Instituto Politécnico Nacional

Escuela Superior de Cómputo

Inteligencia Artificial

6CV2

Prof. ANDRES GARCIA FLORIANO

Ejercicio de laboratorio 10
Clasificador Euclidiano

Integrantes:

- Cadenas Acevedo Jesús Alejandro
- Gomez Jasso Rogelio Asahid
- Hernández Saucedo Brenda
- Ramirez Vazquez Yahaida Michelle
- Rodríguez Escogido Julio

Índice

Objetivo **3**
Introducción **3**
Modelo K-NN 3
Distancia euclidiana 4
Validación y Evaluación del Modelo 4
Hold-Out (70/30) 4
10-Fold Cross-Validation 5
Desarrollo **5**
Explicación del código 5
Salida del programa 9
Interpretación de las salidas 9
Conclusiones **10**
Github **10**
Anexo código **10**
Referencias bibliográficas **12**

Objetivo

Programa el modelo del Clasificador Euclidiano y aplícalo en los Bancos de Datos:

- Iris Plant
- El BD que elegiste en el ejercicio anterior (Breast Cancer)

Los métodos de validación a emplear son:

- Hold-Out 70/30
- 10-Fold Cross-Validation

Se deberá entregar el reporte del accuracy del clasificador con ambos BD y ambos métodos de validación.

Introducción

En el campo del aprendizaje automático, los clasificadores basados en el vecino más cercano son una familia de algoritmos simples pero poderosos que se utilizan ampliamente en una variedad de aplicaciones de clasificación. Estos algoritmos se basan en la idea intuitiva de que puntos cercanos en el espacio de características tienden a tener etiquetas similares.

Modelo K-NN

El algoritmo K-NN (K Nearest Neighbors) o en español, K Vecinos Más Cercanos, trata de buscar los K puntos más cercanos a un punto concreto para poder inferir su valor. Este algoritmo pertenece al conjunto de técnicas del aprendizaje automático supervisado, y puede ser utilizado tanto para problemas de clasificación, como de regresión.

Para los problemas de clasificación, se asigna una etiqueta de clase sobre la base de un voto mayoritario, es decir, se utiliza la etiqueta que se representa con más frecuencia alrededor de un punto de datos determinado; "voto mayoritario" técnicamente requiere una mayoría superior al 50 %, lo que funciona principalmente cuando solo hay dos categorías. Cuando tiene varias clases, por ejemplo, cuatro categorías, no necesita necesariamente el 50 % de los votos para llegar a una conclusión sobre una clase; puede asignar una etiqueta de clase con un voto superior al 25 %.

El objetivo del algoritmo del vecino más cercano es identificar los vecinos más cercanos de un punto de consulta dado, de modo que podamos asignar una etiqueta de clase a ese punto. Para hacer esto, KNN debe definir una métrica de distancia, que ayudará a formar límites de decisión, que dividen los puntos de consulta en diferentes regiones. Si bien hay varias medidas de distancia entre las que puede

elegir, el ejercicio desarrollado en este documento solo utilizará la distancia Euclidiana.
Antes de pasar a explicar la métrica empleada, cabe aclarar que uno de los miembros más simples de esta familia es el clasificador del vecino más cercano (1-NN), donde la etiqueta de un punto de prueba se determina por la etiqueta de su vecino más cercano en el espacio de características y es el que se implementó en este ejercicio.

Distancia euclidiana

La distancia euclídea o distancia euclidiana es una de las distancias más básicas, pero a la vez, de las más utilizadas en el mundo del Machine Learning. Esta distancia está pensada sólo para variables numéricas o bien para vectores de valor real. Si quisiéramos utilizar una variable categórica, tendríamos que utilizar alguna técnica de encoding, u otra distancia que sí permite este tipo de variables. Usando la fórmula a continuación, mide una línea recta entre el punto de consulta y el otro punto que se mide.

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

Validación y Evaluación del Modelo

Para evaluar el desempeño del clasificador euclidiano, se utilizaron dos técnicas de validación diferentes: Hold-Out (70/30) y 10-Fold Cross-Validation. Estas técnicas ya las habíamos trabajado en el ejercicio anterior, permiten estimar la capacidad de generalización del modelo a datos no vistos y proporcionan una medida de su precisión en diferentes escenarios de validación.

Hold-Out (70/30)

El método hold-out es el más sencillo de los distintos métodos de validación cruzada. Este separa el conjunto de datos disponibles en dos subconjuntos, uno utilizado para entrenar el modelo y otro para realizar el test de validación. La proporción de esta división puede variar, como por ejemplo, 70-30, 60-40, 75-25, 80-20, o incluso 50-50, dependiendo de las características del caso de uso, en este caso se utilizó 70-30. De esta manera, se crea un modelo únicamente con los datos de entrenamiento. Con el modelo creado se generan datos de salida que se comparan con el conjunto de datos reservados para realizar la validación (que no

han sido utilizados en el entrenamiento, por lo que no han sido utilizados para generar el modelo. Los estadísticos obtenidos con los datos del subconjunto de validación son los que nos dan la validez del método empleado en términos de error.

10-Fold Cross-Validation

El otro método utilizado, k-fold, está basado en el método anterior, pero con mayor utilidad cuando el conjunto de datos es pequeño. En este caso, el total de los datos se dividen en k subconjuntos de tamaños casi iguales, en cada iteración, aplicamos el método hold-out, uno de estos conjuntos se selecciona como conjunto de prueba, mientras que los k-1 conjuntos restantes se utilizan para entrenar el modelo. Se repite hasta que cada conjunto ha sido utilizado como conjunto de prueba al menos una vez.
La media de los errores de todas las iteraciones se calcula como la estimación del error de prueba CV. Aquí, el número de pliegues k es menor que el número total de observaciones en los datos (k < n), y estamos promediando los resultados de k modelos ajustados. K-Fold CV se realiza comúnmente con k=5 o k=10, valores que han demostrado empíricamente producir estimaciones de error de prueba con bajo sesgo y baja varianza; en este caso se utilizó k=10.

Desarrollo

Explicación del código

A continuación se explicará parte por parte el código implementado.

Primero, importamos las librerías necesarias, estamos utilizando sklearn.datasets para ocupar los datasets necesarios. La sklearn.model_selection para poder utilizar las funciones para dividir los datos en conjuntos de entrenamiento y prueba y realizar validación cruzada. También la sklearn.metrics para utilizar las funciones para evaluar la calidad de los modelos, como la precisión (accuracy_score). Y scipy.spatial.distance para utilizar las funciones para calcular distancias, como la distancia euclidiana.

D: > ESCOM > IA > Lab10 > Claisificador_Euclidiano.py > ...

```
1  import numpy as np
2  import pandas as pd
3  from sklearn.datasets import load_iris, load_breast_cancer
4  from sklearn.model_selection import train_test_split, KFold
5  from sklearn.metrics import accuracy_score
6  from scipy.spatial import distance
```

Ahora vamos a cargar los dos datasets a utilizar, las características se almacenan en matrices X_iris y X_bc, mientras que las etiquetas correspondientes se almacenan en los vectores y_iris y y_bc.

D: > ESCOM > IA > Lab10 > Claisificador_Euclidiano.py

```
8   # Cargar datos Iris
9   iris = load_iris()
10  X_iris = iris.data
11  y_iris = iris.target
12
13  # Cargar datos Breast Cancer
14  breast_cancer = load_breast_cancer()
15  X_bc = breast_cancer.data
16  y_bc = breast_cancer.target
```

A continuación pasamos a la implementación del clasificador euclidiano, el cual clasifica un punto basado en la clase del punto más cercano en el espacio de características, donde almacenamos los datos de entrenamiento y realizamos las predicciones basadas en la distancia euclidiana.

En la clase *EuclideanClassifier* estamos implementando un clasificador del vecino más cercano (1-NN) basado en la distancia euclidiana. En esta clase definimos dos métodos principales: fit y predict. El método *fit* almacena los datos de entrenamiento y las etiquetas, mientras que el método *predict* realiza predicciones para nuevos datos basándose en los vecinos más cercanos en el conjunto de entrenamiento.

D: > ESCOM > IA > Lab10 > Claisificador_Euclidiano.py > ...

```
18  # Definición del Clasificador Euclidiano
19  class EuclideanClassifier:
20      def fit(self, X_train, y_train):
21          self.X_train = X_train
22          self.y_train = y_train
23
24      def predict(self, X_test):
25          y_pred = []
26          for x in X_test:
27              distances = np.array([distance.euclidean(x, x_train) for x_train in self.X_train])
28              nearest_index = np.argmin(distances)
29              y_pred.append(self.y_train[nearest_index])
30          return np.array(y_pred)
```

A continuación, definimos la función para realizar la validación del modelo: hold_out_validation. En la función hold_out_validation dividimos los datos mediante *train_test_split* en conjuntos de entrenamiento y prueba utilizando la estrategia hold-out (70/30), ajustamos el modelo, el modelo predice las etiquetas del conjunto de prueba y finalmente calculamos la precisión de las predicciones en el conjunto de prueba.

D: > ESCOM > IA > Lab10 > Claisificador_Euclidiano.py > ...

```python
# Validación Hold-Out 70/30
def hold_out_validation(X, y, test_size=0.3):
    # División de los datos
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=42)
    # Creación y ajuste del modelo
    model = EuclideanClassifier()
    model.fit(X_train, y_train)
    # Predicción
    y_pred = model.predict(X_test)
    # Evaluación
    accuracy = accuracy_score(y_test, y_pred)
    return accuracy
```

Evaluamos los modelos, para cada dataset, llamamos a la función *hold_out_validation* e imprimimos la precisión obtenida. Esto nos mostrará en consola una medida de qué tan bien el modelo clasifica los datos no vistos.

D: > ESCOM > IA > Lab10 > Claisificador_Euclidiano.py > ...

```python
# Validación Hold-Out para Iris
accuracy_iris_hold_out = hold_out_validation(X_iris, y_iris)
print(f'Iris Hold-Out 70/30 Accuracy: {accuracy_iris_hold_out:.2f}')

# Validación Hold-Out para Breast Cancer
accuracy_bc_hold_out = hold_out_validation(X_bc, y_bc)
print(f'Breast Cancer Hold-Out 70/30 Accuracy: {accuracy_bc_hold_out:.2f}')
```

Ahora vamos a pasar a la segunda validación del modelo. La función *cross_validation* utiliza la validación cruzada k-fold (10-fold) para evaluar el modelo en múltiples divisiones de los datos, calculando la precisión promedio del modelo en todas las divisiones. Lo primero es configurar el K-Fold Cross-Validation, estableciendo que dividiremos el dataset en 10 partes (folds), luego iteramos sobre cada fold, se obtienen índices para los conjuntos de entrenamiento y prueba, ajustamos el modelo con los datos de entrenamiento y predecimos las etiquetas de los datos de prueba. La precisión se calcula y se almacena para cada fold. Finalmente calculamos la media de las precisiones obtenidas en todos los folds.

D: > ESCOM > IA > Lab10 > Claisificador_Euclidiano.py > ...

```python
# Validación 10-Fold Cross-Validation
def cross_validation(X, y, n_splits=10):
    # Configuración del K-Fold Cross-Validation
    kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
    accuracies = []
    # Iteración sobre cada fold
    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        # Creación y ajuste del modelo en cada fold
        model = EuclideanClassifier()
        model.fit(X_train, y_train)
        # Predicción y evaluación en cada fold
        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        accuracies.append(accuracy)
    # Cálculo de la precisión promedio
    return np.mean(accuracies)
```

Evaluamos los modelos, para cada dataset, llamamos a la función *cross_validation* e imprimimos la precisión obtenida. Esto nos mostrará en consola una medida de la capacidad del modelo para generalizar a nuevos datos.

D: > ESCOM > IA > Lab10 > Claisificador_Euclidiano.py > ...

```python
# Cross-Validation para Iris
accuracy_iris_cv = cross_validation(X_iris, y_iris)
print(f'Iris 10-Fold Cross-Validation Accuracy: {accuracy_iris_cv:.2f}')

# Cross-Validation para Breast Cancer
accuracy_bc_cv = cross_validation(X_bc, y_bc)
print(f'Breast Cancer 10-Fold Cross-Validation Accuracy: {accuracy_bc_cv:.2f}')
```

Salida del programa

```
PS C:\Users\brend> & C:/Users/brend/AppData/Local/Programs/Python/Python312/python.exe d:/ESCOM/IA/Lab10/Claisificador_Euclidiano.py
Iris Hold-Out 70/30 Accuracy: 1.00
Breast Cancer Hold-Out 70/30 Accuracy: 0.94
Iris 10-Fold Cross-Validation Accuracy: 0.96
Breast Cancer 10-Fold Cross-Validation Accuracy: 0.92
```

Interpretación de las salidas

- **Iris Hold-Out 70/30 Accuracy: 1.00** → Esta salida nos dice que, al utilizar la validación Hold-Out con una división de datos 70/30 en el conjunto de datos Iris, el modelo de clasificación alcanzó una precisión del 100%. Esto significa que el modelo clasificó correctamente todas las muestras en el conjunto de datos de prueba después de ser entrenado con el conjunto de datos de entrenamiento.
- **Breast Cancer Hold-Out 70/30 Accuracy: 0.94** → En el caso del conjunto de datos de cáncer de mama, utilizando la validación Hold-Out 70/30, el modelo alcanzó una precisión del 94%. Esto indica que el modelo clasificó correctamente el 94% de las muestras en el conjunto de datos de prueba después de ser entrenado con el conjunto de datos de entrenamiento.
- **Iris 10-Fold Cross-Validation Accuracy: 0.96** → Nos dice que utilizando la validación cruzada 10-Fold Cross-Validation en el conjunto de datos Iris, el modelo de clasificación alcanzó una precisión promedio del 96% en todas las divisiones de los datos. Esto significa que, en promedio, el modelo clasificó correctamente el 96% de las muestras en los conjuntos de datos de prueba generados durante la validación cruzada.
- **Breast Cancer 10-Fold Cross-Validation Accuracy: 0.92** → En el caso del conjunto de datos de cáncer de mama, nos dice que utilizando la validación cruzada 10-Fold Cross-Validation, el modelo alcanzó una precisión promedio del 92% en todas las divisiones de los datos. Esto indica que, en promedio, el modelo clasificó correctamente el 92% de las muestras en los conjuntos de datos de prueba generados durante la validación cruzada.

Como podemos ver, los resultados arrojados fueron buenos, por lo que podemos concluir que el modelo es capaz de realizar buenas clasificaciones en datos no vistos.

Conclusiones

Al finalizar el desarrollo de este ejercicio, logramos realizar la implementación y evaluación de un clasificador euclidiano del vecino más cercano (1-NN), utilizando Python con ayuda de las bibliotecas scikit-learn y NumPy. En el clasificador que implementamos usamos la distancia euclidiana para determinar la similitud entre puntos en el espacio de características y empleamos dos estrategias de validación diferentes: Hold-Out (70/30) y 10-Fold Cross-Validation, para evaluar su rendimiento en los conjuntos de datos: Iris y Breast Cancer.

Pudimos notar mediante los resultados de la validación que el clasificador euclidiano pudo alcanzar altas precisiones en ambos conjuntos de datos utilizando la estrategia Hold-Out, con una precisión del 100% en Iris y del 94% en Breast Cancer. Y mediante la validación cruzada 10-Fold, que el modelo mantuvo un buen rendimiento promedio, con precisiones del 96% en Iris y del 92% en Breast Cancer.

Es así que al finalizar este ejercicio logramos entender por completo el funcionamiento e implementación del clasificador Euclidiano y dos técnica muy importantes para evaluar el rendimiento del mismo, ahora somos capaces de aplicar este tipo de clasificador en una variedad de escenarios de clasificación de datos y evaluar si es que está funcionando adecuadamente.

Github

https://github.com/Brend-hs/InteligenciaArtificial/blob/main/EjercicioLab_10/ClasificadorEuclidiano.py

Anexo código

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris, load_breast_cancer
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import accuracy_score
from scipy.spatial import distance

# Cargar datos Iris
iris = load_iris()
X_iris = iris.data
y_iris = iris.target

# Cargar datos Breast Cancer
```


```
breast_cancer = load_breast_cancer()
X_bc = breast_cancer.data
y_bc = breast_cancer.target

# Definición del Clasificador Euclidiano
class EuclideanClassifier:
    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X_test):
        y_pred = []
        for x in X_test:
                distances = np.array([distance.euclidean(x, x_train) for
x_train in self.X_train])
            nearest_index = np.argmin(distances)
            y_pred.append(self.y_train[nearest_index])
        return np.array(y_pred)

# Validación Hold-Out 70/30
def hold_out_validation(X, y, test_size=0.3):
    # División de los datos
        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size, random_state=42)
    # Creación y ajuste del modelo
    model = EuclideanClassifier()
    model.fit(X_train, y_train)
    # Predicción
    y_pred = model.predict(X_test)
    # Evaluación
    accuracy = accuracy_score(y_test, y_pred)
    return accuracy

# Validación Hold-Out para Iris
accuracy_iris_hold_out = hold_out_validation(X_iris, y_iris)
print(f'Iris Hold-Out 70/30 Accuracy: {accuracy_iris_hold_out:.2f}')

# Validación Hold-Out para Breast Cancer
```

```
accuracy_bc_hold_out = hold_out_validation(X_bc, y_bc)
print(f'Breast Cancer Hold-Out 70/30 Accuracy:
{accuracy_bc_hold_out:.2f}')

# Validación 10-Fold Cross-Validation
def cross_validation(X, y, n_splits=10):
    # Configuración del K-Fold Cross-Validation
    kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
    accuracies = []
    # Iteración sobre cada fold
    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        # Creación y ajuste del modelo en cada fold
        model = EuclideanClassifier()
        model.fit(X_train, y_train)
        # Predicción y evaluación en cada fold
        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        accuracies.append(accuracy)
    # Cálculo de la precisión promedio
    return np.mean(accuracies)

# Cross-Validation para Iris
accuracy_iris_cv = cross_validation(X_iris, y_iris)
print(f'Iris 10-Fold Cross-Validation Accuracy:
{accuracy_iris_cv:.2f}')

# Cross-Validation para Breast Cancer
accuracy_bc_cv = cross_validation(X_bc, y_bc)
print(f'Breast Cancer 10-Fold Cross-Validation Accuracy:
{accuracy_bc_cv:.2f}')
```

Referencias bibliográficas

- *¿Qué es KNN?* | IBM. (s. f.). https://www.ibm.com/mx-es/topics/knn
- Na, & Na. (2020, 15 julio). *Algoritmo K-Nearest Neighbor | Aprende Machine Learning*. Aprende Machine Learning.

https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/
- Pérez-Planells, Ll., Delegido, J., Rivera-Caicedo, J.P., Verrelst, J. (2015, 1 diciembre). *Análisis de métodos de validación cruzada para la obtención robusta de parámetros biofísicos*. REVISTA DE TELEDETECCIÓN. https://core.ac.uk/download/pdf/71051261.pdf
- *1.7.1 Cross validation.* (2024, 24 enero). DataQuarks. https://data-quarks.com/2024/01/23/1-7-1-cross-validation/