Instituto Politecnico Nacional

Escuela Superior de Computo

Ingeniería en Sistemas Computacionales

Sexto semestre

Grupo 6CV3

Inteligencia Artificial

Practica 9: Clasificador euclidiano y 1NN

Presentado por:

Flores Vicencio Marco Isaí

Pérez Gonzales Jesús

Reyes Núñez Sebastián

Fecha: 12/11/2024

Índice

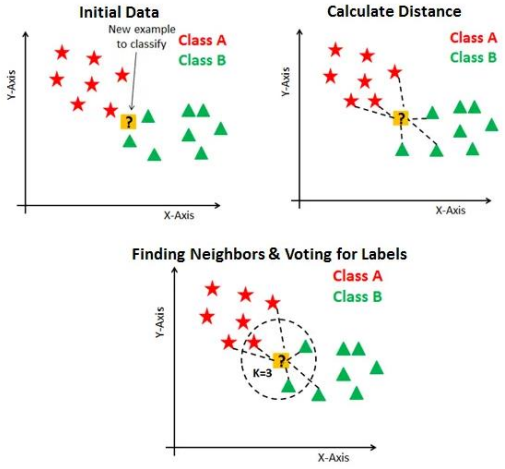
Introducción

El sistema de *clasificación euclidiano* (Euclidean Distance Classifer) es un tipo de algoritmo de Machine Learning que se usa para clasificar tipos de información en diferentes categorías. Hace uso de la distancia euclidiana para clasificar información y encontrar similitudes. Toma un punto dentro del conjunto de datos y calcula la distancia tomando como referencia otro punto determinado. Según el valor de la distancia, el algoritmo es capaz de asignar una etiqueta en el punto de referencia mas cercano. Es un sistema que permite determinar que puntos de datos están más cercanos entre sí y, por tanto, son similares.

Por otro lado, el sistema de *clasificación KNN* (K – Nearest Neighbors) es un método de clasificación no paramétrico. Se basa en la proximidad entre resultados para realizar clasificaciones.

La etiqueta se asigna según un voto mayoritario, es decir, la etiqueta que se presenta con más frecuencia entre los puntos de datos. Hace uso de la distancia euclidiana para realizar el cálculo de los puntos mas cercanos al punto de referencia.

Después de realizar el calculo de distancias, se toman los K vecinos mas cercanos, y el nuevo punto se clasifica según la clase mayoritaria entre los K vecinos.

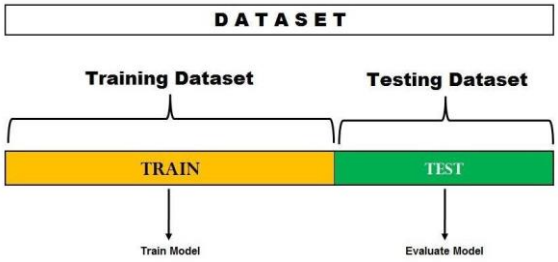


I.Clasificación KNN

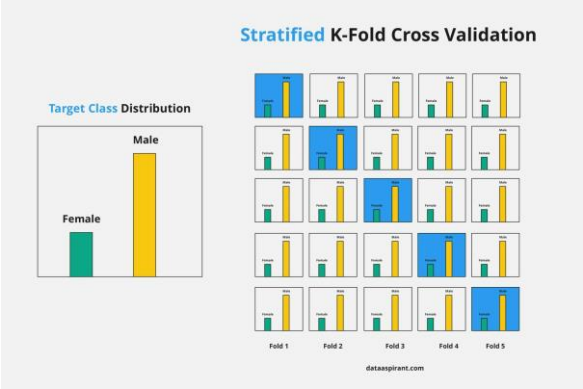
Los sistemas de validación permiten definir si un modelo de Machine Learning se correctamente entrenado. Son métodos estadísticos que permiten estimar el rendimiento de los modelos. Entre los sistemas de validación se encuentra:

- ***Hold Out:*** También conocido como train-test Split. Divide la completitud del data set de manera aleatoria entre un set de entrenamiento y un set de validación. Comúnmente se utiliza el 70% del data set para conformar el set de entrenamiento y el 30% restante para conformar el set de validación.

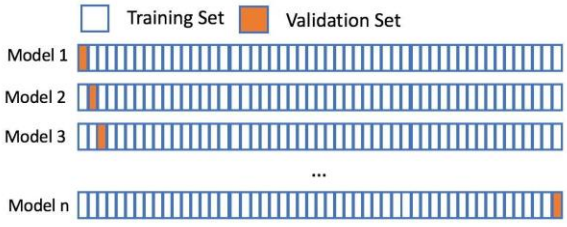
- ***K – Fold Cross:*** El data set es dividido en K conjuntos de igual tamaño. Cada conjunto tiene el mismo ratio de instancias de variables objetivo que se encuentran dentro del dataset. De este modo, es posible aplicar este este sistema de validación para dataset cuya información no esta equilibrada.
- ***Leave One – Out:*** Un punto del data set es utilizado como set de validación y el conjunto restante que forma el data set *(n-1)* es utilizado como el set de entrenamiento. El proceso se repite hasta que cada instancia del data set es usada como set de validación.



II. Hold-Out Validation



III. K - Stratified k-fold cross Validation



IV. Leave One Out Validation

Desarrollo

Considerando los parámetros de la práctica, es necesario realizar la siguiente aclaración: Se podría considerar la distancia euclidiana y el algoritmo KNN como métodos de clasificación distintos. Sin embargo, existe una dependencia clara, pues el algoritmo KNN puede hacer uso de la distancia euclidiana para encontrar la distancia entre los K puntos mas cercanos al punto de referencia. Con lo anterior, se pretende exponer que la clasificación euclidiana es igual al algoritmo KNN cuando K = 1, pues se utiliza un único vecino, el mas cercano.

Por lo anterior, el desarrollo de esta práctica se mostrará de la siguiente manera: Se utilizará un único método de clasificación (1NN) con los tres métodos de validación (Hold – Out, 10 Fold Cross y Leave One Out). Para cada método de validación se mostrará el código fuente utilizado y los resultados para cada uno de los data sets utilizados.

Los data sets utilizados son:

1. Iris plant (archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data)
2. Pima Indias Diabetes (raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv)
3. Wheat Seeds (archive.ics.uci.edu/ml/machine-learning-databases/00236/seeds_dataset.txt)

Hold – Out (70 - 30)

- ***Método de clasificación:*** 1NN
- ***Método de validación:*** Hold – Out (70-30)
- ***Dataset:*** Iris plant

```
import numpy as np      # Librería para cálculo numérico de matrices y vectores
import pandas as pd     # Libreriía de análisis de datos. Trabajar con hojas de cálculo
from sklearn.model_selection import train_test_split        # Dividir un conjunto de datos, para el método Hold - Out
from sklearn.metrics import accuracy_score, confusion_matrix    # Cálculo del accuracy y la matriz de confusión
from scipy.spatial import distance      # Cálculo de distancias, para la distancia Euclidiana
```

Se establecen las bibliotecas a utilizar. La primera corresponde a *numpy*, una librería utilizada para realizar cálculos numéricos, mientras que a biblioteca *panda* sirve para el manejo de datos, en este caso, para los data sets. Finalmente, las bibliotecas *sklearn* y *scipy* son necesarias para implementar el método Hold – Out, calcular el accuracy y la matriz de confusión, así como calcular la distancia euclidiana.

```
# Cargar dataset desde el archivo .data
file_path = 'C:/Users/marco/Downloads/iris/iris.data'

# Cargar el archivo .data sin encabezados, y asumir que la última columna es la clase
column_names = ["sepal_length", "sepal_width", "petal_length", "petal_width", "class"] # Nombre de columnas del dataset
data = pd.read_csv(file_path, header=None, names=column_names) # Lectura del archivo con panda y convertir en DataFrame, sin encabezados

data.dropna(inplace=True) # Eliminar fila que tenga valores NaN, es decir, vacios. Modifica el dataframe

# Separar las características (X) y las etiquetas (y)
X = data.iloc[:, :-1].values    # Todas las columnas excepto la ultima, que pertenece a la clase. Convierte en array
y = data.iloc[:, -1].values     # Etiquetas, solo la ultima columna. Convierte en array

# Dividir el dataset en 70% para entrenamiento y 30% para prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42) # Random para ser reproducible
```

Posteriormente, se realiza la declaración de la ruta donde se encuentra guardado el archivo .data de nuestro data set, en este caso, el data set de *iris plant*. Se definen los nombres de las

columnas que corresponden al data set y mediante la biblioteca *panda* se realiza la lectura del archivo, haciendo uso de los nombres de las columnas y especificando que el archivo del data set no cuenta con encabezado. Posteriormente, con el método *dropna*, se elimina cualquier fila que no cuente con ningún dato con el fin de mantener el orden de la información.

Posteriormente, el data set es dividido de tal forma que, en la variable *X*, se guarda la información de las columnas de características, mientras que en la variable *y* se guarda únicamente la columna de etiquetas, es decir, la que especifica las clases.

Finalmente, mediante el método *train_test_split* se divide el data set para crear dos conjuntos, el conjunto de entrenamiento y el de validación, conformados por el 70% y el 30%, respectivamente. Esta diferenciación se guarda en nuevas variables definidas como *X_train*, *X_test*, *y_train* y *y_test.*

```
# Función para predecir la clase de un punto del data set con 1-NN
def knn_classifier(X_train, y_train, X_test): # Conjunto de entrenamiento y conjunto de validación
    predictions = []        # Lista para almacenar predicciones
    for test_point in X_test:
        distances = [distance.euclidean(test_point, train_point) for train_point in X_train]    # Cálc
        nearest_index = np.argmin(distances) # Índice del punto de entrenamiento que esta proximo
        predictions.append(y_train[nearest_index])  # Usar el punto mas cercano para obtener clase
    return np.array(predictions)

# Predecir las clases en el set de entrenamiento y el set de validación
y_pred = knn_classifier(X_train, y_train, X_test)
```

Después, se define la función *knn_classifier*, que se encarga de realizar la clasificación y predecir los puntos del data set haciendo uso del método 1NN, tomando como parámetros el conjunto de entrenamiento y los datos del conjunto de validación. Se calcula la distancia euclidiana entre un punto de prueba y el resto de los puntos de entrenamiento. Una vez calculadas las distancias, se toma el índice del punto de entrenamiento mas cercano, que es utilizado para realizar las predicciones y obtener la clase del punto.

Dentro de la variable *y_pred* se almacena el resultado de las predicciones, haciendo uso de la función *knn_classifier*.

```
# Cálculo del Accuracy
accuracy = accuracy_score(y_test, y_pred)   # Función de sklearn con las clases del conjunto de entrena
print(f"Accuracy: {accuracy:.2f}")

# Matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)  # Función de sklearn para crear la matriz de confusión
print("Matriz de Confusión:")
print(conf_matrix)
```

Finalmente, se calcula el *accuracy* del modelo haciendo uso de las etiquetas del conjunto de entrenamiento y de la predicción. Del mismo modo, se calcula la matriz de confusión con la misma información.

Resultados:

```
Accuracy: 1.00
Matriz de Confusión:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

Después de ejecutar el código, se obtiene un *accuracy* del 100% y se muestra la matriz de confusión. La matriz de confusión está conformada por 45 elementos (Set de validación): Se predijeron correctamente 19 elementos pertenecientes a la primera clase, sin falsos-positivos ni falsos-negativos. Del mismo modo, se predijeron correctamente 13 elementos pertenecientes a la segunda clase, sin falsos – positivos ni falsos – negativos. Finalmente, se predijeron correctamente 13 elementos pertenecientes a la tercera clase, sin falsos – positivos ni falsos – negativos.

- ***Método de clasificación:*** 1NN
- ***Método de validación:*** Hold – Out (70-30)
- ***Dataset:*** Pima Indians Diabetes

Se realizan los siguientes cambios respecto al código utilizado anteriormente:

```
# Cargar el dataset de Pima Indians Diabetes
file_path = 'C:/Users/marco/Downloads/pima-indians-diabetes.csv'
column_names = [
    "pregnancies", "glucose", "blood_pressure", "skin_thickness", "insulin",
    "bmi", "diabetes_pedigree_function", "age", "outcome"
]
data = pd.read_csv(file_path, header=None, names=column_names)

# Convertir todas las columnas a valores numéricos
data = data.apply(pd.to_numeric, errors='coerce')

data.dropna(inplace=True)

#Imprimir las primeras columnas y filas
print(data.head())
```

Se cambia la ruta donde se encuentra el data set, del mismo modo, el array que contiene el nombre de las columnas es modificado. La lectura del archivo se realiza de la misma manera, pero ya que el data set se conforma de distinta información, es necesario convertir algunas columnas a valores numéricos, para que el método pueda funcionar. Además, se imprimen las primeras columnas y filas del data set.

Resultado:

```
   pregnancies  glucose  blood_pressure  skin_thickness  insulin   bmi  diabetes_pedigree_function  age  outcome
0            6      148              72              35        0  33.6                       0.627   50        1
0            6      148              72              35        0  33.6                       0.627   50        1
1            1       85              66              29        0  26.6                       0.351   31        0
2            8      183              64               0        0  23.3                       0.672   32        1
3            1       89              66              23       94  28.1                       0.167   21        0
4            0      137              40              35      168  43.1                       2.288   33        1
Accuracy: 0.69
Matriz de Confusión:
[[113  38]
 [ 34  46]]
```

Para este caso, el método de validación Hold – Out obtuvo una precisión del 69%, es decir, el 69% de los datos del conjunto de validación fueron clasificados correctamente, mostrando un desempeño de clasificación moderado.

Por otro lado, ya que la predicción de clases solo permite obtener dos valores (0 y 1), se obtiene una matriz de confusión de 2x2. En este caso, se obtuvieron 113 verdaderos negativos (sin diabetes) , 38 falsos positivos, 34 falsos negativos y 46 verdaderos positivos (con diabetes).

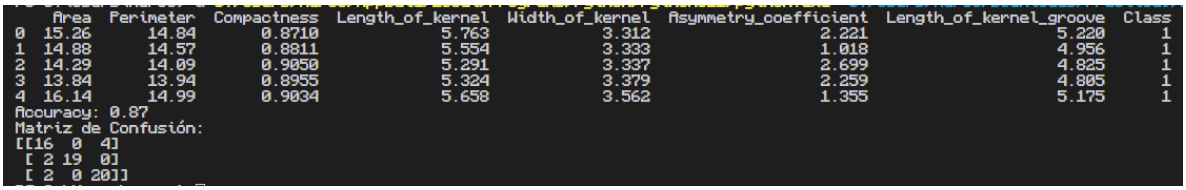
- ***Método de clasificación:*** 1NN
- ***Método de validación:*** Hold – Out (70-30)
- ***Dataset:*** Wheat Seeds

Se realizan ajustes en el código fuente:

```
# Carga el archivo dataset Wheat Seed
file_path = 'C:/Users/marco/Downloads/seeds_dataset.txt'

column_names = [
    "Area", "Perimeter", "Compactness", "Length_of_kernel", "Width_of_kernel",
    "Asymmetry_coefficient", "Length_of_kernel_groove", "Class"
]

data = pd.read_csv(file_path, header=None, names=column_names, delimiter='\t')

data = data.apply(pd.to_numeric, errors='coerce')

data.dropna(inplace=True)

print(data.head())
```

Similar al caso anterior, las ajusten se realizan en la ubicación del archivo del data set, así como el array que almacena el nombre de las columnas. En este caso, ya que el archivo es un *.txt* y no un *.csv* o *.data*, el método de lectura debe definirse con un delimitador que permita diferencias entre valores, en este caso, cada punto del data set se encuentra delimitado por una tabulación

Resultado:



	Area	Perimeter	Compactness	Length_of_kernel	Width_of_kernel	Asymmetry_coefficient	Length_of_kernel_groove	Class
0	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	1
1	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	1
2	14.29	14.09	0.9050	5.291	3.337	2.699	4.825	1
3	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	1
4	16.14	14.99	0.9034	5.658	3.562	1.355	5.175	1

Accuracy: 0.87
Matriz de Confusión:
[[16 0 4]
[2 19 0]
[2 0 20]]

Para este data set, se obtuvo una precisión del 87%, logrando una precisión aceptable. Ya que el data set cuenta con tres etiquetas de diferenciación, la matriz generada es de 3x3 que se puede interpretar de la siguiente forma: Se obtuvieron 16 verdaderos positivos de la clase 1, aunque 4 elementos fueron clasificados erróneamente como clase 3. Se obtienen 19 verdaderos positivos de la clase 2, aunque 2 elementos fueron clasificados erróneamente como clase 1. Finalmente, 20 elementos fueron clasificados como verdaderos positivos de la clase 3, con 2 elementos clasificados erróneamente como clase 1.

10 Fold – Cross Validation

- ***Método de clasificación:*** 1NN
- ***Método de validación:*** 10 Fold – Cross Validation
- ***Dataset:*** Iris plant

El código es similar al utilizado en el método Hold – Out, aunque se realizaron modificaciones en ciertas partes del código fuente.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold       # Libreria para implementación de K Fold Cross
from sklearn.metrics import accuracy_score, confusion_matrix
from scipy.spatial import distance
```

Una de las bibliotecas fue cambiada. Se define a la librería sklearn para invocar la clase KFold, que permite la implementación del método K Fold Cross. Las demás librerías no son modificadas.

```
file_path = 'C:/Users/marco/Downloads/iris/iris.data'

column_names = ["sepal_length", "sepal_width", "petal_length", "petal_width", "class"]
data = pd.read_csv(file_path, header=None, names=column_names)

data.dropna(inplace=True)

X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

def knn_classifier(X_train, y_train, X_test):
    predictions = []
    for test_point in X_test:
        distances = [distance.euclidean(test_point, train_point) for train_point in X_train]
        nearest_index = np.argmin(distances)
        predictions.append(y_train[nearest_index])
    return np.array(predictions)
```

El resto del código fuente no sufre modificaciones, pues aun se usa el método *knn_classifier*.

```
# Implementación de K-Fold Cross Validation con K=10
kfold = KFold(n_splits=10, shuffle=True, random_state=42)  # Dividir en 10 pa

# Variables para almacenar los resultados de cada fold
accuracies = []     #Lista para almacenar los acurracies de cada fold
conf_matrices = []  #Lista par almacenar la matriz de confusión de cada fold
```

Se invoca al metodo KFold, que permite dividir el data set en n cantidad de folds, en este caso 10. Además, recibe como condición *suffle = true* y *random_state = 42*. De esta forma, cada fold es mezclado aleatoriamente y se mantiene la reproducibilidad.

Además, se definen dos listas que servirán para guardar el accuracy y la matriz de confusión para cada fold.

```python
for train_index, test_index in kfold.split(X):  #Proceso de entrenamiento y prueba, devuelve indices de entranm
    # Dividir el dataset en conjunto de entrenamiento y prueba según los índices de K Fold
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Predecir las etiquetas en el conjunto de prueba usando 1-NN
    y_pred = knn_classifier(X_train, y_train, X_test)

    # Calcular el Accuracy y la matriz de confusión para el fold iterado y guardar en la lista correspondiente
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)

    conf_matrix = confusion_matrix(y_test, y_pred)
    conf_matrices.append(conf_matrix)
```

Se realiza el proceso de entrenamiento para cada fold, y cada iteración devuelve el índice de los datos usados para el set de entrenamiento y el set de validación. Estos serán guardados en los subconjuntos *X_train, X_test, y_train* y *y_test.*

Luego, se predicen las etiquetas para cada iteración del fold haciendo uso del metodo *knn_classifier*. Posteriormente, se realiza el calculo del accuracy y matriz de transición de la iteración actual. Cada iteración es guardada en su lista correspondiente.

```python
# Cálculo del accuracy promedio
mean_accuracy = np.mean(accuracies)
print(f"Accuracy promedio en 10 folds: {mean_accuracy:.2f}")

# Matriz de confusión general
total_conf_matrix = np.sum(conf_matrices, axis=0)
print("Matriz de Confusión total:")
print(total_conf_matrix)
```

Finalmente, se realiza el cálculo del accuracy promedio y la matriz de confusión general, tomando en cuenta las listas creadas anteriormente.

Resultado:

```
Accuracy promedio en 10 folds: 0.96
Matriz de Confusión total:
[[50  0  0]
 [ 0 47  3]
 [ 0  3 47]]
```

Se obtiene una precisión promedio del 96% teniendo en cuenta los 10 folds creados, lo que determina un nivel alto de precisión.

Por otro lado, la matriz de confusión general muestra que 50 puntos del data set fueron clasificados correctamente a la clase 1. Sin embargo, 47 puntos del data set fueron clasificados correctamente a la clase 2, con 3 puntos clasificados incorrectamente a la clase

3. Finalmente, 47 puntos fueron clasificados correctamente a la clase 3, con 3 puntos clasificados incorrectamente a la clase 2.

- ***Método de clasificación:*** 1NN
- ***Método de validación:*** 10 Fold – Cross Validation
- ***Dataset:*** Pima Indians Diabetes

Como ya se ha visto anteriormente, el código fuente no sufre modificaciones en la implementación del método *knn_classifier* ni en la implementación del método *10 Fold Cross*. El único cambio proviene en la especificación de la ruta del archivo del data set y la definición de las columnas.

```
# Cargar el dataset de Pima Indians Diabetes
file_path = 'C:/Users/marco/Downloads/pima-indians-diabetes.csv'
column_names = [
    "pregnancies", "glucose", "blood_pressure", "skin_thickness", "insulin",
    "bmi", "diabetes_pedigree_function", "age", "outcome"
]
data = pd.read_csv(file_path, header=None, names=column_names)

data = data.apply(pd.to_numeric, errors='coerce')

data.dropna(inplace=True)

print(data.head())
```

Resultado:

```
Accuracy promedio en 10 folds: 0.68
Matriz de Confusión total:
[[376 124]
 [122 146]]
```

Se obtuvo un accuracy promedio entre los diez fold del 68%, menor al visto en la aplicación del método Hold – Out. Se observa un rendimiento moderado.

La matriz de confusión nos permite observar que se clasificaron correctamente 376 elementos como verdaderos negativos (sin diabetes), mientras que 146 fueron clasificados como verdaderos positivos (con diabetes). Además, se obtuvieron 124 falsos positivos y 122 falsos negativos.

- ***Método de clasificación:*** 1NN
- ***Método de validación:*** 10 Fold – Cross Validation
- ***Dataset:*** Wheat Seeds

```
# Dataset de Wheat Seed
file_path = 'C:/Users/marco/Downloads/seeds_dataset.txt'

column_names = [
    "Area", "Perimeter", "Compactness", "Length_of_kernel", "Width_of_kernel",
    "Asymmetry_coefficient", "Length_of_kernel_groove", "Class"
]

data = pd.read_csv(file_path, header=None, names=column_names, delimiter='\t')

data = data.apply(pd.to_numeric, errors='coerce')
data.dropna(inplace=True)

print(data.head())
```

El único cambio realizado al código fuente es referente a la especificación de la ruta del data set así como el nombre de las columnas que lo conforman.

Resultados:

```
Accuracy promedio en 10 folds: 0.91
Matriz de Confusión total:
[[58  4  8]
 [ 4 66  0]
 [ 3  0 67]]
```

Se obtuvo un accuracy general del 91%, lo que denota un modelo con alta precisión. Por otro lado, la matriz general obtenida nos permite observar que 58 puntos fueron clasificados correctamente a la clase 1, mientras que 4 fueron clasificados incorrectamente a la clase 2 y 8 a la clase 3. Este análisis puede realizarse en las columnas siguientes, donde se observa que 66 elementos fueron clasificados correctamente a la clase 2, mientras que 4 fueron clasificados incorrectamente a la clase 1. Finalmente, 67 puntos fueron clasificados correctamente a la clase 3, mientras que 3 fueron clasificados incorrectamente a la clase 1.

Leave One Out

- ***Método de clasificación:*** 1NN
- ***Método de validación:*** Leave One Out
- ***Dataset:*** Iris plant

Similar a los casos anteriores, no se realizan modificaciones importantes al código fuente, solo la implementación para el método de validación Leave One Out.

```
1   import numpy as np
2   import pandas as pd
3   from sklearn.model_selection import LeaveOneOut      # Libreria
4   from sklearn.metrics import accuracy_score, confusion_matrix
5   from scipy.spatial import distance
```

Se implementa la librería *sklearn* con la clase *LeaveOneOut*, para hacer uso del método Leave One Out.

```
25  # Implementación de Leave-One-Out Cross Validation
26  loo = LeaveOneOut()
27
28  accuracies = []
29  all_y_true = []  # Lista para almacenar las etiquetas verdaderas
30  all_y_pred = []  # Lista para almacenar las etiquetas predichas
31
32  for train_index, test_index in loo.split(X):  # Leave one out una muestra de test por iteración
33      # Dividir el dataset en conjunto de entrenamiento y prueba según los indices en cada iteración
34      X_train, X_test = X[train_index], X[test_index]
35      y_train, y_test = y[train_index], y[test_index]
36
37      X_test = X_test.reshape(1, -1)  # Convierte X_test a un array 2D de forma (1, n_features)
38
39      y_pred = knn_classifier(X_train, y_train, X_test)
40
41      # Calcular el Accuracy y guardar las etiquetas verdaderas y predichas
42      accuracy = accuracy_score(y_test, y_pred)
43      accuracies.append(accuracy)
44
45      # Almacenar las etiquetas verdaderas y las predicciones para la matriz de confusión
46      all_y_true.extend(y_test)
47      all_y_pred.extend(y_pred)
```

Se crea una instancia de Leave One Out, demás de tres listas para el almacenamiento del accurracy y las etiquetas del data set.

Se itera en cada instancia del data set para obtener los indicen del set de entrenamiento y del test de validación.

Igual que en método anteriores, se guarda las etiquetas predichas haciendo uso del método *knn_classifier* y se calcula el acurracy promedio y la matriz de confusión general.

Resultados:

```
Accuracy promedio en Leave-One-Out: 0.96
Matriz de Confusión total:
[[50  0  0]
 [ 0 47  3]
 [ 0  3 47]]

[Done] exited with code=0 in 16.463 seconds
```

Se obtuvo un accuracy del 96%, similar a los resultados obtenidos en los métodos anteriores. Por otro lado, la matriz de confusión general nos permite observar que se clasificaron correctamente 50 elementos en la clase 1. Por otro lado, las clases 2 y 3 muestran un comportamiento similar, cada una clasificando correctamente 47 instancias del data set.

- ***Método de clasificación:*** 1NN
- ***Método de validación:*** Leave One Out
- ***Dataset:*** Pima Indians Diabetes

```
Accuracy promedio en Leave-One-Out: 0.68
Matriz de Confusión total:
[[378 122]
 [124 144]]

[Done] exited with code=0 in 23.11 seconds
```

Se obtuvo un accuracy promedio de 68%. La matriz de confusión general muestra que se clasificaron 378 verdaderos negativos y 144 verdaderos positivos. Por otro lado, se obtuvieron 122 falsos positivos y 124 falsos negativos.

- ***Método de clasificación:*** 1NN
- ***Método de validación:*** Leave One Out
- ***Dataset:*** Wheat Seeds

```
Accuracy promedio en Leave-One-Out: 0.90
Matriz de Confusión total:
[[58  4  8]
 [ 5 65  0]
 [ 3  0 67]]

[Done] exited with code=0 in 7.221 seconds
```

Se obtuvo un accuracy del 90% para este data set. Se clasificaron correctamente 58 puntos en la clase 1, mientras que se obtuvieron 4 clasificaciones incorrectas de la clase 2 y 8 de la clase 3. Del mismo modo, se obtuvieron 65 clasificaciones correctas de clase 2 y 5 clasificaciones incorrectas de la clase 1. Finalmente, se obtuvieron 67 clasificaciones correctas de la clase 3 y 3 clasificaciones incorrectas de la clase 1.

Conclusiones

Una vez concluidos todos los análisis obtenidos utilizando el método de clasificación *1NN* con los tres métodos de validación *(Hold – Out, 10 Cross Fold y Leave One Out)* se puede concluir lo siguiente: El data set de iris plant es que más índice de precisión obtuvo en cada uno de los métodos de validación, cercano al 98% en cada uno de los métodos.

Por otro lado, el data set de Pima Indian Diabetes obtuvo un accuracy promedio del 60% en cada uno de los métodos, presumiblemente al desequilibrio de instancias que existe entre clases, por lo que existe una clase mayoritaria que provoca una mala clasificación. Del mismo modo, puede deberse a la complejidad de datos que maneja el data set así como el uso de un clasificador básico.

Finalmente, el data set correspondiente al Wheat Seeds obtuvo un accuracy promedio del 88%. En este caso, la baja precisión se debe a una mala distinción de los modelos entre la clase numero 1 y la clase numero 3. Esto se debe que estas clases cuentan con características similares en su información, por lo que la aplicación de un método de clasificación KNN con un valor de K mayor puede aumentar la precisión del modelo.

Referencias

- Brownlee, J. (2023). *Standard machine learning datasets*. Machine Learning Mastery. Recuperado de https://machinelearningmastery.com/standard-machine-learning-datasets/

- Turing. (2023). *Different types of cross-validations in machine learning and their explanations*. Recuperado de https://www.turing.com/kb/different-types-of-cross-validations-in-machine-learning-and-their-explanations

- GeeksforGeeks. (2023). *Cross-validation in machine learning*. Recuperado de https://www.geeksforgeeks.org/cross-validation-machine-learning/

- IBM. (2023). *K-nearest neighbors (KNN)*. Recuperado de https://www.ibm.com/mx-es/topics/knn