Preprocesamiento de Datos : Algoritmo KNN

Cielo Coyotl Juan Pablo.

12 diciembre 2023

Índice

1. DESCRIPCIÓN DEL ALGORITMO

1.1. Definición

El algoritmo de las K vecinas más cercanas o K-nearest neighbors (kNN) es un algoritmo de Machine Learning que pertenece a los algoritmos de aprendizaje supervisado ,este algoritmo pueden ser utilizado para resolver problemas de clasificación y de regresión.

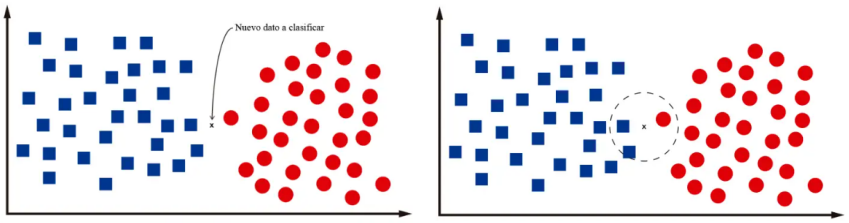
1.2. Lógica

La lógica detrás del algoritmo es una de las más sencillas de todos los algoritmos de Machine Learning supervisados:

- Etapa 1: Seleccionar el número de K vecinas
- Etapa 2: Calcular la distancia desde un punto no clasificado a otros puntos:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{1}$$

- Etapa 3: Tomar las K vecinas más cercanas según la distancia calculada
- Etapa 4: Entre las K vecinas, contar el número de puntos en cada categoría.
- Etapa 5: Atribuir un nuevo punto a la categoría más presente entre las K vecinas.



(a) Dato sin clasificar. (b) Clasificación con K=3.

Este algoritmo es una técnica de clasificación versátil que puede ser útil en muchas situaciones, pero su rendimiento depende de la elección adecuada de parámetros y la naturaleza de los datos. Es importante considerar las ventajas y desventajas al decidir si k-NN es la elección adecuada para un problema de clasificación específico.

2. METODOLOGÍA.

2.1. Cargar datos.

Se crea una función cargar-datos(nombre de archivo) que carga los datos desde un archivo de texto ('golf.txt') y devuelva una lista de instancias de datos.

2.2. Dividir conjuntos de Entrenamiento y Prueba.

Esta función dividir-datos(datos, tamaño-entrenamiento) toma la lista de datos cargados y la divide en conjuntos de entrenamiento y prueba. Define el tamaño del conjunto de entrenamiento y después se utiliza una selección aleatoria para dividir los datos en consecuencia.

2.3. Calcular Distancia Euclidiana.

Se implementa una función distancia-euclidiana(valor1, valor2) que calcula la distancia euclidiana entre dos instancias de datos. Esta función es utilizada para determinar la similitud entre las instancias.

2.4. Obtener Vecinos Cercanos.

Después se utiliza una función predecir-knn (conjunto-entrenamiento, conjunto-prueba, k) que toma el conjunto de entrenamiento, el conjunto de prueba y el valor de k como entrada y devuelve una lista de predicciones para el conjunto de prueba. Para cada instancia en el conjunto de prueba, esta función debe calcular las distancias a todas las instancias en el conjunto de entrenamiento y seleccionar los k vecinos más cercanos. Luego, se determina la clase predominante entre los vecinos cercanos como la "predicción".

2.5. Entrenar y Evaluar.

Aqui se utiliza un bucle para realizar la clasificación y evaluación del modelo varias veces (20 veces) con diferentes divisiones aleatorias de los datos. Esto permitirá obtener una idea del rendimiento promedio del modelo k-NN con los diferentes valores(k).También muestra un gráfico de barras que representa la precisión promedio en función de k.

2.6. Clasificar objetos desconocidos.

La función cargar-segundo-archivo permite al usuario cargar un segundo .txt('golf2') que contendrá objetos sin etiqueta para realizar predicciones utilizando el modelo KNN entrenado con el mejor valor de k.La salida se imprime por consola.

2.7. GUI.

Para usar el programa se implementa una GUI con dos botones que permite al usuario realizar las acciones: 1.- cargar un archivo .txt para entrenar el modelo y 2.- Cargar un segundo archivo .txt para clasificar objetos desconocidos con el modelo previamente entrenado.

3. ENTRENAMIENTO DEL MODELO.

En esta sección, presentamos los resultados obtenidos al entrenar el modelo k-NN con diferentes valores de k en el problema de clasificación de los datos cargados desde el archivo 'golf.txt'.

3.1. Resultados del Entrenamiento.

Se realizaron múltiples ejecuciones del algoritmo k-NN utilizando diferentes valores de k para evaluar su efectividad en la clasificación de los datos. Los resultados se presentan en la siguiente tabla:

Valor de k	**Precisión Promedio (%)**
3	58
5	63
7	56

Cuadro 1: Resultados del entrenamiento con diferentes valores de k.

3.2. Análisis de Resultados.

Observando los resultados obtenidos, podemos realizar las siguientes observaciones:

- Con $k = 5$, se obtiene la mayor precisión promedio, alcanzando un valor del 1.95 %.
- Los valores de $k = 3$ y $k = 7$ también muestran un rendimiento decente, pero son ligeramente inferiores a $k = 5$.

3.3. Selección del Mejor Valor de k.

Basándonos en los resultados obtenidos, sugerimos que el mejor valor de k para este problema es $k = 5$. Este valor proporciona la mayor precisión promedio en la clasificación de los datos y, por lo tanto, es el más adecuado para este conjunto de datos específico.

Es importante tener en cuenta que la elección del valor de k puede variar según el conjunto de datos y el problema. Se recomienda realizar un ajuste y considerar la naturaleza de los datos antes de seleccionar el valor óptimo de k.

4. CLASIFICACIÓN DE DATOS DESCONOCIDOS.

4.1. Datos de entrada sin clasificar.

El segundo data frame contiene 10 objetos , con atributos variados . El programa debe ser capaz de asignarle etiqueta a cada uno de estos objetos haciendo uso del modelo entrenado anteriormente.

Cuadro 2: Datos del conjunto de datos.

Outlook	Temperature	Humidity	Windy
100	78.0	88.0	0
010	85.0	92.0	1
100	72.0	75.0	1
001	68.0	77.0	0
010	90.0	86.0	1
100	76.0	89.0	0
001	62.0	71.0	0
010	80.0	94.0	1
100	75.0	78.0	1
010	88.0	81.0	1

4.2. Salida del programa con las etiquetas de los nuevos objetos.

```
racticas/knn_golf.py"
El mejor valor de k es 5 con una precisión promedio de 60.00%
Predicciones para el segundo archivo con k=5:
['no' 'no' 'yes' 'yes' 'yes' 'no' 'yes' 'no' 'yes' 'yes']
```

5. APENDICES.

5.1. Codigo del programa.

```
import tkinter as tk
from tkinter import filedialog
```

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Variables globales para almacenar los datos de entrenamiento
X_entrenamiento_global = None
y_entrenamiento_global = None

# Función para cargar datos desde un archivo seleccionado en la GUI
def cargar_datos():
    archivo_ruta = filedialog.askopenfilename(filetypes=
    [("Archivos de texto", "*.txt")])
    if archivo_ruta:
        datos = pd.read_csv(archivo_ruta, header=None)
        # Convertir a minúsculas y luego
        reemplazar etiquetas "yes" con 1 y etiquetas "no" con 0
        datos.iloc[:, -1] = datos.iloc[:, -1].str.lower().replace(
        {"yes": 1, "no": 0})
        # Eliminar filas con valores faltantes (NaN) en la última columna
        datos = datos.dropna(subset=[datos.columns[-1]])
        return datos.values
    return None

# Función para dividir los datos en conjuntos de entrenamiento y prueba
def dividir_datos(X, y, tamano_prueba,semilla_aleatoria):
    np.random.seed(semilla_aleatoria)
    n = X.shape[0]
    indices = np.random.permutation(n)
    indice_division = int((1 - tamano_prueba) * n)
    indices_entrenamiento = indices[:indice_division]
    indices_prueba = indices[indice_division:]
    X_entrenamiento, X_prueba = X[indices_entrenamiento], X[indices_prueba]
    y_entrenamiento, y_prueba = y[indices_entrenamiento], y[indices_prueba]
    return X_entrenamiento, X_prueba, y_entrenamiento, y_prueba

# Función para calcular la distancia euclidiana entre dos puntos
def distancia_euclidiana(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))

# Función para predecir la clase de una instancia utilizando el clasificador KNN
def predecir_knn(X_entrenamiento, y_entrenamiento, X_prueba, k):
    y_predicho = []
    for i in range(X_prueba.shape[0]):
        distancias = [(distancia_euclidiana(X_prueba[i], x), y) for
        x, y in zip(X_entrenamiento, y_entrenamiento)]
        distancias.sort(key=lambda x: x[0])
        vecinos_mas_cercanos = distancias[:k]
        etiquetas_vecinos = [vecino[1] for vecino in vecinos_mas_cercanos]
        mas_comun = np.bincount(etiquetas_vecinos).argmax()
        y_predicho.append(mas_comun)
    return np.array(y_predicho)

# Crear la ventana de la GUI
ventana = tk.Tk()
ventana.title("Carga de Datos")
```

```
# Obtener las dimensiones de la ventana
ancho_ventana = ventana.winfo_reqwidth()
alto_ventana = ventana.winfo_reqheight()

# Obtener las dimensiones de la pantalla y calcular el centro
ancho_pantalla = ventana.winfo_screenwidth()
alto_pantalla = ventana.winfo_screenheight()
x = (ancho_pantalla/2) - (ancho_ventana/2)
y = (alto_pantalla/2) - (alto_ventana/2)

# Centrar la ventana en la pantalla
ventana.geometry(f'+{int(x)}+{int(y)}')

# Almacenar las precisiones promedio para cada valor de k
precisiones = []
valores_k = [3, 5, 7]

# Lista para almacenar las precisiones promedio
precisiones_promedio = []

# Función para entrenar el modelo
def entrenar():
    global X_entrenamiento_global, y_entrenamiento_global
    datos = cargar_datos()
    if datos is not None:
        X = datos[:, :-1]
        y = datos[:, -1]
        tamano_prueba=0.2
        semilla_aleatoria=None

        for k in valores_k:
            resultados = []
            for _ in range(20):
                X_entrenamiento, X_prueba, y_entrenamiento, y_prueba =
                dividir_datos(X, y, tamano_prueba, semilla_aleatoria)
                y_predicho = predecir_knn(X_entrenamiento,
                y_entrenamiento, X_prueba, k)
                exactitud = np.mean(y_predicho == y_prueba) * 100
                resultados.append(exactitud)

            promedio_rendimiento = np.mean(resultados)
            precisiones.append(promedio_rendimiento)

            X_entrenamiento_global = X_entrenamiento
            y_entrenamiento_global = y_entrenamiento

        mejor_k = valores_k[np.argmax(precisiones)]
        print(f"El mejor valor de k es {mejor_k}
        con una precisión promedio de {max(precisiones):.2f}%")

        precisiones_promedio.extend(precisiones)

        plt.figure(figsize=(8, 6))
        plt.bar(valores_k, precisiones_promedio, tick_label=valores_k)
```

```
        plt.xlabel("Valor de k")
        plt.ylabel("Precisión Promedio")
        plt.title("Precisión Promedio vs. Valor de k")
        plt.show()

# Función para cargar un segundo archivo y realizar predicciones con
el mejor modelo
def cargar_segundo_archivo():
    archivo_ruta = filedialog.askopenfilename(filetypes=
    [("Archivos de texto", "*.txt")])
    if archivo_ruta:
        datos = pd.read_csv(archivo_ruta, header=None)
        X = datos.values

        mejor_k = valores_k[np.argmax(precisiones)]
        y_predicho = predecir_knn(X_entrenamiento_global,
        y_entrenamiento_global, X, mejor_k)

        etiqueta = np.array(["yes" if i == 1 else "no" for i in y_predicho])

        print(f"Predicciones para el segundo archivo con k={mejor_k}:")
        print(etiqueta)

boton_entrenar = tk.Button(ventana, text="Entrenar Modelo y Graficar",
command=entrenar)
boton_entrenar.pack()

boton_cargar_segundo_archivo = tk.Button(ventana, text="Cargar
Segundo Archivo y Predecir", command=cargar_segundo_archivo)
boton_cargar_segundo_archivo.pack()

ventana.mainloop()
```

5.2. Conceptos Utilizados.

- Distancia Euclidiana :La distancia euclidiana es una medida de distancia en el espacio euclidiano (espacio bidimensional o tridimensional) entre dos puntos. En el contexto del algoritmo k-NN, se utiliza para calcular la similitud o distancia entre dos instancias de datos. La fórmula de la distancia euclidiana entre dos puntos $P(x_1, y_1)$ y $Q(x_2, y_2)$ se define como:

$$DistanciaEuclidiana = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

  Esta fórmula calcula la longitud del segmento de línea recta que conecta los dos puntos en un plano euclidiano. En el programa se usa para medir la similitud entre instancias de datos, donde puntos cercanos tienen una distancia euclidiana pequeña, lo que indica similitud, y puntos distantes tienen una distancia euclidiana grande, lo que indica diferencia.

- División de Conjuntos;la división de conjuntos es el proceso de separar el conjunto de datos original en dos subconjuntos: uno para entrenar el modelo (conjunto de entrenamiento) y otro para probar el modelo (conjunto de prueba).Esta división es crucial para medir la capacidad del modelo para generalizar patrones a nuevos datos no vistos.

- Selección de Vecinos:En k-NN, la selección de vecinos se refiere a la elección de los k puntos de datos más cercanos a una instancia de prueba en función de una métrica de distancia, en este caso la distancia euclidiana.
- Evaluación del Modelo:Es el proceso de medir qué tan bien el modelo k-NN está haciendo predicciones en un conjunto de datos de prueba.Donde la "precisión"del modelo es la proporción de predicciones correctas hechas por el modelo en relación con el total de predicciones realizadas. La precisión es una métrica importante para determinar la efectividad del algoritmo k-NN al clasificar datos.

6. Referencias.

Raschka, S., Mirjalili, V. (2017). Python Machine Learning. Packt Publishing.

Müller, A. C., Guido, S. (2016). Introduction to Machine Learning with Python. O'Reilly Media.