Autor:
Iván García Santillán

Algoritmos predictivos de clasificación: Vecinos más cercanos (K-NN)

Introducción

El algoritmo k-Nearest Neighbors (k-NN) es un método de aprendizaje automático supervisado utilizado para problemas de clasificación y regresión. Este tiene diversas aplicaciones prácticas en campos como el análisis de datos, aprendizaje automático y sistemas de recomendación. En medicina, se utiliza para clasificar pacientes en categorías de riesgo basándose en características similares a casos conocidos. En el sector financiero, ayuda en la detección de fraudes identificando patrones de transacciones sospechosas similares a casos previos de fraude. En el comercio minorista y en servicios de streaming, k-NN es esencial para los sistemas de recomendación, sugiriendo productos, películas o música basándose en las preferencias de usuarios con gustos similares. También es utilizado en la agricultura para la clasificación y diagnóstico de enfermedades de plantas, y en la robótica para la navegación y clasificación de obstáculos. La simplicidad y efectividad del algoritmo lo hacen versátil para aplicaciones que requieren clasificación o recomendación basada en similitudes de datos. Aunque también se debe considerar algunas desventajas como una posible disminución de la precisión comparado con otros algoritmos y un uso de memoria moderado dependiendo del tamaño del conjunto de datos utilizado.

A continuación, se detallan el funcionamiento del algoritmo y un ejemplo en Python, Keras y TensorFlow para la predicción del cáncer de mama utilizando un data set público.

Contenido

La idea fundamental detrás del algoritmo k-NN es que los puntos de datos similares tienden a estar cerca unos de otros en el espacio de características. Por lo tanto, para predecir una nueva instancia, el algoritmo busca los "k" puntos de datos más cercanos en el conjunto de entrenamiento y toma una decisión basada en la mayoría de las etiquetas de clase (en el caso de clasificación) o en el promedio (en el caso de regresión) de las etiquetas de los vecinos más cercanos.

Algunos conceptos clave relacionados con el algoritmo k-NN son los siguientes:
Parámetro k: El valor de "k" en k-NN representa el número de vecinos más cercanos que se deben considerar al hacer una predicción. Un valor pequeño de "k" (como 1) puede llevar a predicciones ruidosas y sensibles a valores atípicos, mientras que un valor grande de "k" suaviza las predicciones, pero puede perder detalles finos en los datos. La elección del valor de "k" es crítica en k-NN y puede afectar significativamente el rendimiento del modelo.
Función de Distancia: Para determinar qué puntos de datos son los más cercanos, se utiliza una función de distancia, como la distancia euclidiana, la distancia de Manhattan o la distancia de Minkowski. La elección de la función de distancia depende del problema y los datos.
Normalización: Antes de aplicar k-NN, es importante normalizar las características para que todas tengan la misma influencia en la distancia entre puntos. Esto es especialmente importante cuando las características tienen escalas muy diferentes.

El algoritmo k-NN es simple de entender y de implementar, y es útil en una variedad de aplicaciones, como clasificación de documentos --basado en su contenido tales como correo electrónico (spam, no spam), artículo (tecnología, educación, deportes, política), comentario en redes sociales (positivo, negativo o neutral), revisión de productos (satisfecho, insatisfecho), etc.--, recomendación de productos/películas, diagnóstico médico y más. Sin embargo, es computacionalmente costoso en conjuntos de datos grandes, ya que requiere calcular la distancia entre la nueva instancia y todos los puntos de datos de entrenamiento.

A continuación, se presenta un ejemplo del algoritmo K-NN en Python, Keras y TensorFlow para la predicción del cáncer de mama (benigno=0, maligno=1)

utilizando el conjunto de datos Breast Cancer Wisconsin (Diagnostic) disponible en la librería scikit-learn.

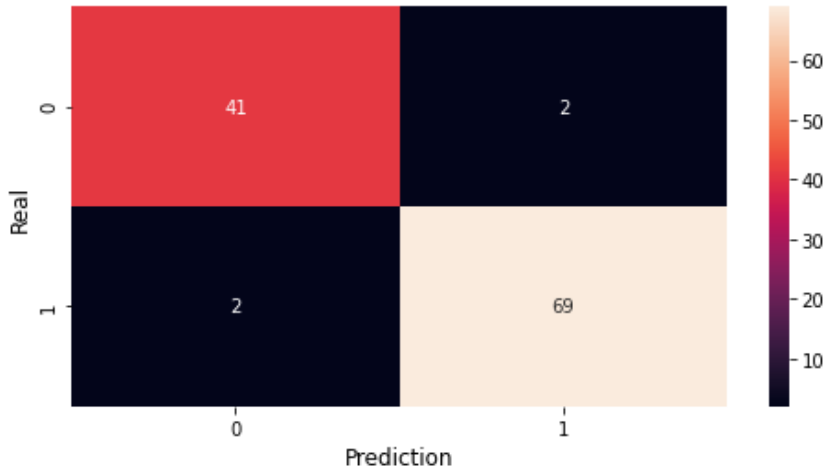
```
# K-NN for breast cancer

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier

# Carga el conjunto de datos Breast Cancer
dataset = load_breast_cancer()
X = dataset.data # 569x30
y = dataset.target # 569x1

# Divide el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normaliza los datos para que todas las características tengan una escala similar
scaler = MinMaxScaler(feature_range=(0,1)) # [0, 1]
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Crea y entrena el modelo K-NN
model = KNeighborsClassifier(n_neighbors=3, p=2,  # Función euclidean
                             weights='distance')

model.fit(X_train, y_train)

# Realiza predicciones usando el conjunto de prueba
y_pred = model.predict(X_test)

# Convierte las probabilidades en etiquetas binarias (0 o 1)
y_pred = (y_pred > 0.5)

# Muestra el informe de evaluación del modelo entrenado
print(classification_report(y_test, y_pred))

# Matriz de confusión:
from sklearn.metrics import confusion_matrix
```

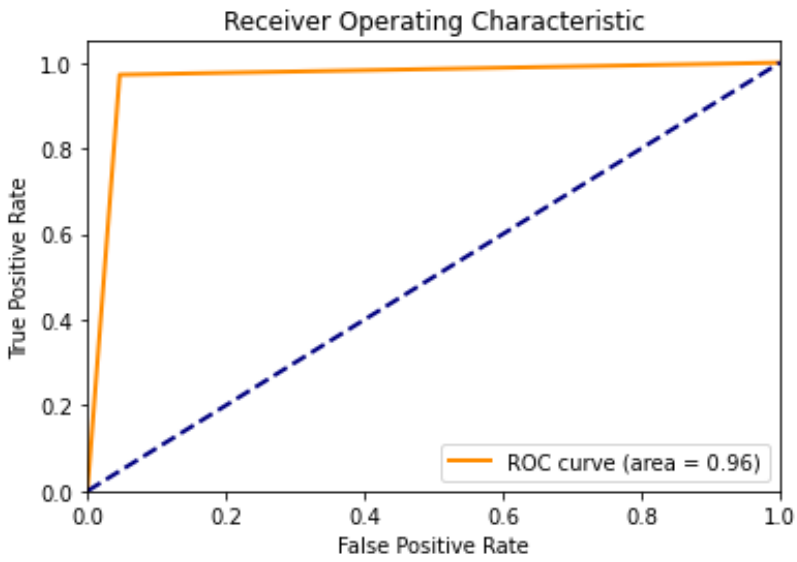
```
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)
print("confusion matrix: \n", cm)
# gráfica de la CM
plt.figure(figsize = (8,4))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Prediction', fontsize = 12)
plt.ylabel('Real', fontsize = 12)
plt.show()

# Exactitud:
from sklearn.metrics import accuracy_score
acc = accuracy_score(y_test, y_pred)
print("accuracy: ", acc)

# Sensibilidad:
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("recall: ", recall)

# Precisión:
from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print("precision: ", precision)

# Especificidad
# 'specificity' is just a special case of 'recall'.
# specificity is the recall of the negative class
specificity = recall_score(y_test, y_pred, pos_label=0)
print("specificity: ", specificity)

# Puntuación F1:
from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("f1 score: ", f1)

# Área bajo la curva:
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y_test, y_pred)
print("auc: ", auc)
```

```
# Curva ROC
from sklearn.metrics import roc_curve
plt.figure()
lw = 2
plt.plot(roc_curve(y_test, y_pred)[0], roc_curve(y_test, y_pred)[1], color='darkorange',lw=lw,
         label='ROC curve (area = %0.2f)' %auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

# R Score (R^2 coefficient of determination)
from sklearn.metrics import r2_score
R = r2_score(y_test, y_pred)
print("R2: ", R)
```

Los resultados del modelo K-NN se presentan a continuación:

accuracy: 0.96
recall: 0.97
precision: 0.97
specificity: 0.95
f1 score: 0.97
auc: 0.96
R^2: 0.85





Un clasificador adecuado es aquel que alcanza una exactitud (accuracy) superior al 85%. Por lo que, este modelo K-NN entrenado cumple con este criterio de calidad.

El algoritmo k-NN es un modelo basado en instancias y, como tal, no aprende parámetros en el sentido tradicional durante el entrenamiento. En lugar de eso, guarda las instancias del conjunto de datos de entrenamiento para hacer predicciones sobre nuevas instancias. Por lo tanto, no tiene "**parámetros**" aprendidos como los coeficientes en la regresión lineal o logística.

Los **hiperparámetros** de k-NN se configuran antes del entrenamiento y afectan cómo el modelo realiza las predicciones. Los principales hiperparámetros en k-NN incluyen *n_neighbors, p, weights* que son esenciales para configurar el comportamiento del modelo K-NN en scikit-learn.

> **n_neighbors**: Especifica el número de vecinos más cercanos que el algoritmo considerará para tomar una decisión sobre la clase o el valor de la predicción. El valor por defecto es 5. Se sugiere que este valor sea impar para evitar empates en la votación.
> **p**: Determina la métrica de distancia utilizada para encontrar los vecinos más cercanos. Con p = 1 utiliza la distancia Manhattan, con p = 2 la distancia Euclidiana.
> **weights**: Especifica cómo se ponderan los votos de los vecinos para determinar la clase o el valor predicho. Con 'uniform' todos los vecinos tienen el mismo peso. Con 'distance' los vecinos tienen un peso inversamente proporcional a su distancia; los vecinos más cercanos tienen mayor influencia.

Aunque el algoritmo k-NN no requiere un entrenamiento explícito en el sentido de ajustar parámetros (como en otros algoritmos), la llamada al método *fit* en scikit-learn es esencial para almacenar los datos de entrenamiento y preparar el modelo para hacer predicciones de manera eficiente. Esto puede incluir la indexación eficiente de los datos o la preparación de estructuras de datos necesarias para la búsqueda rápida de vecinos. Así, el algoritmo puede acceder rápidamente a los datos y calcular las distancias necesarias durante la fase de predicción.

Adicionalmente, tenga en cuenta que se pueden ajustar varios parámetros del algoritmo (como *n_neighbors, p, weights*, etc.) para obtener un modelo óptimo. Revise la documentación (ayuda en línea) del algoritmo para más detalles y experimente con ellos.

Por otro lado, el **método del codo** (elbow) es una técnica utilizada para determinar el número óptimo de vecinos (k) en k-NN. Este método implica entrenar el modelo con varios valores de k y evaluar el rendimiento del modelo

en cada caso, generalmente usando la métrica de error cuadrático medio (MSE) en el caso de regresión o la tasa de error de clasificación (1- *accuracy*). La gráfica resultante mostrará la tasa de error (eje y) en función del número de vecinos (eje x). El "codo" de la gráfica es el punto donde la tasa de error deja de disminuir significativamente con el aumento de k. Este punto es una buena elección para el valor de k y se puede usar para entrenar el modelo k-NN de manera más efectiva.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Cargar el dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Definir el rango de valores de k a evaluar
n=21
k_range = range(1, n, 2) # en saltos de 2 (solo valores impares)
error_rates = []

# Evaluar el modelo para cada valor de k
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k, p=2, weights='distance')
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    error = 1 - accuracy_score(y_test, y_pred)
    error_rates.append(error)

# Graficar la tasa de error para cada valor de k
plt.figure(figsize=(10, 6))
plt.plot(k_range, error_rates, marker='o', linestyle='--', color='b')
plt.title('Elbow method for selecting k in k-NN')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Error Rate')
```

```
plt.xticks(np.arange(1, n, 1))
plt.grid()
plt.show()
```

Finalmente, aunque K-NN es un modelo fundamental en la caja de herramientas de análisis predictivo, como cualquier otra técnica, presenta tanto ventajas como desventajas que se resumen a continuación:

Ventajas de K-NN:

Fácil de entender e implementar: Una de las mayores ventajas de K-NN es su simplicidad. El algoritmo es intuitivo y fácil de implementar, y no requiere entrenamiento explícito, lo que lo convierte en un método perezoso (lazy learning).

Flexible a la función de la distancia: K-NN puede trabajar con cualquier función de distancia, lo que lo hace adecuado para contextos donde la noción de distancia es no estándar o muy específica (como distancia Euclidiana, Manhattan, Minkowski, o incluso medidas personalizadas).

Eficaz con un gran número de clases: K-NN puede ser eficaz cuando hay un gran número de posibles etiquetas de salida; su capacidad para diferenciar no está limitada por las características de los datos.

Desventajas de K-NN:

Alto costo computacional: K-NN puede ser muy lento en bases de datos grandes porque cada nueva predicción requiere un cálculo de distancia a cada uno de los puntos de entrenamiento, lo cual puede ser computacionalmente costoso.

Alto uso de memoria: Dado que K-NN es un algoritmo perezoso (lazy learning) y no genera un modelo explícito, necesita almacenar todo el conjunto de datos de entrenamiento para hacer predicciones, lo que puede resultar en un alto uso de memoria.

Sensible a variables irrelevantes: K-NN puede ser muy sensible a características irrelevantes o redundantes porque todas las características contribuyen aproximadamente por igual a la distancia entre las muestras.

Sensible a datos no balanceados: En un conjunto de datos donde las clases no están equitativamente representadas, K-NN puede estar sesgado hacia las clases más frecuentes.

Dificultad con clases overlapping: K-NN puede tener dificultades cuando los límites de decisión entre clases son borrosos. Su desempeño puede degradarse significativamente si las clases se superponen considerablemente.

Efecto de la maldición de la dimensionalidad: A medida que el número de características aumenta, el espacio de características se vuelve exponencialmente grande, lo que diluye la efectividad de la distancia Euclidiana y puede hacer que K-NN actúe de manera subóptima.

Conclusiones

En esta lectura se ha revisado los fundamentos del algoritmo k-Nearest Neighbors (k-NN) para abordar problemas de clasificación binaria utilizando Python y TensorFlow-Keras. K-NN es un algoritmo potente y flexible que se adapta bien a una variedad de problemas de clasificación y regresión. Es crucial tener en cuenta sus limitaciones, especialmente en términos de eficiencia y rendimiento en grandes volúmenes de datos o datos con muchas características. A menudo, técnicas como la reducción de dimensiones, la selección de características, y el ajuste del número de vecinos (k) son necesarias para mejorar su eficacia y eficiencia.

Por lo que, se motiva a los estudiantes a revisar material y ejemplos adicionales disponibles libremente en Internet para profundizar en la comprensión del algoritmo y su aplicación eficaz en diferentes problemas de variada complejidad.

Bibliografía:

Berzal, Fernando (2019). Redes neuronales & Deep Learning. USA: Independently published.

Gerón, A. (2023). Aprende Machine Learning con Scikit-Learn, Keras y TensorFlow: Conceptos, herramientas y técnicas para conseguir sistemas inteligentes. Tercera Edición. Anaya Multimedia.

Torres, J. (2020). Python Deep Learning: Introducción práctica con Keras y TensorFlow 2. Marcombo.