PEC3: Secuencias promotoras en E. Coli

Demetrio Muñoz Alvarez

2024-01-02

Contents

Algoritmo "Support Vector Machine"

Las máquinas de soporte vectorial (**SVM**) son un tipo de algoritmo de aprendizaje supervisado para tareas de clasificación y regresión. Desarroladas por Vladimir Vapnik y su equipo en la decada de los noventa (Cortes and Vapnik 1995).

El concepto principal de las SVM es separar los datos con el mayor margen posible y de manera homogénea, mediante la identificación del mejor hiperplano. Un **hiperplano** es una superficie de decisión que divide el espacio de características en regiones asociadas con diferentes clases.

Las SVM son capaces de abordar tanto problemas de clasificación lineal como no lineal. En el caso de problemas no lineales, se recurre al uso de funciones kernel. Estas funciones matemáticas transforman el espacio de características original en un espacio de características de mayor dimensión, donde la separación lineal es más fácil de lograr. Algunos ejemplos de funciones kernel comunes incluyen el kernel lineal, el kernel polinómico, el kernel gaussiano, el string kernel, el chi-square kernel, entre otros.

En el ámbito de la bioinformática, las SVM abordan diversas tareas relacionadas con el análisis de datos biológicos, como:

1. Clasificación de secuencias genéticas
2. Análisis de expresión génica
3. Predicción de estructuras de proteínas
4. Análisis de imágenes médicas
5. Análisis de secuencias filogenéticas

Las fortalezas y debilidades que encontramos en las SVM se reflejan en la siguiente tabla (Lantz 2019):

Fortalezas	Debilidades
Uso en predicción y clasificación. Uso bastante extendido	Requiere especificar parámetro C y función de kernel (prueba y error)
Funciona de forma óptima con ruido	Lento de entrenar, sobre todo a medida que aumenta el número de características
Facilidad de uso en comparación de las redes neuronales	Al igual que las redes neuronales es difícil de interpretar el funcionamiento interno.

Implementar una función para realizar una transformación one-hot encoding de las secuencias del fichero de datos "promoters.txt".

La función **'onehot_encoding()'** toma las secuencias de ADN de un conjunto de datos y aplica una transformación remplazando cada nucleótido por su vector de 4 componentes. Como resultado crea nuevas columnas para cada nueva componente.

```
onehot_encoding <- function(df) {
  # Creamos el diccionario de sustitución para transformar los nucleotidos:
  nucleotides_dict <- list(
    'a' = c(0, 0, 0, 1),
    'c' = c(0, 1, 0, 0),
    'g' = c(0, 0, 1, 0),
    't' = c(1, 0, 0, 0))
  # Aplicamos la transformación y a la columna sequences de nuestro data frame:
  df$sequences <- lapply(df$sequences, function(sequence) {
    for (nucleotide in names(nucleotides_dict)) {
      sequence <- gsub(nucleotide, paste(nucleotides_dict[[nucleotide]], collapse = ""), sequence)
    }
```

```
    return(as.numeric(unlist(strsplit(sequence, ''))))
  })
  # Creamos las columnas para cada componente transformada:
  df[, paste0("V", 1:length(df$sequences[[1]]))] <- do.call(rbind.data.frame, df$sequences)
  # Descartamos la columna original "sequences"
  df$sequences <- NULL
  return(df)
}
```

Desarrollar un código en R (o en Python) que implemente un clasificador de SVM.

Leer y codificar los datos con la función one-hot desarrollada.

```
# Cargamos los datos y creamos los nombres de las columnas.
data_pec3 <- read.csv(data, header = FALSE, col.names = c("class", "name", "sequences"))
```

```
# Usamos la función para ejecutar la codificación one-hot y transformar nuestras secuencias.
data_onehot <- onehot_encoding(data_pec3)
data_onehot$name <- NULL # Eliminamos la columna "name".
# Transformamos la columna "class" a factor para implementar los algoritmos SVM.
data_onehot$class <- factor(data_onehot$class, levels = c("-", "+"))
```

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

El conjunto de datos resultante de la implementación de la función **'onehot_encoding()'** se divide en dos partes: un **conjunto de entrenamiento** (67% de las secuencias) y otro **conjunto de prueba** (33% de las secuencias), que se utilizarán para evaluar el algoritmo **SVM** que implementaremos posteriormente. La separación se realiza utilizando una semilla aleatoria para garantizar la reproducibilidad.

```
set.seed(seed)
# Extraemos los indices para la division de los datos en entrenamiento/prueba.
indices <- sample(1:nrow(data_onehot), size = nrow(data_onehot), replace = FALSE)
# Calculamos el tamaño de los datos de entrenamiento (67%).
train_size <- round(0.67 * nrow(data_onehot))
# Dividimos los datos onehot en entrenamiento/prueba.
train_data <- data_onehot[indices[1:train_size], ]
test_data <- data_onehot[indices[(train_size + 1):nrow(data_onehot)], ] # Conjunto de prueba (33%).
```

Utilizar el kernel lineal y el kernel RBF para crear sendos modelos SVM basados en el training para predecir las clases en los datos del test.

Para implementar el algoritmo SVM con kernel lineal y RBF usamos la funcion `ksvm` del paquete *'kernlab'* (Karatzoglou, Smola, and Hornik 2023). Para el kernel lineal usamos la opción **"vanilladot"** y **"rbfdot"** para RBF. Luego mostramos las matrices de confusión de cada modelo con la función `confusionMatrix()`del paquete *'caret'* (Kuhn and Max 2008).

```r
set.seed(seed)
# Entrenarmos el modelo SVM lineal
svm_linear_model <- kernlab::ksvm(class ~ ., data = train_data, kernel = "vanilladot")
```

```
 Setting default kernel parameters
```

```r
# Predecimos las clases en los datos de prueba para el kernel lineal.
predictions_linear <- predict(svm_linear_model, newdata = test_data)

# Entrenamos el modelo SVM RBF.
svm_rbf_model <- kernlab::ksvm(class ~ ., data = train_data, kernel = "rbfdot")
# Predecimos las clases en los datos de prueba para el kernel RBF.
predictions_rbf <- predict(svm_rbf_model, newdata = test_data)

# Matrices confusión de los modelos lineal y RBF.
linear_model <- caret::confusionMatrix(predictions_linear, test_data$class, positive= "+")
rbf_model <- caret::confusionMatrix(predictions_rbf, test_data$class, positive= "+")
# Mostramos los datos.
linear_model
```

```
Confusion Matrix and Statistics

          Reference
Prediction  -  +
         - 17  2
         +  3 13

               Accuracy : 0.8571
                 95% CI : (0.6974, 0.9519)
    No Information Rate : 0.5714
    P-Value [Acc > NIR] : 0.0003014

                  Kappa : 0.7107

 Mcnemar's Test P-Value : 1.0000000

            Sensitivity : 0.8667
            Specificity : 0.8500
         Pos Pred Value : 0.8125
         Neg Pred Value : 0.8947
             Prevalence : 0.4286
         Detection Rate : 0.3714
   Detection Prevalence : 0.4571
      Balanced Accuracy : 0.8583

       'Positive' Class : +
```

```r
rbf_model
```

```
Confusion Matrix and Statistics

          Reference
Prediction  -  +
```

```
         - 16  1
         +  4 14

               Accuracy : 0.8571
                 95% CI : (0.6974, 0.9519)
    No Information Rate : 0.5714
    P-Value [Acc > NIR] : 0.0003014

                  Kappa : 0.7154

 Mcnemar's Test P-Value : 0.3710934

            Sensitivity : 0.9333
            Specificity : 0.8000
         Pos Pred Value : 0.7778
         Neg Pred Value : 0.9412
             Prevalence : 0.4286
         Detection Rate : 0.4000
   Detection Prevalence : 0.5143
      Balanced Accuracy : 0.8667

       'Positive' Class : +
```

```r
# Mostramos los resultados de los modelo en la siguiente tabla.
table_2 <- data.frame(
  Modelo = c("SVM Lineal", "SVM RBF"),
  Kappa = round(c(linear_model$overall["Kappa"], rbf_model$overall["Kappa"]),2),
  Accuracy = round(c(linear_model$overall["Accuracy"], rbf_model$overall["Accuracy"]),2),
  Sensitivity = round(c(linear_model$byClass["Sensitivity"], rbf_model$byClass["Sensitivity"]),2),
  Specificity = round(c(linear_model$byClass["Specificity"], rbf_model$byClass["Specificity"]),2))

kable(table_2, caption = "Comparación de Modelos ('kernlab')", format = "markdown")
```

Table 2: Comparación de Modelos ('kernlab')

Modelo	Kappa	Accuracy	Sensitivity	Specificity
SVM Lineal	0.71	0.86	0.87	0.85
SVM RBF	0.72	0.86	0.93	0.80

Al comparar los modelos en la tabla anterior observamos que ambos modelos tienen la misma precisión. El modelo SVM RBF tiene una mayor sensibilidad . Por último, el modelo SVM lineal tiene una mayor especificidad .

En conclusión, si ambos modelo son muy similares selecionariamos el modelo más sencillo.

Usar el paquete caret con el modelo svmLinear para implementar un SVM con kernel lineal y 3-fold crossvalidation. Comentar los resultados.

En este apartado vamos a repetir el modelo SVM lineal usando el paquete *'caret'* con la función `train()`y el metodo 'svmLinear'. Usamos una validacion cruzada de 3-fold. Además, utilizamos los mismos conjutnos de entrenamiento y prueba de los modelos anteriores.

```r
set.seed(seed)

# Entrenamos el modelo SVM con kernel lineal y la funcion 'train()'del paquete "caret".
svm_linear_caret_model <- caret::train(class ~ ., data = train_data, method = "svmLinear", trControl = tra

# Predicciones del modelo.
prediction_linear_caret <- predict(svm_linear_caret_model, test_data)

# Matriz de confusión.
linear_caret_model <- confusionMatrix(prediction_linear_caret, test_data$class, positive="+")
linear_caret_model
```

```
Confusion Matrix and Statistics

          Reference
Prediction  -  +
         - 17  2
         +  3 13

               Accuracy : 0.8571
                 95% CI : (0.6974, 0.9519)
    No Information Rate : 0.5714
    P-Value [Acc > NIR] : 0.0003014

                  Kappa : 0.7107

 Mcnemar's Test P-Value : 1.0000000

            Sensitivity : 0.8667
            Specificity : 0.8500
         Pos Pred Value : 0.8125
         Neg Pred Value : 0.8947
             Prevalence : 0.4286
         Detection Rate : 0.3714
   Detection Prevalence : 0.4571
      Balanced Accuracy : 0.8583

       'Positive' Class : +
```

Si lo comparamos con el modelo lineal realizado con el paquete 'kernlab' anterior observamos que ambos modelos tienen las mismas métricas

En resumen, el modelo generado con el paquete 'caret', muestra un **precisión** del 85.71% e indica el porcentaje de predicciones correctas. Los valores de **sensibilidad y especificidad**, 86.67% y 85%, muestran la tasa de verdaderos positivos y tasa de verdaderos negativos respectivamente. Como última metrica, el valor **Kappa** del modelo, con un valor 71.07%, mide la concordancia entre observaciones más allá de lo esperado por azar, un valor mayor que 0 indica que el modelo las predicciones del modelo son mejores de las esperadas aleatoriamente.

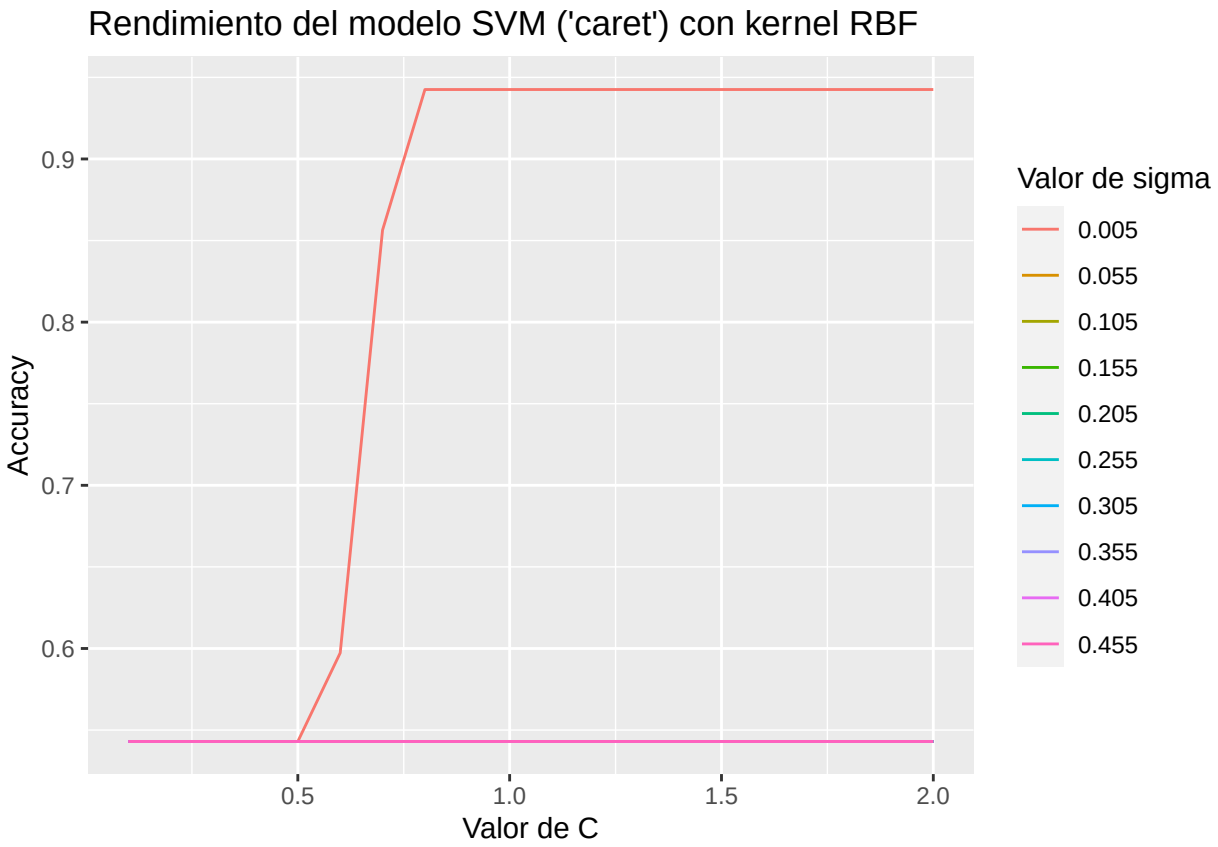
Evaluar el rendimiento del algoritmo SVM con kernel RBF para diferentes valores de los hiperparámetros C y sigma. Orientativamente, se propone explorar valores de sigma en el intervalo (0.005,0.5) y valores de C en el intervalo (0.1, 2). Mostrar un gráfico del rendimiento según los valores de los hiperparámetros explorados.

Para el último modelo, implementamos un algoritmo de clasificación SVM con el kernel 'RBF'. Utilizamos la función *'train()'* del paquete *'caret'* con el método *'svmRadial'*. Además, en este caso, llevamos a cabo una exploración de varios valores para los hiperparámetros C y sigma, los cuales visualizaremos en un gráfico.

```
set.seed(seed)

# Entrenamos el modelo RBF y establecemos los intervalos de sigma y C.
svm_rbf_caret_model <- train(class ~ ., data = train_data, method = "svmRadial",
                             trControl = trainControl(method = "cv", number = 5),
                             tuneGrid = expand.grid(C = seq(0.1, 2, by = 0.1),
                                                    sigma = seq(0.005, 0.5, by = 0.05)))

# Mostramos los hiperparametros mas optimos selecionados.
svm_rbf_caret_model$finalModel
```

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

SV type: C-svc  (classification)
 parameter : cost C = 0.8

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.005

Number of Support Vectors : 69

Objective Function Value : -26.3383
Training error : 0
```

```
# Creamos y mostramos el grafico de rendimiento para los distintos valores de sigma y C.
ggplot(svm_rbf_caret_model$results, aes(x = C, y = Accuracy, color = as.factor(sigma))) +
  geom_line() +
  labs(title = "Rendimiento del modelo SVM ('caret') con kernel RBF",
       x = "Valor de C",
       y = "Accuracy",
       color = "Valor de sigma") +
  scale_color_discrete(name = "Valor de sigma")
```




```r
# Predicciones del modelo.
prediction_rbf_caret <- predict(svm_rbf_caret_model, test_data)
# Calclulamos y mostramos la matrix de confusión.
rbf_caret_model <- confusionMatrix(prediction_rbf_caret, test_data$class, positive= "+")
rbf_caret_model
```

```
Confusion Matrix and Statistics

          Reference
Prediction  -  +
         - 16  1
         +  4 14

               Accuracy : 0.8571
                 95% CI : (0.6974, 0.9519)
    No Information Rate : 0.5714
    P-Value [Acc > NIR] : 0.0003014

                  Kappa : 0.7154

 Mcnemar's Test P-Value : 0.3710934

            Sensitivity : 0.9333
            Specificity : 0.8000
         Pos Pred Value : 0.7778
         Neg Pred Value : 0.9412
             Prevalence : 0.4286
         Detection Rate : 0.4000
```

```
   Detection Prevalence : 0.5143
      Balanced Accuracy : 0.8667

       'Positive' Class : +
```

El modelo **'svm_rbf_caret_model'** ha selecionado el valor **0.005** para el hiperpárametro sigma y **0.8** para el parámetro C, la precisión que se consigue con estos parametros es del **94**%. En el gráfico podemos analizar como varía la precisíon del modelo dependiendo de los diferentes valores de sigma y C y observar que combinación de valores da la mejor precisión.

Con los valores optimos selecionados y las predicciones realizadas con el conjunto de pruebas el modelo muestra las siguientes métricas: precisión del 85.71%, sensibilidad del 93.33%, especificidad del 80% y un valor kappa de 71.54%.

Crear una tabla resumen de los diferentes modelos y sus rendimientos. Comentar y comparar los resultados de la clasificación en función de los valores generales de la clasificación como accuracy y otros para los diferentes clasificadores obtenidos. ¿Qué modelo resulta ser el mejor?

```r
# Mostramos los resultados de todos los modelos.
table_3 <- data.frame(
  Modelo = c("Kernlab SVM Lineal", "Kernlab SVM RBF", "Caret SVM Lineal", "Caret SVM RBF"),
  Kappa = round(c(linear_model$overall["Kappa"], rbf_model$overall["Kappa"], linear_caret_model$overall["K
  Accuracy = round(c(linear_model$overall["Accuracy"], rbf_model$overall["Accuracy"], linear_caret_model$o
  Sensitivity = round(c(linear_model$byClass["Sensitivity"], rbf_model$byClass["Sensitivity"], linear_care
  Specificity = round(c(linear_model$byClass["Specificity"], rbf_model$byClass["Specificity"],linear_caret_

kable(table_3, caption = "Comparación de Modelos ('kernlab vs Caret')", format = "markdown")
```

Table 3: Comparación de Modelos ('kernlab vs Caret')

Modelo	Kappa	Accuracy	Sensitivity	Specificity
Kernlab SVM Lineal	0.71	0.86	0.87	0.85
Kernlab SVM RBF	0.72	0.86	0.93	0.80
Caret SVM Lineal	0.71	0.86	0.87	0.85
Caret SVM RBF	0.72	0.86	0.93	0.80

En general, los cuatro modelos son muy similares y no se observan diferencias al implementar los modelos con un paquete específico. Los modelos lineal y RBF son idénticos, ya sea que utilicemos el paquete *caret* o *kernlab*.

En conclusión, dado que los modelos presentan métricas identicas, se podría considerar que el modelo lineal es la mejor opción. Además, al ser modelos idénticos, no hay distinción significativa entre elegir el modelo lineal realizado con el paquete *kernlab* o *caret*.

Referencias

Cortes, Corinna, and Vladimir Vapnik. 1995. "Support-Vector Networks." *Machine Learning* 20: 273–97.

Karatzoglou, Alexandros, Alex Smola, and Kurt Hornik. 2023. *Kernlab: Kernel-Based Machine Learning Lab.* https://CRAN.R-project.org/package=kernlab.

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

Lantz, Brett. 2019. *Machine Learning with r: Expert Techniques for Predictive Modeling.* Packt publishing ltd.