

Predicción de la estructura secundaria de proteínas globulares

Maria Lucas

2023-03-24

Índice

1. Algoritmo k-NN	2
2. Codificación one-hot	2
3. Clasificador knn	3
(a) Carga del fichero y tabla resumen	3
(b) Aplicación one-hot	4
(c) Separación de los datos	4
(d) Aplicación k-NN para predecir la estructura	5
(e) Predicción coil/non-coil y curva ROC	8
(f) Resultados	11
Clasificación para tres estructuras	11
Clasificación para coil y non-coil	12
Comentario global	13

1. Algoritmo k-NN

El algoritmo k-nn (k-nearest neighbors) es un algoritmo de aprendizaje supervisado utilizado para clasificación y regresión. En el proceso de clasificación, el algoritmo busca encontrar la clase más común entre los k ejemplos de entrenamiento más cercanos al punto de consulta. En el proceso de regresión, el algoritmo busca encontrar el valor medio de los k ejemplos de entrenamiento más cercanos al punto de consulta.

El funcionamiento del algoritmo k-nn es bastante sencillo. Primero, se carga un conjunto de datos de entrenamiento que consta de entradas y etiquetas correspondientes. Luego, se toma un punto de consulta (una entrada sin etiquetar) y se calcula la distancia entre ese punto y cada punto en el conjunto de datos de entrenamiento. Las distancias más comunes utilizadas en k-nn son la distancia euclidiana y la distancia Manhattan.

Una vez que se han calculado las distancias, se seleccionan los k puntos de entrenamiento más cercanos al punto de consulta. Si se está realizando clasificación, se seleccionan las etiquetas correspondientes a estos puntos de entrenamiento y se toma la etiqueta más común como la etiqueta asignada al punto de consulta. Si se está realizando regresión, se toma el valor medio de las etiquetas de los k puntos de entrenamiento más cercanos como el valor asignado al punto de consulta.

La elección del valor de k en el algoritmo k-nn es un factor crítico que puede afectar significativamente el rendimiento del modelo. Si k es demasiado pequeño, el modelo puede ser sensible al ruido en los datos y puede sobreajustarse. Si k es demasiado grande, el modelo puede subajustarse y no ser capaz de capturar patrones sutiles en los datos. El valor de k dependerá del conjunto de datos y el problema específico que se está abordando, aunque se puede empezar por la raíz cuadrada del número de datos e ir ajustando.

Ventajas	Inconvenientes
Simple y fácil de interpretar	No produce un modelo
Rápida fase de entrenamiento	Lenta fase de clasificación
No paramétrico	Se debe escoger una k apropiada
Buen rendimiento en datos con pocos atributos	Computacionalmente costoso para gran cantidad de datos
Se puede actualizar a tiempo real con nuevos datos	Sensible a datos redundantes y a valores atípicos
	Requiere pre-procesamiento de los datos

2. Codificación one-hot

```
def encode_sequence(sequence):  
  
    # Define a dictionary that maps amino acids to their corresponding  
    # positions in the one-hot encoding  
    aa_to_index = {'A': 0, 'R': 1, 'N': 2, 'D': 3, 'C': 4, 'Q': 5, 'E': 6, 'G':  
    : 7, 'H': 8, 'I': 9, 'L': 10, 'K': 11, 'M': 12, 'F': 13, 'P': 14, 'S': 15,  
    'T': 16, 'W': 17, 'Y': 18, 'V': 19}  
  
    # Empty list to store the encoding  
    encoding = []  
  
    # Iterate over each amino acid in the sequence and encode it using the  
    # aa_to_index dictionary  
    for aa in sequence:  
        index = aa_to_index[aa]
```

```

        # Creates a list of 20 "0" except for the element at the position of
        the index where it puts a 1
        # The extend method makes the new encoded aa be added to the encoding
        list
        encoding.extend([1 if i == index else 0 for i in range(20)])

    return encoding

```

3. Clasificador knn

(a) Carga del fichero y tabla resumen

```

# Loading data
import pandas as pd
data = pd.read_csv('data4.csv', delimiter=";").values

from tabulate import tabulate
import numpy as np

def sum_table(data):

    # Creates a dictionary to store the counts of each structure
    counts = {}

    # Iterates over the sequence and adds 1 to the counter of the structure
    encountered
    for seq in data:
        # We use column -1 because it is where the structure info is located
        structure = seq[-1]
        # If the structure is already in counts add 1
        if structure in counts:
            counts[structure] += 1
        # If the structure is not in counts adds it and makes it 1
        else:
            counts[structure] = 1

    # Calculates the total number of structures
    total = sum(counts.values())

    # Formats the table itself
    headers = ["Estructura", "Contador", "Porcentaje"]
    tabla = []
    # For each structure in the dic
    for structure in counts:
        count = counts[structure] # Saves the number of instances as count
        percentage = count / total * 100 # Calculates the percentage
        tabla.append([structure, count, percentage])# Adds it to the table

    return print(tabulate(tabla, headers=headers, floatfmt=".2f"))

# Apply the function

```

```
sum_table(data)
```

## Estructura	Contador	Porcentaje
## _	5557	55.57
## e	1935	19.35
## h	2508	25.08

Parece ser que la clase coil está sobrerrepresentada (representa más de la mitad del total de los datos) con respecto a las otras dos clases, esto nos afectará al modelo de predicción. Cuando una de las clases está sobrerrepresentada en un conjunto de datos, el modelo puede ser más propenso a predecir esa clase con mayor frecuencia y tener dificultades para reconocer las otras clases menos frecuentes. En este caso, el modelo puede tener una tendencia a predecir que los nuevos casos pertenecen a la clase coil con mayor frecuencia que al resto de clases. Para abordar este problema, se pueden tomar medidas como aumentar la cantidad de casos de las clases menos frecuentes o ajustar las ponderaciones de las clases en el modelo durante el entrenamiento para equilibrar la influencia de cada clase.

Nota: He añadido el apartado porcentaje aunque no se pidiera para que fuera más fácil ver la representación de cada clase.

(b) Aplicación one-hot

```
import numpy as np

# Save the last column as y because it contains the structures to predict
y = data[:, -1]

# Removed the structures columns from the data
list_seq = np.delete(data, -1, 1)

# Create the list to store the encoded data
seq_encoded = []

# Iterate over the data applying the encoding function and storing it in the list
for seq in list_seq:
    seq_encoded.append(encode_sequence(seq))
```

(c) Separación de los datos

```
from sklearn.model_selection import train_test_split

# Transform the encoded list sequence to an array
seq_encoded = (np.array(seq_encoded))

# Split the data into training and testing sets setting the seed to 123
seq_encoded_train, seq_encoded_test, y_train, y_test = train_test_split(
    seq_encoded, y, test_size=0.33, random_state=123)

# Apply the sum function to examine the data sets
sum_table(y_test)
```

## Estructura	Contador	Porcentaje
## h	779	23.61
## _	1829	55.42
## e	692	20.97

```
sum_table(y_train)
```

## Estructura	Contador	Porcentaje
## h	1729	25.81
## _	3728	55.64
## e	1243	18.55

Como podemos observar, la clase coil sigue teniendo una mayor cantidad de muestras. Sin embargo, es importante destacar que las clases están igualmente representadas en los conjuntos de entrenamiento y prueba. Si por casualidad una clase estuviera subrepresentada o incluso ausente en el conjunto de entrenamiento, el modelo podría tener dificultades para generalizar bien y hacer predicciones precisas en nuevos datos. Esto subraya la importancia de asegurarse de que todas las clases estén adecuadamente representadas en el conjunto de entrenamiento con respecto al conjunto de pruebas y a la realidad en sí misma, para lograr un modelo más preciso y confiable.

(d) Aplicación k-NN para predecir la estructura

```
# Transform the training encoded list sequence to an array
seq_encoded_train = np.array(seq_encoded_train)

from sklearn.metrics import confusion_matrix, cohen_kappa_score,
    classification_report
from sklearn.neighbors import KNeighborsClassifier

# Create the list of k to study
k_list = [1, 3, 5, 7, 11]

# Defining list to store data
results = []
reports = []

# We iterate over the k list to generate a model for each k
for i in k_list:

    knn_i = KNeighborsClassifier(n_neighbors = i) # Creates the
        KNeighborsClassifier object with k=i

    '''Note: By default KNeighborsClassifier takes k=5 and thus n_neighbors=5 is
        not printed in the console'''

    knn_i.fit(seq_encoded_train, y_train) # Fit the model to the training set
    y_pred = knn_i.predict(seq_encoded_test) # Predicts the labels of the test
        data

    # Calculate the parameters to determine de adjust of the model:

    accuracy_i = knn_i.score(seq_encoded_test, y_test) # Accuracy
```

```

kappa_i = cohen_kappa_score(y_test, y_pred) # Kappa value

cm_i = confusion_matrix(y_test, y_pred, labels=["_", "e", "h"]) # Confusion
matrix
order_i = knn_i.classes_ # order of the labels

error_i = 1 - knn_i.score(seq_encoded_test, y_test) # Error

report_i = classification_report(y_test, y_pred, labels=["_", "e", "h"]) #
Report

# Adds the parameters to the corresponding lists
results.append([i, accuracy_i, kappa_i, error_i, cm_i, order_i])
reports.append([i, report_i])

# Creates the results tables from the lists

## 'Note: By default KNeighborsClassifier takes k=5 and thus n_neighbors=5 is
not printed in the console '
## KNeighborsClassifier(n_neighbors=1)
## 'Note: By default KNeighborsClassifier takes k=5 and thus n_neighbors=5 is
not printed in the console '
## KNeighborsClassifier(n_neighbors=3)
## 'Note: By default KNeighborsClassifier takes k=5 and thus n_neighbors=5 is
not printed in the console '
## KNeighborsClassifier()
## 'Note: By default KNeighborsClassifier takes k=5 and thus n_neighbors=5 is
not printed in the console '
## KNeighborsClassifier(n_neighbors=7)
## 'Note: By default KNeighborsClassifier takes k=5 and thus n_neighbors=5 is
not printed in the console '
## KNeighborsClassifier(n_neighbors=11)

tabla = tabulate(results, headers=["k", "Accuracy", "Kappa Value", "Error", "
Confusion matrix", "Class order"])
tabla2 = tabulate(reports, numalign = "right", stralign = "right", headers = [
"k", "Others"])
# Prints the tables
print(tabla)

```

##	k	Accuracy	Kappa Value	Error	Confusion matrix	Class order
##	1	0.759394	0.594782	0.240606	[[1497 162 170] [160 461 71] [165 66 548]]	['_' 'e' 'h']
##	3	0.711212	0.491203	0.288788	[[1536 139 154] [271 359 62] [275 52 452]]	['_' 'e' 'h']
##	5	0.682424	0.428372	0.317576	[[1551 143 135] [334 301 57] [318 61 400]]	['_' 'e' 'h']
##	7	0.653636	0.366468	0.346364	[[1556 136 137] [362 266 64]]	['_' 'e' 'h']

```
##
## 11      0.625758      0.293059  0.374242  [[ 374   70  335]]
##                                     [[1593  109  127]]  ['_' 'e' 'h']
##                                     [ 410  218   64]
##                                     [ 462   63  254]]
```

```
print(tabla2)
```

```
##      k                                     Others
## ----
##      1      precision      recall      f1-score      support
##
##          _      0.82      0.82      0.82      1829
##          e      0.67      0.67      0.67      692
##          h      0.69      0.70      0.70      779
##
##          accuracy                                     0.76      3300
##          macro avg      0.73      0.73      0.73      3300
##          weighted avg      0.76      0.76      0.76      3300
##      3      precision      recall      f1-score      support
##
##          _      0.74      0.84      0.79      1829
##          e      0.65      0.52      0.58      692
##          h      0.68      0.58      0.62      779
##
##          accuracy                                     0.71      3300
##          macro avg      0.69      0.65      0.66      3300
##          weighted avg      0.71      0.71      0.70      3300
##      5      precision      recall      f1-score      support
##
##          _      0.70      0.85      0.77      1829
##          e      0.60      0.43      0.50      692
##          h      0.68      0.51      0.58      779
##
##          accuracy                                     0.68      3300
##          macro avg      0.66      0.60      0.62      3300
##          weighted avg      0.67      0.68      0.67      3300
##      7      precision      recall      f1-score      support
##
##          _      0.68      0.85      0.76      1829
##          e      0.56      0.38      0.46      692
##          h      0.62      0.43      0.51      779
##
##          accuracy                                     0.65      3300
##          macro avg      0.62      0.56      0.57      3300
##          weighted avg      0.64      0.65      0.63      3300
##     11      precision      recall      f1-score      support
##
##          _      0.65      0.87      0.74      1829
##          e      0.56      0.32      0.40      692
##          h      0.57      0.33      0.42      779
##
##          accuracy                                     0.63      3300
##          macro avg      0.59      0.50      0.52      3300
##          weighted avg      0.61      0.63      0.59      3300
```

(e) Predicción coil/non-coil y curva ROC

```
# Preparing y
y2 = np.copy(y) # Copy y to y2 to preserve the original

# Change the structures (h and e to = 1, _ = 0), so 1 = nc, 0 = coil
y2[y2 == "h"] = "1"
y2[y2 == "e"] = "1"
y2[y2 == "_"] = "0"

# Examine the data set
sum_table(y2)

# Split the data set into training and test using the seed 123
```

##	Estructura	Contador	Porcentaje
##	0	5557	55.57
##	1	4443	44.43

```
seq_encoded_train2, seq_encoded_test2, y2_train, y2_test = train_test_split(
    seq_encoded, y2, test_size=0.33, random_state=123)

# Examine the data set
sum_table(y2_train)
```

##	Estructura	Contador	Porcentaje
##	1	2972	44.36
##	0	3728	55.64

```
sum_table(y2_test)
```

#k-NN and ROC curve

##	Estructura	Contador	Porcentaje
##	1	1471	44.58
##	0	1829	55.42

```
from sklearn.metrics import roc_curve, roc_auc_score

# Create the list of k to study
k_list = [1, 3, 5, 7, 11]

# Defining list to store data
results_list2 = []
metrics_list = []
roc_auc_list = []
fpr_list = [] # False positive ratio
tpr_list = [] # True positive Ratio

# We iterate over the k list to generate a model for each k
for i in k_list:
```



```

knn_i = KNeighborsClassifier(n_neighbors=i) # Creates the
      KNeighborsClassifier object with k=i
knn_i.fit(seq_encoded_train2, y2_train) # Fit the model to the training set
y_pred_proba = knn_i.predict_proba(seq_encoded_test2)[: , 1] # Predicted
      probabilities of class 1
y_pred = np.where(y_pred_proba >= 0.5, '1', '0') # Convert predicted
      probabilities to binary class labels

# Calculate the parameters to create roc_curve
fpr, tpr, thresholds = roc_curve(y2_test, y_pred_proba, pos_label = '1')
# Calculate ROC_curve
roc_auc = roc_auc_score(y2_test, y_pred_proba)

# We calculate the parameters to determine de adjust of the model:
# Accuracy
accuracy_i = knn_i.score(seq_encoded_test2, y2_test)

#Kappa Value
kappa_i = cohen_kappa_score(y2_test, y_pred)

# Confusion matrix and order of the labels
cm_i = confusion_matrix(y2_test, y_pred, labels=["0", "1"])
order_i = knn_i.classes_
tn, fp, fn, tp = confusion_matrix(y2_test, y_pred, labels=["0", "1"]).ravel
      ()

# Compute the evaluation metrics
VPP = tp / (tp + fp)
VPN = tn / (tn + fn)
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)

# Error
error_i = 1 - accuracy_i

# Adds the parameters to the results list
results_list2.append([i, accuracy_i, kappa_i, error_i, cm_i, order_i])

metrics_list.append([i, sensitivity, specificity, VPP, VPN])

# Adds the ROC curve, fpr and tpr to the list
roc_auc_list.append(roc_auc)
fpr_list.append(fpr) # add fpr for this value of k to the list
tpr_list.append(tpr) # add tpr for this value of k to the list

# Creates the table to show the parameters

### KNeighborsClassifier(n_neighbors=1)
### KNeighborsClassifier(n_neighbors=3)
### KNeighborsClassifier()
### KNeighborsClassifier(n_neighbors=7)
### KNeighborsClassifier(n_neighbors=11)

```

```

tabla = tabulate(results_list2, headers=["k", "Accuracy", "Kappa Value", "
    Error", "Confusion matrix", "Class order"])
tabla2 = tabulate(metrics_list, headers=["k", "Sensibilidad", "Especificidad",
    "VPP", "VPN"])
print(tabla) # Prints the table

```

##	k	Accuracy	Kappa Value	Error	Confusion matrix	Class order
##	1	0.800909	0.597264	0.199091	[[1497 332] [325 1146]]	['0' '1']
##	3	0.754545	0.502118	0.245455	[[1441 388] [422 1049]]	['0' '1']
##	5	0.730909	0.451173	0.269091	[[1443 386] [502 969]]	['0' '1']
##	7	0.709697	0.407354	0.290303	[[1415 414] [544 927]]	['0' '1']
##	11	0.680909	0.346082	0.319091	[[1396 433] [620 851]]	['0' '1']

```
print(tabla2)
```

```
# Plots the ROC curves
```

##	k	Sensibilidad	Especificidad	VPP	VPN
##	1	0.779062	0.81848	0.775372	0.821625
##	3	0.71312	0.787862	0.729993	0.773484
##	5	0.658736	0.788956	0.715129	0.741902
##	7	0.630184	0.773647	0.691275	0.722307
##	11	0.578518	0.763259	0.662773	0.69246

```
import matplotlib.pyplot as plt
```

```

plt.figure(figsize=(8, 6)) # Set figure size
for i in range(len(k_list)): # Draw as many curves as items in k_list
    plt.plot(fpr_list[i], tpr_list[i], label='k = %d, AUC = %.2f' % (k_list[i],
        roc_auc_list[i])) # Plots the curve for each k and sets the AUC values
    for the legend
plt.plot([0, 1], [0, 1], 'k—') # Draws diagonal line for reference
plt.xlim([0.0, 1.0]) # Set X axis

```

```
## (0.0, 1.0)
```

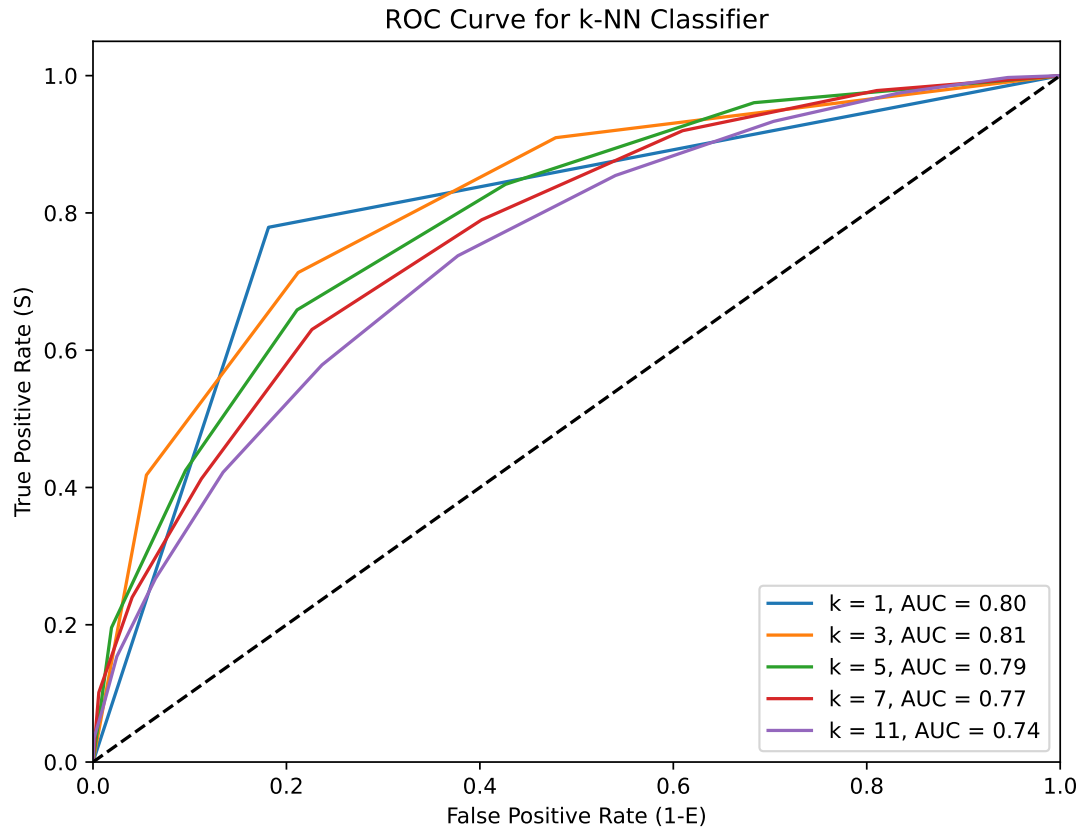
```
plt.ylim([0.0, 1.05]) # Set Y axis
```

```
## (0.0, 1.05)
```

```

plt.xlabel('False Positive Rate (1-E)') # X label
plt.ylabel('True Positive Rate (S)') # Y label
plt.title('ROC Curve for k-NN Classifier') # Graph title
plt.legend(loc="lower right") # Legend
plt.show() # Prints the graph

```



Es importante destacar que, en este nuevo modelo, los datos se distribuyen de manera más equitativa entre las clases. Esto puede ayudar a evitar un posible sesgo en las predicciones del modelo, ya que cada clase tiene una presencia más equilibrada en el conjunto de datos.

Para el cálculo de la curva roc, la función `roc_curve` de `sklearn` calcula automáticamente los thresholds necesarios para poder obtener valores que representen adecuadamente los FPR y TPR sin que sea computacionalmente costoso. En cambio, para el cálculo de la matriz de confusión, se ha usado un threshold estándar de 0.5 que podría no ser el más interesante para el modelo.

El uso de diferentes thresholds para el cálculo de la curva ROC y la matriz de confusión se debe a la naturaleza de cada métrica. Para la curva ROC, se requiere calcular los valores de TPR y FPR para diferentes puntos de corte o thresholds en la salida del modelo. Para facilitar este cálculo, la función `roc_curve` de `sklearn` utiliza un enfoque automático que encuentra los thresholds necesarios de manera eficiente.

Por otro lado, para la matriz de confusión, se utiliza un threshold estándar de 0.5 para clasificar las predicciones del modelo en TN, FP, FN y TP. Sin embargo, este threshold puede no ser el más adecuado para todos los modelos y puede variar según la aplicación. Por lo tanto, sería interesante considerar diferentes thresholds y evaluar el rendimiento del modelo en diferentes puntos de corte para comprender mejor su comportamiento y desempeño en diferentes situaciones.

(f) Resultados

Clasificación para tres estructuras

Previamente a la aplicación del algoritmo, es conveniente asegurarse que todos los tipos de estructura están correctamente representados tanto en el set de entrenamiento como en el de test. Como hemos comentado,

aunque la clase coil está sobrerrepresentada, los sets de entrenamiento y prueba contienen un porcentaje de estructuras similar de cada clase.

Observando los resultados de la clasificación para las tres clases de estructuras secundarias, podemos ver que el modelo que mejor predice es el que usa $k=1$. Podemos deducirlo puesto que tiene un error de clasificación menor (0,24) lo que indica que tan solo predice mal un 24% de los casos del data test. Como bien sabemos este error no deja de ser 1-exactitud (accuracy), así que estos dos parámetros nos aportan la idea de qué tan bien nuestro modelo clasifica la estructura de la cadena de aminoácidos.

El valor kappa está estrechamente relacionado con la exactitud, excepto que éste tiene en cuenta la probabilidad de clasificar correctamente por azar. Cuánto más cercano a 1 es el valor, mejor predice el modelo, mientras que si el valor es cercano a 0, el modelo no predice mejor que una clasificación aleatoria. En este caso el valor más elevado es el de $k=1$, con 0.59. Si seguimos la guía general proporcionada en el libro (Lantz, Brett. Machine Learning with R (2). Packt Publishing, 2015. 452 p. ISBN 9781784394523), el modelo sigue un ajuste moderado-bueno. Depende de la aplicación del modelo si este ajuste es suficientemente bueno para ser usado o no, según si se pueden permitir errores o no.

La precisión, el recall y el F-score también parecen indicar que el modelo con $k=1$ se ajusta mejor. En este caso como hay 3 posibles resultados se calculan sobre cada una de las clases. Aunque el $k=1$ parezca ajustarse mejor, se debe escoger el modelo según la naturaleza de nuestros datos y el problema a resolver. La precisión representa la proporción de ejemplos positivos que son realmente positivos (que se han clasificado como esa clase y lo eran). Si queremos que el modelo sólo clasifique coil en los casos que realmente sea coil, buscaremos un valor alto de precisión para coil. Por otro lado, un valor elevado de recall indicará que nuestro modelo detecta gran parte de los casos positivos, siguiendo el ejemplo anterior, de todas las estructuras que eran coil ha clasificado bien muchas de ellas. Finalmente, si queremos hacernos una idea general podemos consultar el F-score, que combina recall y precisión en un solo valor.

Es importante tener en cuenta que ninguno de los parámetros mencionados por sí solos puede proporcionar una idea completa sobre si un modelo está overfitted o underfitted a los datos de entrenamiento, ya que cada uno de ellos mide diferentes aspectos del rendimiento del modelo. Por lo tanto, es recomendable evaluar múltiples parámetros diferentes y realizar una evaluación cruzada para evitar el overfitting y obtener una visión más completa del rendimiento del modelo.

Clasificación para coil y non-coil

Primeramente, cabe destacar que al separar los datos de esta manera, la clase non-coil está mejor representada que las clases a-helix y b-sheet por separado, así que a priori, pensaríamos que obtendremos un mejor modelo.

Igual que en el modelo anterior, parece ser que el $k=1$ es el más exacto y con menor error. Nuevamente el valor kappa nos indica que estamos ante un modelo de ajuste moderado-bueno. En este caso no examinamos los valores de precisión y recall, ya que al tener sólo dos posibles resultados equivalen a Valor Predictivo Positivo y Sensibilidad, que examinaremos más cómodamente mediante curvas ROC.

Analizando la curva ROC podría parecer que $k=1$ es el que mejor se ajusta, ya que es el que más se aproxima a 1 de sensibilidad y 0 de 1 - especificidad. El modelo parece clasificar de forma bastante equilibrada los verdaderos positivos sin elevar mucho el número de falsos positivos. Lo mismo sucede con los negativos. Lo podemos ver en la curva ROC, pero también en la propia matriz de confusión. El número de falsos positivos y de falsos negativos es bastante similar, sobretodo para $k=1$.

Los falsos positivos y falsos negativos son dos tipos de errores que pueden ocurrir en un modelo de clasificación. Un falso positivo ocurre cuando el modelo predice incorrectamente que un evento pertenece a una clase, cuando en realidad no lo hace. Por otro lado, un falso negativo ocurre cuando el modelo predice incorrectamente que un evento no pertenece a una clase, cuando en realidad sí lo hace.

Si examinamos el AUC, vemos que $k = 3$ es ligeramente superior. Al ser una diferencia de tan solo 0.01, al tener en cuenta el resto de factores, $k=1$ es el modelo que mejor se ajusta. Consultando el libro de referencia (Lantz, Brett. Machine Learning with R (2). Packt Publishing, 2015. 452 p. ISBN 9781784394523), podemos determinar que el ajuste de acuerdo al AUC es bueno/excelente.

Comentario global

En ambos casos hemos obtenido un relativamente buen modelo para predecir la estructura de la secuencia de aminoácidos. Parecería que el modelo coil/non-coil sería más exacto al sólo tener dos posibles resultados en lugar de tres, pero tras examinar los distintos parámetros vemos que en realidad ambos modelos son buenos.

Un valor de k muy pequeño como es $k=1$, puede significar que el modelo está muy ajustado a los datos de entrenamiento (overfitted) y no generalizará bien con datos nuevos. Si quisiéramos ajustar mejor los modelos necesitaríamos usar técnicas como las cross-validation para usar el máximo número de datos posibles para su entrenamiento.

También sería conveniente una vez tengamos el modelo que queremos utilizar, después de ajustar la k y las estructuras a predecir, que creemos un set de datos de validación para asegurarnos que efectivamente el modelo es bueno. Al evaluar el ajuste de varios modelos utilizando el mismo conjunto de prueba, hemos determinado el modelo que mejor predice estos datos en particular, creando así un “overfitting” a los datos de prueba, aunque en realidad, este modelo podría no ser el que mejor prediga datos futuros. Pasando un nuevo set de datos de validación (“nunca vistos”), los nuevos parámetros calculados a partir de este nos darán una idea de la capacidad de predicción del modelo en cuanto a datos futuros.