```

Entrenamiento del modelo en los datos: `kernlab::ksvm`

```
> if(!(require(kernlab))) install.packages("kernlab")
> library(kernlab)
>
> set.seed(params$seed_nn_svm)
>
> lmod <- ksvm(label ~ ., data=data_train_2, kernell="vanilladot")
>
> lmod
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.00291971510799512

Number of Support Vectors : 1289

Objective Function Value : -467.0374
Training error : 0.001989
```

Predicción y Evaluación del desempeño del modelo lineal

```
> pred_lmod <- predict(lmod, data_test_2)
```

```
> library(caret)
>
> # Crosstable
> ct <- table(pred_lmod, data_test_2$label)
> (cm_1 <- caret::confusionMatrix(ct, positive="NB"))
```

```
Confusion Matrix and Statistics


pred_lmod   NB   SB
       NB 2632    0
       SB   27 2621

               Accuracy : 0.9949
                 95% CI : (0.9926, 0.9966)
    No Information Rate : 0.5036
    P-Value [Acc > NIR] : < 2.2e-16
```

```
                 Kappa : 0.9898

 Mcnemar's Test P-Value : 5.624e-07

            Sensitivity : 0.9898
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9898
             Prevalence : 0.5036
         Detection Rate : 0.4985
   Detection Prevalence : 0.4985
      Balanced Accuracy : 0.9949

       'Positive' Class : NB
```

El modelo SVM lineal con categoría positiva "NB" tiene una precisión de 0.9948864, una sensibilidad de 0.9898458 y especificidad de 1.

Entrenamiento del modelo en los datos: `RSNNS::rbf`

Se utilizó RSNNS (Bergmeir and Benítez 2012)

```
> library(RSNNS)
> set.seed(params$seed_nn_svm)
>
> rbf_model <- rbf(x=data_train_2[,1:180], y=as.numeric(data_train_2$label), size = c(5), maxit=1000)
```

```
> pred_rbf <- predict( rbf_model, data_test_2[,1:180] )
```

```
> ct_rbf <- table(pred_lmod, data_test_2$label)
```

```
> (cm_rbf <- caret::confusionMatrix(ct_rbf, positive="NB"))
```

```
Confusion Matrix and Statistics


pred_lmod   NB   SB
       NB 2632    0
       SB   27 2621

               Accuracy : 0.9949
                 95% CI : (0.9926, 0.9966)
    No Information Rate : 0.5036
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9898

 Mcnemar's Test P-Value : 5.624e-07

            Sensitivity : 0.9898
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9898
             Prevalence : 0.5036
```

```
         Detection Rate : 0.4985
   Detection Prevalence : 0.4985
      Balanced Accuracy : 0.9949

       'Positive' Class : NB
```

El modelo RBF con categoría positiva "NB" tiene una precisión de 0.9948864, una sensibilidad de 0.9898458 y especificidad de 1. Es el mismo resultado que el del modelo anterior.

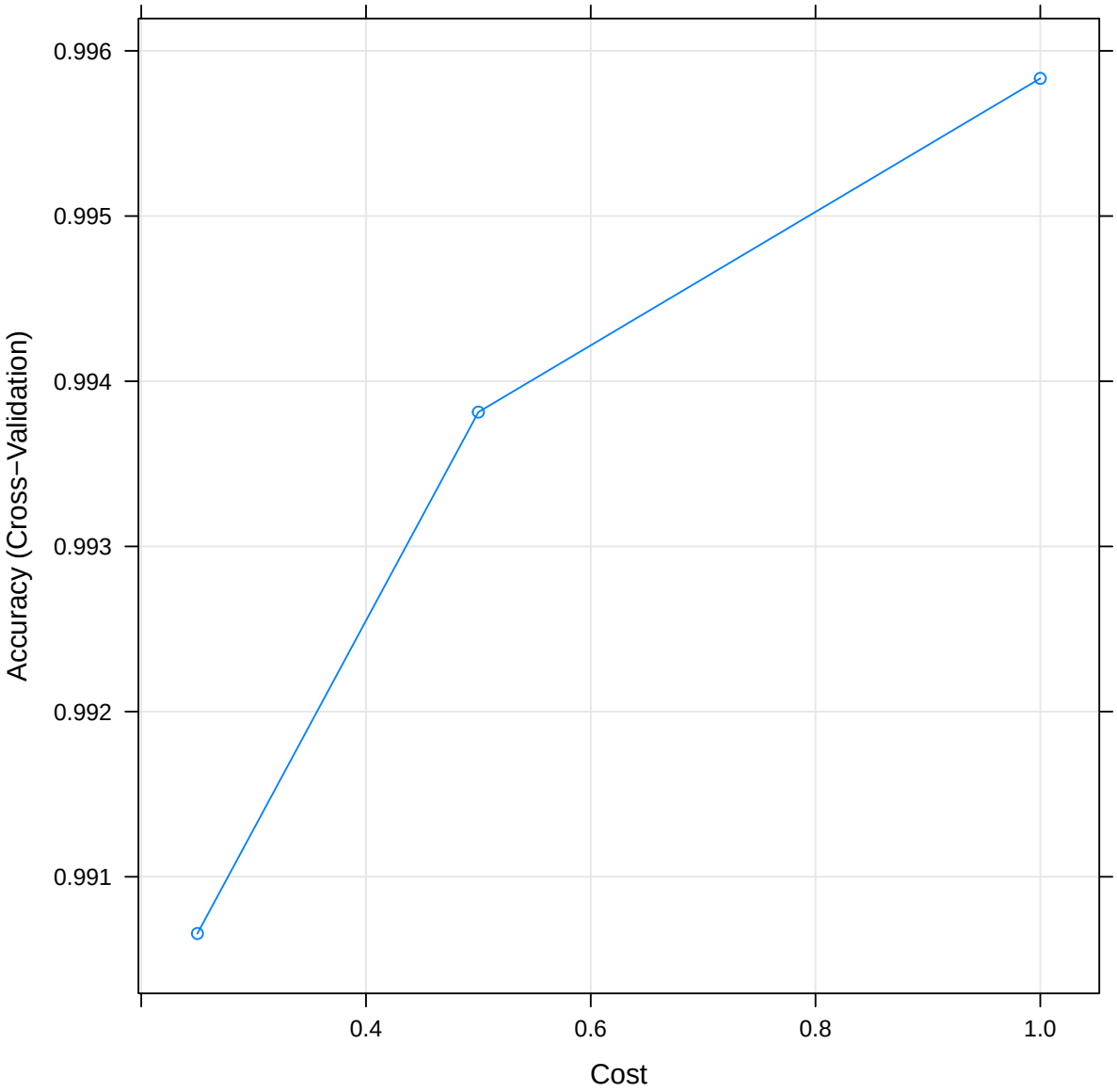
Entrenamiento del modelo en los datos: `caret` modelo `svmRBF` con 5-fold crossvalidation .

```
> library(caret)
> radial_mod <- caret::train(x=p_2_one_hot, y=peptides$label,method="svmRadial",
+                            size=3,trControl=trainControl(method="cv",
+                                                          number=5),trace=F)
```

```
> radial_mod
```

```
Support Vector Machines with Radial Basis Function Kernel

15840 samples
  180 predictor
    2 classes: 'NB', 'SB'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 12672, 12672, 12672, 12672, 12672
Resampling results across tuning parameters:

  C     Accuracy   Kappa
  0.25  0.9906566  0.9813131
  0.50  0.9938131  0.9876263
  1.00  0.9958333  0.9916667

Tuning parameter 'sigma' was held constant at a value of 0.002909547
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.002909547 and C = 1.
```

```
> plot(radial_mod)
```



En el modelo "svmRadial" con crossvalidation y 5 nodos, se observa la mayor exactitud (0.9938131) con un costo de 1.

Comentario de todos los modelos:

Ambos modelos se comportaron de manera muy precisa (exactitud > 0.99), a pesar de la varibilidad de condiciones. Sin embargo, es difícil entender cómo se llega a los resultados, debido a que los algoritmos utilizados ofuscan el proceso dentro de una "caja negra": sea un conjunto de capas ocultas o un hiperplano que divide de la mejor manera posible dos conjuntos de puntos. Además, la matemática detrás de los cálculos es avanzada (Lantz 2013). Por suerte, la aplicación es menos complicada que la matemática subyacente.

En este caso, los modelos sencillos (1 nodo y lineal) se desempeñaron bastante bien. Aunque utilizando SVM, el método que utiliza *five-fold crossvalidation* da mejores resultados, están suficientemente cerca del modelo simple para afirmar que es suficiente con utilizar el modelo lineal.

Table 3: Resultados obtenidos con los diferentes modelos

Algoritmo	Modelo	Train:Test	Exactitud	Kappa
ANN	1 nodo	2:1 (66%:33%)	0.9989	0.9977
ANN	3 nodos	2:1 (66%:33%)	0.9989	0.9977
ANN	MLP, 3 nodos	CV 5x	0.9979	0.9958
SVM	lineal	2:1 (66%:33%)	0.9949	0.9898
SVM	Gausiano (RBF)	2:1 (66%:33%)	0.9949	0.9898
SVM	Gausiano (RBF)	CV 5x	0.9957	0.9914

Parece que la codificación "one hot" es bastante útil para aumentar la exactitud del modelo, posiblemente porque la ortonormalización del sistema implícita en el uso de one hot facilita la separación de las categorías.

Referencias

Bergmeir, Christoph, and José M. Benítez. 2012. "Neural Networks in R Using the Stuttgart Neural Network Simulator: RSNNS." *Journal of Statistical Software* 46 (7): 1–26. http://www.jstatsoft.org/v46/i07/.

Fritsch, Stefan, Frauke Guenther, Marvin N. (creator) Wright, Marc Suling, and Sebastian M. Mueller. 2017. "Neuralnet Package of R." https://journal.r-project.org/archive/2010/RJ-2010-006/RJ-2010-006.pdf.

Kuhn, Max. 2008. "Building Predictive Models in R Using the Caret Package." *Journal of Statistical Software, Articles* 28 (5): 1–26. https://doi.org/10.18637/jss.v028.i05.

Lantz, Brett. 2013. *Machine Learning with R*. Packt Publishing Ltd.

Nelson, David L, and Michael M Cox. 2009. *Lehninger Principles of Biochemistry*. WH Freeman New York.

"The caret Package." 2020. http://topepo.github.io/caret/train-models-by-tag.html.