

# Predicción de la estructura secundaria de proteínas globulares

Maria Lucas

2023-03-24

## Contents

|   |          |
|---|----------|
| <b>1. Algoritmo k-NN</b>                                  | <b>2</b> |
| <b>2. Codificación one-hot</b>                            | <b>2</b> |
| <b>3. Clasificador knn</b>                                | <b>3</b> |
| (a) Carga del fichero y tabla resumen . . . . .           | 3        |
| (b) Aplicación one-hot . . . . .                          | 3        |
| (c) Separación de los datos . . . . .                     | 4        |
| (d) Aplicación k-NN para predecir la estructura . . . . . | 4        |
| (e) Predicción coil/non-coil y curva ROC . . . . .        | 6        |
| (f) Resultados . . . . .                                  | 9        |
| Clasificación para tres estructuras . . . . .             | 9        |
| Clasificación para coil y non-coil . . . . .              | 9        |
| Comentario global . . . . .                               | 9        |

## 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

```
library(reticulate)
```

```
## Warning: package 'reticulate' was built under R version 4.2.3
```

```
use_python("C:/Users/Arialux/AppData/Local/Programs/Python/Python311")
```

```
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':
```

```
    # Initialize an 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]
        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

```

# Carga de los datos
#pip install pandas
import pandas as pd
data = pd.read_csv('data4.csv', delimiter=";").values

# Creación tabla resumen

def sum_table(data):

    coil = 0
    bsheet = 0
    ahelix = 0

    for seq in data:
        if seq[-1] == "_":
            coil += 1
        if seq[-1] == "e":
            bsheet += 1
        if seq[-1] == "h":
            ahelix += 1

    tabla = [
        ["a-helix", ahelix],
        ["b-sheet", bsheet],
        ["coil", coil]
    ]

    return print(tabulate(tabla, headers=["Estructura", "Contador"]))

# Aplicación

from tabulate import tabulate

sum_table(data)

## Estructura      Contador
## -----
## a-helix         2508
## b-sheet         1935
## coil            5557

```

#### (b) Aplicación one-hot

```

import numpy as np

# Guardamos la última columna como y, ya que son las estructuras a predecir

```

```

y = data[:, -1]

# Quitamos la última columna que es la estructura. 1 es columna 0 es row
list_seq = np.delete(data, -1, 1)

seq_encoded = []

for seq in list_seq:
    seq_encoded.append(encode_sequence(seq))

```

### (c) Separación de los datos

```

from sklearn.model_selection import train_test_split

# Volvemos a poner la columna de la estructura
seq_encoded = (np.array(seq_encoded))

# Split the data into training and testing sets
seq_encoded_train, seq_encoded_test, y_train, y_test = train_test_split(seq_encoded, y ,test_size=0.33,

sum_table(y_test)

```

```

## Estructura      Contador
## -----
## a-helix         779
## b-sheet         692
## coil            1829

```

```
sum_table(y_train)
```

```

## Estructura      Contador
## -----
## a-helix         1729
## b-sheet         1243
## coil            3728

```

### (d) Aplicación k-NN para predecir la estructura

```

seq_encoded_train = np.array(seq_encoded_train)

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

k_list = [1, 3, 5, 7, 11]
results = []
reports = []

for i in k_list:
    #results = []

    knn_i = KNeighborsClassifier(n_neighbors = i)
    knn_i.fit(seq_encoded_train, y_train)

```

```

y_pred = knn_i.predict(seq_encoded_test)

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

# Kappa value
kappa_i = cohen_kappa_score(y_test, y_pred)

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

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

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

results.append([i, accuracy_i, kappa_i, error_i, cm_i])
reports.append([i, report_i])

```

```

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

tabla = tabulate(results, headers=["k", "Accuracy", "Kappa Value", "Classification Error ", "Confusion matrix"])
tabla2 = tabulate(reports, numalign = "right", stralign = "right", headers = ["k", "Others"])
print(tabla)

```

| ## | k   | Accuracy | Kappa Value | Classification Error | Confusion matrix |
|----|-----|----------|-------------|----------------------|------------------|
| ## | --- | -----    | -----       | -----                | -----            |
| ## | 1   | 0.759394 | 0.594782    | 0.240606             | [[1497 162 170]  |
| ## |     |          |             |                      | [ 160 461 71]    |
| ## |     |          |             |                      | [ 165 66 548]]   |
| ## | 3   | 0.711212 | 0.491203    | 0.288788             | [[1536 139 154]  |
| ## |     |          |             |                      | [ 271 359 62]    |
| ## |     |          |             |                      | [ 275 52 452]]   |
| ## | 5   | 0.682424 | 0.428372    | 0.317576             | [[1551 143 135]  |
| ## |     |          |             |                      | [ 334 301 57]    |
| ## |     |          |             |                      | [ 318 61 400]]   |
| ## | 7   | 0.653636 | 0.366468    | 0.346364             | [[1556 136 137]  |
| ## |     |          |             |                      | [ 362 266 64]    |
| ## |     |          |             |                      | [ 374 70 335]]   |
| ## | 11  | 0.625758 | 0.293059    | 0.374242             | [[1593 109 127]  |
| ## |     |          |             |                      | [ 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

```

def count_instances(array):
    unique, counts = np.unique(array, return_counts=True)
    dic = dict(zip(unique, counts))

    return print(dic)

```

```

# Preparing y
y2 = np.copy(y)

y2[y2 == "h"] = "1"
y2[y2 == "e"] = "1"
y2[y2 == "_"] = "0"

count_instances(y2)

## {'0': 5557, '1': 4443}

seq_encoded_train2, seq_encoded_test2, y2_train, y2_test = train_test_split(seq_encoded, y2, test_size=0.2, random_state=42)

count_instances(y2_train)

## {'0': 3728, '1': 2972}

count_instances(y2_test)

#k-NN and ROC curve

## {'0': 1829, '1': 1471}

from sklearn.metrics import roc_curve, roc_auc_score

k_list = [1, 3, 5, 7, 11]
accuracy_list2 = []
roc_auc_list = []
fpr_list = [] # create empty list to store fpr for each k
tpr_list = [] # create empty list to store tpr for each k

for i in k_list:
    knn_i = KNeighborsClassifier(n_neighbors=i)
    knn_i.fit(seq_encoded_train2, y2_train)
    y_pred_proba = knn_i.predict_proba(seq_encoded_test2)[: , 1] # predicted probabilities of class 1
    fpr, tpr, thresholds = roc_curve(y2_test, y_pred_proba, pos_label = "1") # calculate fpr, tpr, and thresholds
    roc_auc = roc_auc_score(y2_test, y_pred_proba) # calculate ROC AUC
    accuracy_i = knn_i.score(seq_encoded_test2, y2_test)
    accuracy_list2.append(accuracy_i)
    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

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

print('Accuracy:', accuracy_list2)

## Accuracy: [0.8009090909090909, 0.7545454545454545, 0.7309090909090909, 0.7096969696969697, 0.6809090909090909]

print('ROC AUC:', roc_auc_list)

## ROC AUC: [0.7987709532090992, 0.8143476633540968, 0.7916117658734068, 0.769447518062903, 0.7429243481000001]

```

```

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
for i in range(len(k_list)):
    plt.plot(fpr_list[i], tpr_list[i], label='k = %d, AUC = %.2f' % (k_list[i], roc_auc_list[i]))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])

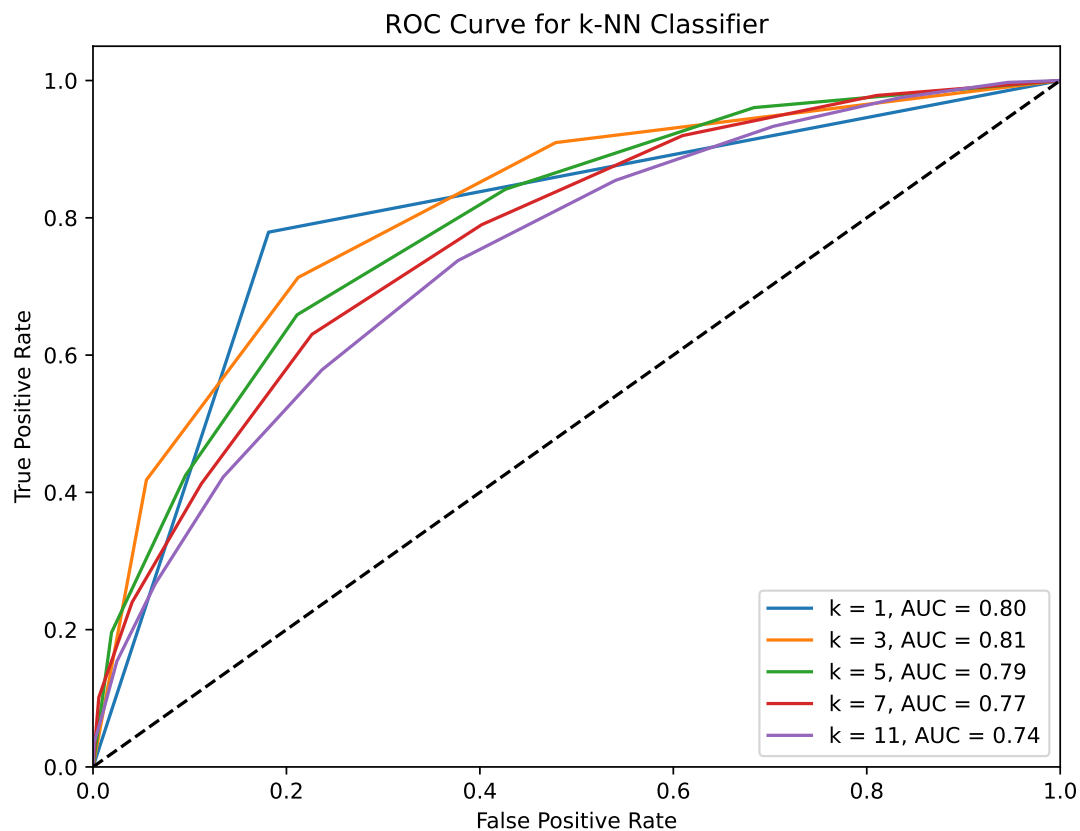
## (0.0, 1.0)

plt.ylim([0.0, 1.05])

## (0.0, 1.05)

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for k-NN Classifier')
plt.legend(loc="lower right")
plt.show()

```





## **(f) Resultados**

### **Clasificación para tres estructuras**

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, 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, escogemos el modelo según la naturaleza de nuestros datos y el problema a resolver. La precisión representa la proporción de ejemplos positivos que eran 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.

Cabe destacar que ninguno de los parámetros que hemos mencionado por sí solos puede proporcionar una idea clara de si un modelo está overfitted o underfitted a los datos, ya que cada uno de ellos mide diferentes aspectos del rendimiento del modelo. En general, se recomienda evaluar varios parámetros diferentes y realizar una evaluación cruzada para evitar overfitting.

### **Clasificación para coil y non-coil**

#### **Comentario global**