Predicción de interacción entre péptido y el complejo mayor de histocompatibilidad tipo I

Marc Bañuls Tornero

25/10/2019

Contents

1.1. Algoritmo Red Neuronal Artificial

El algoritmo de Redes Neuronales Artificiales se un modelo que basa las predicciones a partir de los inputs de un trainng Dataset. Este modelo recoge todos los inputs del Dataset a estudiar y les atribuye una importancia o peso en el resultado de la clasificación. Todos los inputs valorados por su peso se suman en un nodo y si llegan a un límite umbral, activan su función. De esta manera según la cantidad de nodos de la red neuronal y la cantidad de inputs, se establece un peso a todos los inputs y se puede predecir posteriormente la probabilidad de que un valor sea positivo o negativo, según la cantidad de señal que reciba el nodo de output.

Fortalezas	Debilidades
· Puede adaptarse a la clasificación o predicción de problemas numéricos	· Muy lento de entrenar y extremadamente intensivo computacionalmente
· Puede producir modelos a partir de casi cualquier algoritmo y de patrones complejos	· Es muy facil que realice un overfitting de training data
· Realiza pocas suposiciones de las relaciones de los datos	· Crea un modelo de caja negra que resulta imposible de interpretar posteriormente

1.2. Algoritmo Support Vector Machine

Los Support Vector Machines o SVM crean un hiperplano que divide el espacio para crear particiones de los datos de manera homogénea según su clasificación. El algoritmo de SVM separa estas particiones mediante líneas en el hiperplano, buscando la llamada línea de máximo margen hiperplano (MMH) la cual resulta en la mejor separación de dos clases. Para ello, el algoritmo se basa en los valores de cada clase que se encuentran más cerca del MMH.
Los SVM tienen un alto rendimiento a la hora de separar o clasificar datos linealmente separables. En el caso de tratar con datos que no pueden ser separados linealmente, se utilizan variables "de holgura", que permite que algunos datos se clasifiquen incorrectamente (aunque estos penalicen el modelo). También, en casos de datos que no pueden separarse linealmente, se utilizan kernels. Éstos tienen como función la extrapolación de los datos a mayores niveles de dimensionalidad, permitiendo que estas características puedan llegar a ser linealmente separables según las dimensiones en que se interpreten las características.

Los SVM tienen varias fortalezas y debilidades:

Fortalezas	Debilidades
· Puede adaptarse a la clasificación o predicción de problemas numéricos	· Encontrar el mejor modelo requiere de probar varias combinaciones de kernels y parámetros de modelos
· Los datos con ruido tienen poca influencia en el modelo y éste no es propenso a sobreajustar	· Puede ser lento para entrenar, sobretodo si los datos de entrada tienen muchas características o ejemplos
· Está ganando popularidad debido a su alta precisión y un alto ratio de victorias en competiciones de data-mining	· Crea un modelo de caja negra que resulta imposible de interpretar posteriormente

2. Desarrollar una función en R que implemente la codificación "one-hot" (*one-hot encoding*) de las secuencias.

Podemos utilizar una función similar a la realizada en la PEC 1, ya que también se basaba en la codificación one-hot de aminoácidos.

```
onehot_f<-function(x){
  x<-as.data.frame(lapply(x, function(y) gsub("A","10000000000000000000",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("R","01000000000000000000",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("N","00100000000000000000",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("D","00010000000000000000",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("C","00001000000000000000",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("Q","00000100000000000000",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("E","00000010000000000000",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("G","00000001000000000000",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("H","00000000100000000000",y)),stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("I","00000000010000000000",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("L","00000000001000000000",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("K","00000000000100000000",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("M","00000000000010000000",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("F","00000000000001000000",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("P","00000000000000100000",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("S","00000000000000010000",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("T","00000000000000001000",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("W","00000000000000000100",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("Y","00000000000000000010",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("V","00000000000000000001",y)), stringsAsFactors = FALSE)

  x<-as.data.frame(lapply(x, function(y) gsub("1","1 ",y)), stringsAsFactors = FALSE)
  x<-as.data.frame(lapply(x, function(y) gsub("0","0 ",y)), stringsAsFactors = FALSE)

  x<-cSplit(x, names(x), " ")
  x<-lapply(x,as.numeric)
  peptidos<-as.data.frame(x)

  return(peptidos)
}
```

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

(a) Leer el fichero data4.csv donde cada registro contiene una secuencia de 17 aminoácidos y la clase de estructura secundaria correspondiente al aminoácido central (posición 9), donde los caracteres 'h', 'e' y '-' representan α-helix, β-sheet y coil, respectivamente.

Leemos el fichero **data4.csv** y observamos como se encuentran distribuidos los datos:

```
datos<-read.delim("data4.csv", header = TRUE, sep = ";")
str(datos)
```

```
## 'data.frame':    10000 obs. of  18 variables:
##  $ V1 : Factor w/ 20 levels "A","C","D","E",..: 16 10 10 10 17 6 6 10 16 16 ...
##  $ V2 : Factor w/ 20 levels "A","C","D","E",..: 16 10 10 10 18 14 7 11 1 6 ...
```

```
##  $ V3 : Factor w/ 20 levels "A","C","D","E",..: 13 14 12 12 16 20 1 18 3 3 ...
##  $ V4 : Factor w/ 20 levels "A","C","D","E",..: 5 20 13 1 5 1 5 3 3 16 ...
##  $ V5 : Factor w/ 20 levels "A","C","D","E",..: 16 20 9 9 4 18 12 12 18 20 ...
##  $ V6 : Factor w/ 20 levels "A","C","D","E",..: 14 6 9 6 1 7 18 19 9 17 ...
##  $ V7 : Factor w/ 20 levels "A","C","D","E",..: 9 11 20 4 17 6 4 15 9 10 ...
##  $ V8 : Factor w/ 20 levels "A","C","D","E",..: 6 17 8 17 5 18 5 13 1 16 ...
##  $ V9 : Factor w/ 20 levels "A","C","D","E",..: 3 4 13 5 1 17 3 1 5 10 ...
##  $ V10: Factor w/ 20 levels "A","C","D","E",..: 4 11 6 4 5 16 3 14 1 16 ...
##  $ V11: Factor w/ 20 levels "A","C","D","E",..: 15 12 17 18 10 5 16 13 8 6 ...
##  $ V12: Factor w/ 20 levels "A","C","D","E",..: 20 20 9 1 8 18 4 10 8 1 ...
##  $ V13: Factor w/ 20 levels "A","C","D","E",..: 9 20 11 10 9 16 3 9 3 16 ...
##  $ V14: Factor w/ 20 levels "A","C","D","E",..: 20 17 18 16 16 15 9 12 14 20 ...
##  $ V15: Factor w/ 20 levels "A","C","D","E",..: 18 18 5 12 13 10 1 15 3 17 ...
##  $ V16: Factor w/ 20 levels "A","C","D","E",..: 3 10 13 9 3 6 18 14 9 1 ...
##  $ V17: Factor w/ 20 levels "A","C","D","E",..: 13 5 6 6 16 2 10 8 16 6 ...
##  $ V18: Factor w/ 3 levels "_","e","h": 1 1 1 2 2 2 1 1 3 1 ...
```

Tenemos 16 columnas que indican qué aminoácido se encuentra en qué posición de la proteína. Además, nos encontramos con la columna 18 que contiene la clase de aminoácido que se encuentra en la posición central de la proteína, es decir, su estructura secundaria (α-helix, β-sheet o coil). Cabe mencionar que tenemos 10000 proteínas, una cantidad suficiente para poder obtener consistentes resultados a partir de un test dataset y un training dataset.
Antes de avanzar al siguiente apartado debemos procesar los datos para que se encuentren en codificación "one-hot". Para ello, debemos separar de la función las etiquetas que se encuentran en la columna 18 (el tipo de estructura secundaria del aminoácido 9 de la proteína).

```
peptidos_ann<-datos[,1:17]
```

Utilizamos la función creada previamente en el apartado 2 para realizar la codificación onehot de los aminoácidos. Utilizando esta función obtendremos 10000 filas (todos los registros) y 340 columnas (1 columna para cada número de la codificación onehot) todas en formato numérico, formato necesario para el entrenamiento posterior.

```
onehot_ann<-onehot_f(peptidos_ann)
```

Ahora que hemos realizado correctamente la codificación onehot de los datos, unimos las etiquetas para tener el dataset completo pero antes debemos modificar su formato. Como queremos realizar una clasificación entre las tres estructuras secundarias, podemos crear 3 variables en la tabla, una para cada estructura secundaria. Éstas indicarán con TRUE si la estructura secundaria de la proteína en el aminoácido 9 es igual a su variable y FALSE si no se da el caso.
En el estudio de los datos se podía observar que la columna que indicaba esto se encuentra en formato factor, dando a cada estructura secundaria el valor de un número. La estructura de coil se clasifica como "_" la de β-sheet como "e" y la de α-helix como "h". Podemos utilizar estos valores para obtener las 3 variables con valores TRUE o FALSE según su estructura secundaria:

```
onehot_ann$giro<- datos$V18=="_"
onehot_ann$tira<- datos$V18=="e"
onehot_ann$helice<- datos$V18=="h"
```

(b) Utilizando la semilla aleatoria 12345, separar los datos en dos partes, una parte para training (67%) y una parte para test (33%).

Hacemos la selección de datos, 67% para train dataset y 33% para el test dataset:

```
set.seed(12345)
recogida_datos_onehot_ann<-sample(1:nrow(onehot_ann), nrow(onehot_ann)*0.67,replace = FALSE)
```

Creamos dos variables para guardar los datos seleccionados para el train dataset y test dataset:

```
train_onehot_ann<-onehot_ann[recogida_datos_onehot_ann,]
test_onehot_ann<-onehot_ann[-recogida_datos_onehot_ann,]
```

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

Utilizaremos la función `set.seed(1234567)` antes de ejecutar cada modelo de clasificación.

(d) Crear dos modelos de red neuronal artificial de una sola capa oculta con 10 nodos y 40 nodos, respectivamente. Aplicar los datos de training para ajustar los modelos y posteriormente, predecir el tipo de estructura secundaria en los datos del test.

Red neuronal artificial con 10 nodos

Para crear el modelo de red neuronal artificial utilizaremos el paquete `neuralnet`.

Introducimos las líneas de código pertinentes para crear un modelo de red neuronal con 10 nodos (hidden = 10). Para incluir todas las columnas en el modelo utilizamos la wildcard ".". Para crear el modelo utilizamos las variables del train dataset.

```
set.seed(1234567)
data_model_h10<- neuralnet(giro+tira+helice ~ ., data = train_onehot_ann,hidden = 10,linear.output = FA
```

Ahora podemos realizar la evaluación del modelo comparandolo con los resultados del test dataset (recogemos el valor de probabilidad de ser clasificado en cada caso, guardado en la variable "net.result"):

```
data_model_results_h10<- compute(data_model_h10,test_onehot_ann[,1:340])$net.result
```

Creamos una función que nos permita transformar el output binario en output categórico:

```
maxidx <- function(arr) {
  return(which(arr == max(arr)))
}
```

Aplicamos la función recién creada en los resultados del modelo:

```
transf_data_h10<-apply(data_model_results_h10, 1, maxidx)
```

Realizamos la predicción con los resultados anteriores:

```
predicted_data_h10<- c("_","e","h")[transf_data_h10]
```

Red neuronal artificial con 40 nodos

Para crear el modelo de red neuronal artificial utilizaremos el paquete `neuralnet`.

Introducimos las líneas de código pertinentes para crear un modelo de red neuronal con 10 nodos (hidden = 10). Para incluir todas las columnas en el modelo utilizamos la wildcard ".". Para crear el modelo utilizamos las variables del train dataset.

```
set.seed(1234567)
data_model_h40<- neuralnet(giro+tira+helice ~ ., data = train_onehot_ann,hidden = 40,linear.output = FA
```

Ahora podemos realizar la evaluación del modelo comparándolo con los resultados del test dataset (recogemos el valor de probabilidad de ser clasificado en cada caso, guardado en la variable "net.result"):

```
data_model_results_h40<- compute(data_model_h40,test_onehot_ann[,1:340])$net.result
```

Aplicamos la función `maxidx()` creada anteriormente en los resultados del modelo:

```
transf_data_h40<-apply(data_model_results_h40, 1, maxidx)
```

Realizamos la predicción con los resultados anteriores:

```
predicted_data_h40<- c("_","e","h")[transf_data_h40]
```

(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.

Ppodemos utilizar el paquete `caret` para observar los resultados de las predicciones de ambos modelos:

```
caret::confusionMatrix(table(predicted_data_h10,datos$V18[-recogida_datos_onehot_ann]))
```

```
## Confusion Matrix and Statistics
##
##
## predicted_data_h10    _    e    h
##                   _ 1497  189  205
##                   e  130  339   75
##                   h  191   89  585
##
## Overall Statistics
##
##                Accuracy : 0.7336
##                  95% CI : (0.7182, 0.7487)
##     No Information Rate : 0.5509
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5445
##
##  Mcnemar's Test P-Value : 0.005581
##
## Statistics by Class:
##
##                      Class: _ Class: e Class: h
## Sensitivity            0.8234   0.5494   0.6763
## Specificity            0.7341   0.9236   0.8850
## Pos Pred Value         0.7916   0.6232   0.6763
## Neg Pred Value         0.7722   0.8991   0.8850
## Prevalence             0.5509   0.1870   0.2621
## Detection Rate         0.4536   0.1027   0.1773
## Detection Prevalence   0.5730   0.1648   0.2621
## Balanced Accuracy      0.7788   0.7365   0.7807
```

En la matriz de confusión del modelo de 10 nodos obtenemos que la precisión balanceada de las predicciones para los giros es de 0.78, para las tiras es de 0.73 y para las hélices es de 0.78.

```
caret::confusionMatrix(table(predicted_data_h40,datos$V18[-recogida_datos_onehot_ann]))
```

```
## Confusion Matrix and Statistics
```

```
##
##
## predicted_data_h40    _    e    h
##                   _ 1553  148  191
##                   e  131  403   71
##                   h  134   66  603
##
## Overall Statistics
##
##                Accuracy : 0.7755
##                  95% CI : (0.7608, 0.7896)
##     No Information Rate : 0.5509
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.6169
##
##  Mcnemar's Test P-Value : 0.01062
##
## Statistics by Class:
##
##                      Class: _ Class: e Class: h
## Sensitivity            0.8542   0.6532   0.6971
## Specificity            0.7713   0.9247   0.9179
## Pos Pred Value         0.8208   0.6661   0.7509
## Neg Pred Value         0.8118   0.9206   0.8951
## Prevalence             0.5509   0.1870   0.2621
## Detection Rate         0.4706   0.1221   0.1827
## Detection Prevalence   0.5733   0.1833   0.2433
## Balanced Accuracy      0.8127   0.7889   0.8075
```

En la matriz de confusión del modelo de 40 nodos obtenemos que la precisión balanceada de las predicciones para los giros es de 0.81, para las tiras es de 0.79 y para las hélices es de 0.81.

Gracias a ambas matrices de confusión obtenidas podemos comparar la precisión de los dos modelos. Observando tanto la sensibilidad como la especificidad de ambos modelos se ve que en ambos atributos los tres clasificadores del modelo de 40 nodos son significativamente mejores que en el modelo de 10 nodos.Por lo tanto, observando la precisión balanceada de los tres clasificadores, se observa consecuentemente una mejora de dicha precisión en los tres clasificadores cuando se realiza el modelo con 40 nodos en comparación del modelo con 10 nodos.
Con estos datos podemos decir con certeza que el modelo de 40 nodos tiene un mejor rendimiento que el modelo de 10 nodos. Igualmente, se recomendaría buscar el modelo con mayor número de nodos que tenga la mayor precisión posible sin tener overfitting o que su complejidad no valga la pena por su pequeño aumento en la precisión.

(f) Usar el paquete caret modelo 'mlp' para implementar la arquitectura de 40 nodos en la capa oculta, usando 5-fold crossvalidation. Repetir el análisis balanceando las tres clases, 'h', 'e' y '_', a 1000 observaciones por clase y usar la semilla *12345* para el muestreo. Comentar los resultados.

Uso del paquete caret con el modelo 'mlp'

Como el paquete caret nos permite tener los valores de clasificación como factor en una única columna, creamos un nuevo dataframe con los datos en codificación onehot que contienen ya los tipos de esctructura del aminoácido 9:

```
caret_onehot_ann<-onehot_ann[,1:340]
caret_onehot_ann$estructura<-datos$V18
```

Creamos el train y test datasets utilizando las funciones del paquete caret:

```
set.seed(12345)
particion_caret_ann<- createDataPartition(caret_onehot_ann$estructura, p=0.6666666666666667, list = FALS
caret_train_ann<-caret_onehot_ann[particion_caret_ann,]
caret_test_ann<-caret_onehot_ann[-particion_caret_ann,]
```

Seguidamente utilizamos el paquete `caret` para crear un modelo con el método `mlp`, con 40 nodos ocultos y 5-fold crossvalidation:

```
set.seed(1234567)
caret_model_h40<- train(estructura ~ ., data = caret_train_ann, method = 'mlp',
                        trControl= trainControl(method = 'cv', number = 5),
                        tuneGrid= expand.grid(size = 40))
```

Ahora realizamos la predicción del test dataset con el modelo creado y analizamos la matriz de confusión de esta predicción:

```
prediccion_caret_h40<- predict(caret_model_h40, caret_test_ann[,1:340], type="prob")
```

```
caret_transf_data_h40<-apply(prediccion_caret_h40, 1, maxidx)
```

```
caret_predicted_data_h40<- c("_","e","h")[caret_transf_data_h40]
```

```
caret::confusionMatrix(table(caret_predicted_data_h40,caret_test_ann$estructura))
```

```
## Confusion Matrix and Statistics
##
##
## caret_predicted_data_h40    _    e    h
##                        _ 1610  192  220
##                        e   97  392   55
##                        h  145   60  560
##
## Overall Statistics
##
##                Accuracy : 0.7691
##                  95% CI : (0.7544, 0.7834)
##     No Information Rate : 0.556
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5974
##
##  Mcnemar's Test P-Value : 3.728e-10
##
## Statistics by Class:
##
##                      Class: _ Class: e Class: h
## Sensitivity            0.8693   0.6087   0.6707
## Specificity            0.7214   0.9434   0.9179
## Pos Pred Value         0.7962   0.7206   0.7320
## Neg Pred Value         0.8151   0.9096   0.8928
## Prevalence             0.5560   0.1933   0.2507
## Detection Rate         0.4833   0.1177   0.1681
```

```
## Detection Prevalence    0.6070   0.1633   0.2297
## Balanced Accuracy       0.7954   0.7761   0.7943
```

Uso del paquete caret con el modelo 'mlp' con muestras balanceadas

Para balancear las tres clases debemos escoger 1000 observaciones de cada clase de manera aleatoria. Para ello podemos crear subsets de cada clase para escoger un `sample` de 1000 muestras (utilizando `set.seed(12345)` de cada subset y unir todos los registros en un nuevo dataset.

```
set.seed(12345)
caret_giro<-subset(caret_onehot_ann,caret_onehot_ann$estructura =="_")
caret_tira<-subset(caret_onehot_ann,caret_onehot_ann$estructura =="e")
caret_helice<-subset(caret_onehot_ann,caret_onehot_ann$estructura =="h")

recogida_datos_giro<-sample(1:nrow(caret_giro), 1000, replace = FALSE)
recogida_datos_tira<-sample(1:nrow(caret_tira), 1000, replace = FALSE)
recogida_datos_helice<-sample(1:nrow(caret_helice), 1000, replace = FALSE)

sample_giro<-caret_giro[recogida_datos_giro,]
sample_tira<-caret_tira[recogida_datos_tira,]
sample_helice<-caret_helice[recogida_datos_helice,]

caret_balanceado<-rbind(sample_giro,sample_tira,sample_helice)
```

Ahora que tenemos el dataset con 3000 muestras donde hay 1000 muestras por cada tipo de estructura secundaria, realizamos el mismo proceso con el paquete caret realizado anteriormente pero con estos datos balanceados:

```
set.seed(12345)
particion_caret_balanceada<- createDataPartition(caret_balanceado$estructura, p=0.6666666666666667, lis
caret_train_balanceado<-caret_balanceado[particion_caret_balanceada,]
caret_test_balanceado<-caret_balanceado[-particion_caret_balanceada,]
```

```
set.seed(1234567)
caret_model_h40_balanceado<- train(estructura ~ ., data = caret_train_balanceado, method = 'mlp',
                                   trControl= trainControl(method = 'cv', number = 5),
                                   tuneGrid= expand.grid(size = 40))
```

Ahora realizamos la predicción del test dataset con el modelo creado y analizamos la matriz de confusión de esta predicción:

```
prediccion_caret_h40_balanceada<- predict(caret_model_h40_balanceado, caret_test_balanceado[,1:340], ty
```

```
caret_transf_data_h40_balanceado<-apply(prediccion_caret_h40_balanceada, 1, maxidx)
```

```
caret_predicted_data_h40_balanceado<- c("_","e","h")[caret_transf_data_h40_balanceado]
```

```
caret::confusionMatrix(table(caret_predicted_data_h40_balanceado,caret_test_balanceado$estructura))
```

```
## Confusion Matrix and Statistics
##
##
## caret_predicted_data_h40_balanceado   _   e   h
##                                   _ 195  60  66
##                                   e  68 226  57
##                                   h  70  47 210
##
```

```
## Overall Statistics
##
##                Accuracy : 0.6316
##                  95% CI : (0.6009, 0.6616)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4474
##
##  Mcnemar's Test P-Value : 0.6641
##
## Statistics by Class:
##
##                      Class: _ Class: e Class: h
## Sensitivity            0.5856   0.6787   0.6306
## Specificity            0.8108   0.8123   0.8243
## Pos Pred Value         0.6075   0.6439   0.6422
## Neg Pred Value         0.7965   0.8349   0.8170
## Prevalence             0.3333   0.3333   0.3333
## Detection Rate         0.1952   0.2262   0.2102
## Detection Prevalence   0.3213   0.3514   0.3273
## Balanced Accuracy      0.6982   0.7455   0.7275
```

Comentario de resultados

Las predicciones del modelo en el que se ha utilizado el paquete `caret` con 5-fold Crossvalidation comparado con el modelo de `neuralnet` (sin 5-fold Crossvalidation) tiene en general una menor precisión, ya que se observa una bajada entre un 1% y 2% en las precisiones balanceadas (tanto la general como las precisiones de cada estructura).
Utilizando 1000 muestras de cada estructura (con un total de 3000 muestras) encontramos una significativa disminución en la precisión de los resultados con una disminución de la precisión general del 14% y disminución de precisiones por estructura entre un 5% y un 10%. Esto puede deberse a que aunque los datos se encuentren balanceados para tener similar cantidad de muestras por estructura secundaria en el train dataset, el número de muestras con los que puede trabajar el modelo se reduce drásticamente. Esto implica que el modelo tiene un menor número de muestras en las que sustentar sus predicciones posteriores, dando mayor o menor peso del que debería a las distintas características. Por lo tanto podemos indicar que nos encontramos en un modelo "underfitted" que da lugar a unas predicciones menos precisas que los anteriores modelos.
Concluímos así que el mejor modelo hasta ahora con 40 nodos es el que utiliza el paquete `nnet` sin 5-fold Crossvalidation.

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

(a) Leer el fichero *data4.csv* donde cada registro contiene una secuencia de 17 aminoácidos y su clase de estructura secundaria del aminoácido central (posición 9), donde los caracteres 'h', 'e' y '-' representan α-helix, β-sheet y coil, respectivamente.

Leemos el fichero **data4.csv** y observamos como se encuentran distribuidos los datos:

```
datos<-read.delim("data4.csv", header = TRUE, sep = ";")
str(datos)
```

```
## 'data.frame':    10000 obs. of  18 variables:
```

```
##  $ V1 : Factor w/ 20 levels "A","C","D","E",..: 16 10 10 10 17 6 6 10 16 16 ...
##  $ V2 : Factor w/ 20 levels "A","C","D","E",..: 16 10 10 10 18 14 7 11 1 6 ...
##  $ V3 : Factor w/ 20 levels "A","C","D","E",..: 13 14 12 12 16 20 1 18 3 3 ...
##  $ V4 : Factor w/ 20 levels "A","C","D","E",..: 5 20 13 1 5 1 5 3 3 16 ...
##  $ V5 : Factor w/ 20 levels "A","C","D","E",..: 16 20 9 9 4 18 12 12 18 20 ...
##  $ V6 : Factor w/ 20 levels "A","C","D","E",..: 14 6 9 6 1 7 18 19 9 17 ...
##  $ V7 : Factor w/ 20 levels "A","C","D","E",..: 9 11 20 4 17 6 4 15 9 10 ...
##  $ V8 : Factor w/ 20 levels "A","C","D","E",..: 6 17 8 17 5 18 5 13 1 16 ...
##  $ V9 : Factor w/ 20 levels "A","C","D","E",..: 3 4 13 5 1 17 3 1 5 10 ...
##  $ V10: Factor w/ 20 levels "A","C","D","E",..: 4 11 6 4 5 16 3 14 1 16 ...
##  $ V11: Factor w/ 20 levels "A","C","D","E",..: 15 12 17 18 10 5 16 13 8 6 ...
##  $ V12: Factor w/ 20 levels "A","C","D","E",..: 20 20 9 1 8 18 4 10 8 1 ...
##  $ V13: Factor w/ 20 levels "A","C","D","E",..: 9 20 11 10 9 16 3 9 3 16 ...
##  $ V14: Factor w/ 20 levels "A","C","D","E",..: 20 17 18 16 16 15 9 12 14 20 ...
##  $ V15: Factor w/ 20 levels "A","C","D","E",..: 18 18 5 12 13 10 1 15 3 17 ...
##  $ V16: Factor w/ 20 levels "A","C","D","E",..: 3 10 13 9 3 6 18 14 9 1 ...
##  $ V17: Factor w/ 20 levels "A","C","D","E",..: 13 5 6 6 16 2 10 8 16 6 ...
##  $ V18: Factor w/ 3 levels "_","e","h": 1 1 1 2 2 2 1 1 3 1 ...
```

Tenemos 17 columnas que indican qué aminoácido se encuentra en qué posición de la proteína. Además, nos encontramos con la columna 18 que contiene la clase de aminoácido que se encuentra en la posición central de la proteína, es decir, su estructura secundaria (α-helix, β-sheet o coil). Cabe mencionar que tenemos 10000 proteínas, una cantidad suficiente para poder obtener consistentes resultados a partir de un test dataset y un training dataset.
Antes de avanzar al siguiente apartado debemos procesar los datos para que se encuentren en codificación "one-hot". Para ello, debemos separar de la función las etiquetas que se encuentran en la columna 18 (el tipo de estructura secundaria del aminoácido 9 de la proteína).

```
peptidos_svm<-datos[,1:17]
```

Utilizamos la función creada previamente en el apartado 2 para realizar la codificación onehot de los aminoácidos. Utilizando esta función obtendremos 10000 filas (todos los registros) y 340 columnas (1 columna para cada número de la codificación onehot) todas en formato numérico, formato necesario para el entrenamiento posterior.

```
onehot_svm<-onehot_f(peptidos_svm)
```

Ahora que hemos realizado correctamente la codificación onehot de los datos, unimos las etiquetas para tener el dataset completo. En este caso las etiquetas pueden encontrarse en formato categórico, así que las introducimos directamente en la columna `class`:

```
onehot_svm$class<-datos$V18
```

(b) Utilizando la semilla aleatoria 12345, separar los datos en dos partes, una parte para training (67%) y una parte para test (33%).

Hacemos la selección de datos, 67% para train dataset y 33% para el test dataset:

```
set.seed(12345)
recogida_datos_onehot_svm<-sample(1:nrow(onehot_svm), nrow(onehot_svm)*0.67,replace = FALSE)
```

Creamos dos variables para guardar los datos seleccionados para el train dataset y test dataset:

```
train_onehot_svm<-onehot_svm[recogida_datos_onehot_svm,]
test_onehot_svm<-onehot_svm[-recogida_datos_onehot_svm,]
```

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

Utilizaremos la función `set.seed(1234567)` antes de ejecutar cada modelo de clasificación.

(d) Utilizar la función lineal y la RBF para ajustar un modelo de SVM basado en el training para predecir el tipo de estructura secundaria en los datos del test.

SVM utilizando la función lineal

Para entrenar el modelo vamos a utilizar la función `ksvm()` del paquete `kernlab`. Con esta función vamos a entrenar el modelo con las muestras de `train_onehot_svm` utilizando como objetivo la variable categórica `class` y como predictores todas las características del data frame. Utilizaremos para el modelo el kernel 'vanilladot', que es el la función lineal en la que se pide el ajuste del modelo.

```
set.seed(1234567)
data_model_linear <- ksvm(class ~ ., data = train_onehot_svm, kernel = "vanilladot")
```

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

Ahora utilizamos la función `predict()` para utilizar el modelo recién entrenado para predecir el test dataset:

```
predicted_data_linear <- predict(data_model_linear, test_onehot_svm[1:340])
```

SMV utilizando la función RBF

Para entrenar el modelo vamos a utilizar el mismo código que en la función lineal hecha anteriormente pero modificando el kernel de 'vanilladot' a 'rbfdot', que es la función que se pide ahora:

```
set.seed(1234567)
data_model_rbf <- ksvm(class ~ ., data = train_onehot_svm, kernel = "rbfdot")
```

Ahora utilizamos la función `predict()` para utilizar el modelo recién entrenado para predecir el test dataset:

```
predicted_data_rbf <- predict(data_model_rbf, test_onehot_svm)
```

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

Para observar el rendimiento de ambos modelos, podemos realizar una matriz de confusión para cada modelo:

```
caret::confusionMatrix(predicted_data_linear, test_onehot_svm$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    _    e    h
##          _ 1501  292  426
##          e  134  240   76
##          h  183   85  363
##
## Overall Statistics
##
##                Accuracy : 0.6376
##                  95% CI : (0.6209, 0.654)
##     No Information Rate : 0.5509
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                   Kappa : 0.3457
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: _ Class: e Class: h
## Sensitivity            0.8256  0.38898   0.4197
## Specificity            0.5155  0.92173   0.8899
## Pos Pred Value         0.6764  0.53333   0.5753
## Neg Pred Value         0.7068  0.86772   0.8119
## Prevalence             0.5509  0.18697   0.2621
## Detection Rate         0.4548  0.07273   0.1100
## Detection Prevalence   0.6724  0.13636   0.1912
## Balanced Accuracy      0.6706  0.65535   0.6548
```

En la matriz de confusión del modelo SVM con un kernel lineal obtenemos que la precisión balanceada de las predicciones para los giros es de 0.67, para las tiras es de 0.65 y para las hélices es de 0.65.

```
caret::confusionMatrix(predicted_data_rbf, test_onehot_svm$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    _    e    h
##          _ 1708  317  388
##          e   30  252   28
##          h   80   48  449
##
## Overall Statistics
##
##                Accuracy : 0.73
##                  95% CI : (0.7145, 0.7451)
##     No Information Rate : 0.5509
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4942
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: _ Class: e Class: h
## Sensitivity            0.9395  0.40843   0.5191
## Specificity            0.5243  0.97838   0.9474
## Pos Pred Value         0.7078  0.81290   0.7782
## Neg Pred Value         0.8760  0.87793   0.8472
## Prevalence             0.5509  0.18697   0.2621
## Detection Rate         0.5176  0.07636   0.1361
## Detection Prevalence   0.7312  0.09394   0.1748
## Balanced Accuracy      0.7319  0.69341   0.7333
```

En la matriz de confusión del modelo SVM con un kernel RBF obtenemos que la precisión balanceada de las predicciones para los giros es de 0.73, para las tiras es de 0.69 y para las hélices es de 0.73.

Podemos comparar el rendimiento de ambos modelos observando los resultados de sus matrices de confusión. Observamos que el modelo SVM que utiliza el kernel RBF tienen una especificidad y sensibilidad significativamente más elevadas que el modelo SVM que utiliza el kernel lineal. Por lo tanto, la precisión balanceada de los tres tipos de estructura secundaria es significativamente más elevada en el modelo SVM que utiliza el kernel RBF que el que utiliza el kernel linear, teniendo una mayor precisión en los tres tipos de estructuras secundarias.
Con los motivos mencionados anteriormente, podemos afirmar que para este tipo de datos el modelo SVM que utiliza el kernel RBF realiza mejores predicciones que el modelo SVM que utiliza el kernel linear. Aun así, cabe mencionar que igualmente la precisión sigue sin ser muy elevada así que se aconsejaría buscar otro kernel o modificar el modelo.

(f) Usar el paquete caret modelo svmRBF para aplicar el algoritmo de SVM con 5-fold crossvalidation. Repetir el análisis balanceando las tres clases, 'h', 'e' y '_', a 1000 observaciones por clase y usar la semilla *12345* para el muestreo. Comentar los resultados.

Uso del paquete caret con el modelo svmRBF

Realizamos un proceso similar al explicado en el ejercicio 3 apartado f, pero cambiando el método a utilizar por el paquete caret.

Como el paquete caret nos permite tener los valores de clasificación como factor en una única columna, creamos un nuevo dataframe con los datos en codificación onehot que contienen ya los tipos de esctructura del aminoácido 9:

```
caret_onehot_svm<-onehot_svm[,1:340]
caret_onehot_svm$estructura<-datos$V18
```

Creamos el train y test datasets utilizando las funciones del paquete caret:

```
set.seed(12345)
particion_caret_svm<- createDataPartition(caret_onehot_svm$estructura, p=0.6666666666666667, list = FALS
caret_train_svm<-caret_onehot_svm[particion_caret_svm,]
caret_test_svm<-caret_onehot_svm[-particion_caret_svm,]
```

Seguidamente utilizamos el paquete `caret` para crear un modelo con el método `svmRBF`, llamado simplemente `svmRadial` en este paquete, y creamos el modelo con 5-fold crossvalidation:

```
set.seed(1234567)
caret_model_svm<- train(estructura ~ ., data = caret_train_svm, method = 'svmRadial',
                        trControl= trainControl(method = 'cv', number = 5))
```

Ahora realizamos la predicción del test dataset con el modelo creado y analizamos la matriz de confusión de esta predicción:

```
prediccion_caret_svm<- predict(caret_model_svm, caret_test_svm[,1:340])
```

```
caret::confusionMatrix(table(prediccion_caret_svm,caret_test_svm$estructura))
```

```
## Confusion Matrix and Statistics
##
##
## prediccion_caret_svm    _    e    h
##                    _ 1723  336  353
##                    e   25  239   21
##                    h  104   69  461
##
## Overall Statistics
```

```
##
##                Accuracy : 0.7274
##                  95% CI : (0.7119, 0.7425)
##     No Information Rate : 0.556
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4887
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: _ Class: e Class: h
## Sensitivity            0.9303  0.37112   0.5521
## Specificity            0.5341  0.98288   0.9307
## Pos Pred Value         0.7143  0.83860   0.7271
## Neg Pred Value         0.8596  0.86704   0.8613
## Prevalence             0.5560  0.19334   0.2507
## Detection Rate         0.5173  0.07175   0.1384
## Detection Prevalence   0.7241  0.08556   0.1903
## Balanced Accuracy      0.7322  0.67700   0.7414
```

Uso del paquete caret con el modelo svmRBF con muestras balanceadas

Como ya hicimos este procedimiento en el Ejercicio 3 apartado f), utilizaremos la variable `caret_balanceado` que tiene las muestras balanceadas aleatoriamente.

Seguidamente utilizamos el paquete `caret` para crear un modelo con el método `svmRBF`, llamado simplemente `svmRadial` en este paquete, y creamos el modelo con 5-fold crossvalidation:

```
set.seed(1234567)
caret_model_svm_balanceado<- train(estructura ~ ., data = caret_train_balanceado, method = 'svmRadial',
                         trControl= trainControl(method = 'cv', number = 5))
```

Ahora realizamos la predicción del test dataset con el modelo creado y analizamos la matriz de confusión de esta predicción:

```
prediccion_caret_svm_balanceada<- predict(caret_model_svm_balanceado, caret_test_balanceado[,1:340])
```

```
caret::confusionMatrix(table(prediccion_caret_svm_balanceada,caret_test_balanceado$estructura))
```

```
## Confusion Matrix and Statistics
##
##
## prediccion_caret_svm_balanceada   _   e   h
##                               _ 194  70  80
##                               e  68 210  48
##                               h  71  53 205
##
## Overall Statistics
##
##                Accuracy : 0.6096
##                  95% CI : (0.5786, 0.64)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4144
```

```
##
##  Mcnemar's Test P-Value : 0.8464
##
## Statistics by Class:
##
##                      Class: _ Class: e Class: h
## Sensitivity            0.5826   0.6306   0.6156
## Specificity            0.7748   0.8258   0.8138
## Pos Pred Value         0.5640   0.6442   0.6231
## Neg Pred Value         0.7878   0.8172   0.8090
## Prevalence             0.3333   0.3333   0.3333
## Detection Rate         0.1942   0.2102   0.2052
## Detection Prevalence   0.3443   0.3263   0.3293
## Balanced Accuracy      0.6787   0.7282   0.7147
```

Comentario de resultados

El modelo obtenido utilizando el paquete `caret` y con 5-fold Crossvalidation tiene unas resultados en las predicciones muy similares a los procedentes del paquete `svm`, ya que la diferencia del porcentaje de precisión es mínima tanto en la precisión general como en las precisiones balanceadas de cada estructura secundaria. Aunque haya un muy pequeño aumento de la precisión en el modelo del paquete `smv`, esto puede cambiar en el caso de que utilizáramos otra seed, por lo que podemos considerar ambos modelos igual de eficientes.
En cambio, en el modelo en que se utiliza el paquete `caret` y con 5-fold Crossvalidation pero con muestras balanceadas, se observa una disminución en la precisión del modelo respecto a los anteriores. De igual manera a lo comentado en el apartado f del Ejercicio 3 de esta PEC, esto puede deberse a la falta de una mayor cantidad de muestras, generándose también underfitting (aunque este caso la precisión no disminuye tan notablemente como en el modelo del ejercicio 3).
Por lo tanto, podemos decir que tanto el modelo creado por el paquete `svm` como el del paquete `caret` son igual de buenos a la hora de predecir estas estructuras secundarias.

5. Comentar todos los resultados obtenidos y escoger qué modelo puede ser el mejor.

De los 4 modelos utilizados y observando sus respectivas matrices de confusión podemos descartar para estos datos los modelos basados en SVM, ya que ambos de ellos tienen una menor precisión y, consecuentemente, rinden peor que los modelos basados en Redes Neuronales artificiales. Dentro de los dos modelos que utilizan redes neuronales artificiales, el modelo que utiliza 40 nodos ocultos tiene una mejora entre el 3% y 5% en su precisión al predecir estructuras secundarias que el modelo con 10 nodos ocultos. Esto indica que utilizando 40 nodos ocultos el modelo aún no presenta overfitting. A partir de este pensamiento, se debería decidir teniendo en cuenta tres puntos. Uno, si el aumento de precisión no es realmente significativo en relación al aumento de complejidad del modelo (de 10 nodos a 40), sería preferible quedarse con el modelo con 10 nodos ocultos. Si en cambio, este aumento de precisión se considera significativo sin importar la mayor complejidad del modelo, entonces deberíamos escoger el modelo con 40 nodos ocultos. Finalmente, como el modelo no parece haber presentado overfitting aún, podríamos pensar en crear modelos con un mayor número de nodos y así aumentar un poco más la precisión del modelo, aunque nos encontremos con "diminishing returns" (el modelo tendrá mayor complejidad y cada vez se aumentará menos su precisión).

En el caso de los modelos creados con el paquete `caret` y con 5-fold Crossvalidation, no parece que éstos den lugar a una mejora de la precisión comparado con los comentados en el anterior párrafo. En ambos modelos no se obtienen mejoras significativas (el modelo de nnet sufre una leve disminución en la precisión y el modelo de svmRadial tiene resultados similares al modelo del paquete `svm`), por lo que podemos descartarlos como mejores modelos en este apartado de conclusiones.

Los modelos creados con el dataset de las muestras balanceadas (1000 muestras por cada estructura) se observa

una disminución en su rendimiento general respecto a los modelos comentados anteriormente, seguramente debido al underfitting provocado por la menor cantidad de muestras utilizadas en el entrenamiento. Por este motivo, también podemos decir que estos modelos no son los mejores obtenidos en esta PEC.

Como en mi opinión una mejora del 3% al 5% de la precisión es una mejora significativa, escojo el modelo de red neuronal artificial con 40 nodos ocultos como mejor modelo obtenido en este ejercicio.