Unidad Profesional Interdisciplinaria de Ingeniería campus Tlaxcala

INSTITUTO POLITÉCNICO NACINAL

FUNDAMENTOS DE INTELIGENCIA ARTIFICIAL.

Algoritmo KNN aplicado a la base de datos Handwritten Digits.

Equipo:

Navil Pineda Rugerio.

Luis Daniel Islas García.

Miguel Ángel Sánchez Zanjuampa

12 de junio, 2023

Resumen

En el presente documento se detalla el proceso para realizar un algoritmo de clasificación utilizando KNN (K vecinos más cercanos), el cual es aplicado a la base de datos ya conocida Handwritten Digits, la cual etiqueta imágenes de 8x8 con clases de números de 0 a 9. Para ello se entrena el modelo, que más adelante se explica, con 1000 datos, y se utilizan 500 datos para prueba.

Palabras clave: Algoritmo de clasificación, KNN, K vecinos más cercanos, handwritten digits dataset .

Abstract

This paper details the process to perform a classification algorithm using KNN (K nearest neighbors), which is applied to the already known database Handwritten Digits, which labels 8x8 images with number classes from 0 to 9. For this purpose, the model, which is explained below, is trained with 1000 data, and 500 data are used for testing.

Keywords: classification algorithm, KNN, K nearest neighbors, handwritten digits dataset .

I. MARCO TEÓRICO.

El algoritmo KNN, conocido como K-Nearest Neighbors o Vecinos más Cercanos, es un método de clasificación y regresión ampliamente utilizado en el campo del aprendizaje automático (machine learning). Se basa en el principio de que los puntos de datos con características similares tienden a agruparse en el espacio.

El algoritmo KNN no hace suposiciones explícitas sobre la forma funcional de los datos, por su parte, su objetivo principal es clasificar un punto de datos desconocido basándose en la información de sus vecinos más cercanos en un espacio de características.

Para clasificar un nuevo punto de datos, se calcula la distancia entre ese punto y todos los demás puntos de datos en el conjunto de entrenamiento.

La elección de una medida de distancia adecuada es fundamental para el funcionamiento del algoritmo KNN. Las medidas de distancia comunes incluyen la distancia euclidiana, la distancia de Manhattan y la distancia de Minkowski. La elección de la fórmula de distancia depende del problema y de las características de los datos.

El valor K en KNN se refiere al número de vecinos más cercanos que se utilizan para tomar una decisión de clasificación. La elección de K es un hiperparámetro crítico y puede afectar significativamente el rendimiento del algoritmo. Si se elige un valor K más pequeño puede llevar a una clasificación más sensible al ruido, mientras que un valor K más grande puede generar una frontera de decisión más suave pero con una mayor complejidad computacional.

Una vez que se han identificado los K vecinos más cercanos, el algoritmo KNN realiza una clasificación o una regresión.

Para el caso de esta práctica se lleva a cabo una clasificación, en la que se asigna la etiqueta más común entre los vecinos al punto de datos desconocido

Objetivo

El objetivo de esta práctica es diseñar un algoritmo KNN aplicado a la base de datos de dígitos escritos a mano, con el fin de clasificar nuevos dígitos desconocidos, es decir que no se encuentran en el entrenamiento, y asignarle una etiqueta.

II. MATERIALES

- Bibliotecas y herramientas para leer los datos de un archivo, para el manejo de arreglos y el ploteo de las imágenes.
- Funciones y clases para la lectura del archivo y el algoritmo KNN.
- Jupyter Notebook.

III. DESARROLLO

A. Librerías

Se importan las librerías, utilizamos numpy para las operaciones entre los vectores de datos, matplotlib para mostrar las imágenes, pandas para mostrar la información, y random para aleatorizar los datos leidos del archivo.

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

B. Clase para lectura del archivo.

Para cargar la información del archivo se creó una clase que tiene un método para leer el archivo y guardar la información en una matriz que contiene todas las imágenes, y un vector que contiene todas las clases correspondientes a las imágenes.

El constructor de la clase no recibe ningún parámetro, y sus atributos se inicializan en None, para que en los métodos posteriores se guarde la información cargada del archivo. También se declaran los métodos setters y getters que devuelven los datos ya aleatorizados.

```python
# Leer el archivo de texto
class Data:
    def __init__(self):
        self._data = None
        self._target = None
        self._randomData = None
        self._randomTarget = None

    # Metodos getters
    def getRandomData(self):
        return self._randomData

    def getRandomTarget(self):
        return self._randomTarget
```

Para cargar el archivo se tiene el método load_data, el cual lee el archivo de texto plano y lo separa por líneas, separando cada token cuando encuentra un coma (,) cada una de las líneas del archivo ya sin las comas se guarda en una lista llamada values, esta lista contiene el vector de las imágenes y en la última posición contiene la clase a la que pertenece.

```python
# Funcion para leer el archivo y guardar las imagenes y sus clases
def load_data(self):
    with open("optdigits.tes", "r") as datos:
        valores = []
        for linea in datos:
            valores.append([x for x in linea.strip().split(",")])

        # Crear matriz para guardar los valores de data
        data = np.zeros((len(valores), 64))
        for i in range(len(valores)):
            data[i] = valores[i][:64]

        self._data = np.copy(data)

        # Crear matriz para guardar los valores de target (clases)
        target = np.zeros((len(valores)))
        for i in range(len(valores)):
            target[i] = valores[i][64]

        self._target = np.copy(target)
```

Para guardar la información de las imágenes y las clases se guarda en una matriz llamada data los valores de values de la posición 0 a la posición 64, lo que corresponde a la imagen, y para las clases, se guardan en un arreglo los valores en la posición 65, que corresponden al número que representa la imagen.

C. Aleatorizar Datos.

Para aleatorizar los datos ya guardados se tiene el método randomizer.

```python
# Funcion para aleatorizar la informacion guardada
def randomizer(self):
    # Crear matriz auxiliar para permutar filas de la matriz
    numDatos, numChar = self._data.shape

    randomData = np.copy(self._data)
    # Crear arreglo auxiliar para permutar clases
    numTarget = self._target.shape

    randomTarget = np.copy(self._target)

    # Tomar un valor random desde 0 hasta numDatos y mezclar
    a, b = shuffle(randomData, randomTarget)

    # Guardar la matriz con filas intercambiadas y el arreglo
    self._randomData = np.copy(a)
    self._randomTarget = np.copy(b)
    print("Datos aleatorizados")
```

Primero, se obtienen las dimensiones del arreglo "data" utilizando la función "shape". Esto permite conocer el número de datos y de características (o columnas) de la matriz. A continuación, se crea un arreglo nuevo llamado "randomData" utilizando la función "np.copy()", que crea una copia en memoria del arreglo original.

Lo mismo se hace con el arreglo "target", excepto que aquí sólo se necesita conocer el número de filas.

Después se utiliza la función shuffle que será explicada a continuación.

```
def shuffle(data, target):
    # Hacer un shuffle de las listas
    dataRand, targetRand = shuffle_lists(data, target)
    return dataRand, targetRand
```

Esta función transforma las copias de la matriz y el arreglo en listas y devuelve los datos mezclados tanto de las imágenes, como de las clases. Para ello también se utiliza la función shuffle_lists.

```
def shuffle_lists(lst, lst2):
    n = len(lst2)
    for i in range(n - 1, 0, -1):
        j = random.randint(0, i)
        lst[i], lst[j] = np.copy(lst[j]), np.copy(lst[i])
        lst2[i], lst2[j] = lst2[j], lst2[i]
    return lst, lst2
```

Esta función obtiene un número aleatorio de 0 a n, donde n es la longitud de la lista de datos, y se van intercambiando los datos tanto de las imágenes como de las clases, finalmente se devuelven esas dos listas, que ya son las listas aleatorizadas.

D. Leer la información del archivo.

Para cargar el archivo, instanciamos una variable de tipo de la clase Data, y llamamos al método load_data. Después se llama al método randomizer para aleatorizar los datos ya guardados.

```
data = Data()
data.load_data()
data.randomizer()
```

```
Se cargaron los datos
Datos aleatorizados
```

Para obtener los datos ya aleatorizados se utiliza el método getRandomData, y se muestran los resultados, para este caso solo son las primeras dos imágenes.

```
# Devuelve los imagenes aleatorias con las que se va a trabajar
datos = data.getRandomData()
datos[:2]
```

```
array([[ 0.,  0.,  1., 12.,  7.,  0.,  0.,  0.,  0.,  0.,  8., 14.,  1
         0.,  0.,  0.,  0.,  0., 15.,  7.,  0.,  3.,  5.,  0.,  0.,  5
        16.,  0.,  4., 15.,  4.,  0.,  0.,  5., 16., 16., 16., 15.,  2
         0.,  0.,  0., 11., 12., 16.,  8.,  0.,  0.,  0.,  0.,  0.,  5
        16.,  3.,  0.,  0.,  0.,  0.,  0., 13., 10.,  0.,  0.,  0.],
       [ 0.,  0.,  5., 16., 14.,  2.,  0.,  0.,  0.,  1., 13., 14., 16
         8.,  0.,  0.,  0.,  9., 15.,  3., 16.,  5.,  0.,  0.,  0., 16
        13.,  3., 16.,  3.,  0.,  0.,  0.,  3.,  3., 11., 13.,  0.,  6
         0.,  0.,  0.,  0., 13., 10.,  0.,  0.,  0.,  0.,  0.,  2., 16
        16., 16., 10.,  0.,  0.,  0.,  6., 16., 14., 12.,  9.,  0.]])
```

También se muestran los resultados de las clases aleatorizadas.

```
# Devuelve las clases de las imagenes aleatorizadas
target = data.getRandomTarget()
target
```

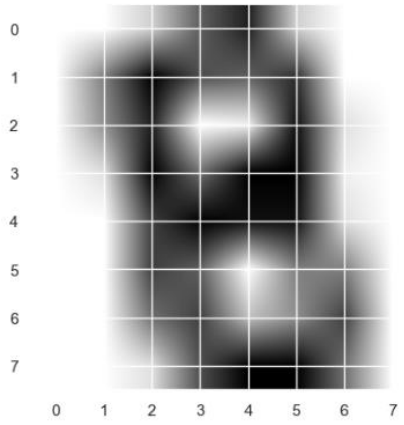
```
array([4., 2., 9., ..., 1., 4., 9.])
```

Después de haber aleatorizado los datos, es común usar algunas medidas para comprobar que la aleatoriedad fue correcta. En este caso, se muestra una imagen y la clase correspondiente para verificar que los datos aleatorios no hayan afectado la correspondencia entre datos y objetivos.

Por ejemplo, para un problema de clasificación se podría seleccionar una imagen aleatoria y su clase correspondiente antes y después de la aleatorización, comprobando que la clase sigue siendo la misma. Si la aleatoriedad no fue correcta, podrían existir errores en la correspondencia entre datos y objetivos, lo que podría impactar negativamente en el rendimiento del modelo de aprendizaje automático.

```
# Ejemplo
num = random.randint(0, 1700)
datos_res = np.reshape(datos[num], (8,8))
print(f"Numero{target[num]}")
plt.imshow(datos_res, cmap=plt.cm.gray_r, interpolation="bilinear")
```

```
Numero8.0
<matplotlib.image.AxesImage at 0x21aa2205220>
```

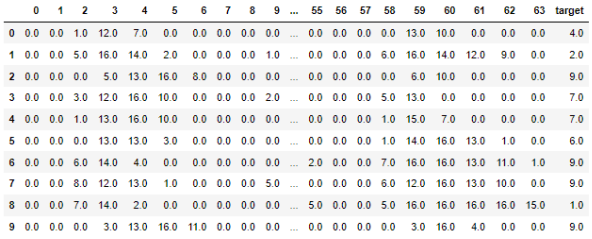


Como se observa la aleatoriedad fue correcta, pues clasifica al 8 como 8.

E. Data Frame para observar los datos.

Después se utiliza un Data Frame de la librería pandas para mostrar los datos en una tabla, se muestran los valores de los 64 pixeles de las imágenes y las clases a las que corresponden.

```
# Dataframe para visualizar los datos
df = pd.DataFrame(datos)
df["target"] = target
df.head(10)
```



	0	1	2	3	4	5	6	7	8	9	...	55	56	57	58	59	60	61	62	63	target
0	0.0	0.0	1.0	12.0	7.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	13.0	10.0	0.0	0.0	0.0	4.0
1	0.0	0.0	5.0	16.0	14.0	2.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	6.0	16.0	14.0	12.0	9.0	0.0	2.0
2	0.0	0.0	0.0	5.0	13.0	16.0	8.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	6.0	10.0	0.0	0.0	0.0	9.0
3	0.0	0.0	3.0	12.0	16.0	10.0	0.0	0.0	0.0	2.0	...	0.0	0.0	0.0	5.0	13.0	0.0	0.0	0.0	0.0	7.0
4	0.0	0.0	1.0	13.0	16.0	10.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	15.0	7.0	0.0	0.0	0.0	7.0
5	0.0	0.0	0.0	13.0	13.0	3.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	14.0	16.0	13.0	1.0	0.0	6.0
6	0.0	0.0	6.0	14.0	4.0	0.0	0.0	0.0	0.0	0.0	...	2.0	0.0	0.0	7.0	16.0	16.0	13.0	11.0	1.0	9.0
7	0.0	0.0	8.0	12.0	13.0	1.0	0.0	0.0	0.0	5.0	...	0.0	0.0	0.0	6.0	12.0	16.0	13.0	10.0	0.0	9.0
8	0.0	0.0	7.0	14.0	2.0	0.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	5.0	16.0	16.0	16.0	16.0	15.0	1.0
9	0.0	0.0	0.0	3.0	13.0	16.0	11.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	3.0	16.0	4.0	0.0	0.0	9.0

Una vez visualizados, ya podemos preparar los datos que se van a utilizar para el entrenamiento y para el test del modelo. Para ello se dividide la información en dos grupos, uno que contiene 1000 datos, este es para el entrenamiento, y 5000 datos, para el test. Y como se utilizan los Data Frames para separar la información, en el caso de X_train y X_test se eliminan las columnas que tienen el valor “target”, pues solo queremos las imágenes y no las clases, y para y_train y y_test solo se guardan precisamente las target o clases.

```
# Se separan los datos de prueba (500) y los datos
df_train, df_test = df[:1000], df[1000:1500]

# Se preparan los datos de train y los datos de tes
X_train = df_train.drop(columns=["target"]).values
y_train = df_train["target"].values

X_test = df_test.drop(columns=["target"]).values
y_test = df_test["target"].values
```

Es importante realizar esta división para poder evaluar el desempeño del modelo de aprendizaje automático. Al dividir los datos en dos grupos, se evita que el modelo aprenda solo de los datos de entrenamiento, lo que podría llevar a un sobreajuste del modelo. Además, utilizar un conjunto de datos de prueba independiente permite evaluar la precisión y el rendimiento del modelo en datos no vistos previamente.

F. Modelo KNN

Una vez listos los datos, se explicará el modelo de KNN para hacer predicciones con esos datos.

Para este modelo se crea una clase, la cual recibe el parámetro k, que representa el número de vecinos más cercanos que se quiere tomar.

```
class knnAlgorithm:
    def __init__(self, k):
        self.k = k
```

También se tiene un método para guardar la información con la que se entrena el modelo.

```
def fit(self, X, y):
    self.X_train = X # Imagenes
    self.y_train = y # Clases
```

Después se crea el método principal, el más importante, pues en esta parte se realiza la predicción.

En este método recibe como parámetro la imagen que se quiere clasificar, primero se inicializa una lista que contendrá el valor de la predicción final. Para hacer esto, siguiendo con el algoritmo de KNN, se itera sobre cada imagen del entrenamiento y se compara la distancia euclideana con la imagen pasada para predecir, para esto se tiene la función de distancia.

De forma detallada, la función imprime un mensaje indicando que se están preparando las predicciones. A continuación, se utiliza un bucle for para recorrer los elementos del conjunto de datos X y se inicializa un arreglo vacío para almacenar las predicciones correspondientes, Para cada elemento de X, se calcula la distancia euclidiana entre ese elemento y cada uno de los elementos del conjunto de entrenamiento (self.X_train). Este cálculo se realiza utilizando una función llamada euc_dist.

```
# Funcion para calcular la distancia euclideana
def euc_dist(x1, x2):
    return np.sqrt(np.sum((x1-x2)**2))
```

Todas las distancias calculadas se guardan en un arreglo de distancias, de este arreglo se ordenan y se obtienen las primeras k distancias, estas son guardadas en sort_distance, a continuación, se buscan las clases de las distancias en el arreglo de y_train, que son las clases de entrenamiento. Como ya se tienen los primeros k vecinos con sus clases, se decide cual es la clase predominante, esto se hace con la función np.unique que devuelve las clases únicas y el número de veces que se repite esa clase, ahora con la función np.argmax se obtiene la clase que tiene la mayor

cantidad de repeticiones, esta ya será la clase que se predijo.

```
def predict(self, X):
    print("Preparando predicciones")
    y_pred = []

    # Obtener la distancia a todos los puntos
    for i in range(len(X)):
        distances = np.array([euc_dist(X[i], x_t) for x_t in self.X_t

        # Ordenar las distancias y devolver solo los primeros k vecin
        sort_distance = np.argsort(distances)[: self.k]

        # Guardar las distancias en un arreglo
        knn = self.y_train[sort_distance]

        # Contar las clases más coumnes
        label, counts = np.unique(knn, return_counts = True)

        y_pred.append(label[np.argmax(counts)])
    return np.array(y_pred)
```

IV. Resultados

Ahora se puede utilizar el algoritmo para hacer predicciones con los datos de test. Pero primero se puede comprobar la precisión del algoritmo con los mismos datos de entrenamiento.

Para ello se entrena el inicializa la variable model de tipo knnAlgorithm con un k=3, y se entrena el modelo llamando al método fit.

```
model = knnAlgorithm(k=3)
model.fit(X_train, y_train)
```

Ahora se utiliza el método predict y se pasan los datos de las imágenes de entrenamiento, es decir X_train, y se muestran las primeras 10 predicciones.

```
y_pred = model.predict(X_train)
y_pred[:10]
```

```
Preparando predicciones

array([4., 2., 9., 7., 7., 6., 9., 9., 1., 9.])
```

Ahora se muestran las primeras 10 clases reales, y se comparan para ver que tan atinado fue el modelo.

```
y_train[:10]
```

```
array([4., 2., 9., 7., 7., 6., 9., 9., 1., 9.])
```

```
mask = y_pred == y_train
mask[:10]
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True])
```

Y se calcula el porcentaje de precisión del modelo.

```
model_accuracy = np.mean(y_pred == y_train)
print(f"Precisión del modelo: {model_accuracy * 100:.2f}%")
```

```
Precisión del modelo: 99.00%
```

Ahora se hacen las pruebas con los datos de test, y se realiza el mismo procedimiento anterior para calcular el porcentaje de precisión.

Prueba con datos de test

```
y_pred = model.predict(X_test)
y_pred[:100]
```

```
Preparando predicciones

array([5., 6., 7., 4., 4., 9., 5., 7., 2., 8., 1., 2., 4., 6., 9., 9., 3.,
       0., 6., 6., 1., 0., 3., 6., 6., 1., 0., 3., 1., 7., 6., 5., 5., 2.,
       1., 2., 5., 5., 0., 8., 1., 6., 7., 1., 7., 0., 4., 3., 8., 0., 5.,
       1., 2., 7., 4., 8., 6., 6., 9., 9., 2., 7., 1., 2., 3., 3., 7., 7.,
       5., 7., 5., 9., 9., 4., 9., 6., 6., 6., 5., 0., 0., 9., 3., 5., 6.,
       0., 3., 4., 0., 5., 0., 2., 8., 8., 8., 8., 3., 4., 0., 7.])
```

```
y_test[:100]
```

```
array([5., 6., 7., 4., 4., 9., 5., 7., 2., 8., 1., 2., 4., 6., 9., 9., 3.,
       0., 6., 6., 1., 0., 3., 6., 6., 1., 0., 3., 1., 7., 6., 5., 5., 2.,
       1., 2., 5., 5., 0., 8., 1., 6., 7., 1., 7., 0., 4., 3., 8., 0., 5.,
       1., 2., 7., 4., 8., 6., 6., 9., 9., 2., 7., 1., 2., 3., 3., 7., 7.,
       5., 4., 5., 9., 9., 4., 9., 6., 6., 6., 5., 0., 0., 9., 3., 5., 6.,
       0., 3., 4., 0., 5., 0., 2., 8., 8., 8., 8., 3., 4., 0., 7.])
```

```
mask = y_pred == y_test
mask[:100]
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True, False,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True])
```

La precision del modelo para los datos de test fue:

```
model_accuracy = np.mean(y_pred == y_test)
print(f"Model accuracy: {model_accuracy * 100:.2f}%")
```

```
Model accuracy: 98.80%
```

Finalmente, el modelo tuvo una precisión de 98.80%

V. CONCLUSIONES

En esta práctica se aplicó el algoritmo KNN (K-Nearest Neighbors) a la base de datos de dígitos escritos a mano con el objetivo de clasificar y reconocer patrones en imágenes de dígitos numéricos. Los resultados obtenidos demuestran la eficacia y versatilidad del algoritmo KNN en este contexto.

Al utilizar la información de los vecinos más cercanos en el espacio de características, el algoritmo pudo realizar clasificaciones precisas, y al ser aprendizaje supervizado se pueden observar los errores y los aciertos en la clasificación.

Es importante destacar que, si bien el algoritmo KNN es relativamente sencillo de implementar y entender, puede ser computacionalmente costoso para grandes conjuntos de datos, ya que requiere calcular las distancias entre el punto de datos desconocido y todos los demás puntos de datos en el conjunto de entrenamiento, sin embargo, para aplicaciones como estas puede resultar eficiente.

REFERENCIAS

[1] IBM. (s. f.). K-Nearest Neighbors (KNN). Recuperado de https://www.ibm.com/mx-es/topics/knn

[2] *Descubra el algoritmo KNN : un algoritmo de aprendizaje supervisado*. (2021, diciembre 28). Formation Data Science | Datascientest.com. https://datascientest.com/es/que-es-el-algoritmo-knn

[3] (S/f). Medium.com. Recuperado el 13 de junio de 2023, de https://medium.com/analytics-vidhya/a-beginners-guide-to-knn-and-mnist-handwritten-digits-recognition-using-knn-from-scratch-df6fb982748a