INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Integrantes:

Hernández Sumaya Lucas Melvin Peralta Moreno Rafael
Pérez Reyes Sergio Adrián Ruiz Reyes Luis Ángel Velasco
Cruz Laura Itzel

Grupo:

6CV3

Materia:

Inteligencia Artificial

Profesor:

García Floriano Andrés

Laboratorio 10

Introducción

El clasificador Naive Bayes es un método de aprendizaje supervisado basado en la aplicación del teorema de Bayes, que se utiliza ampliamente para problemas de clasificación. Este modelo asume que las características del conjunto de datos son condicionalmente independientes entre sí, lo que significa que la presencia (o ausencia) de una característica no influye en la presencia (o ausencia) de otra, dado el valor de la clase objetivo. Aunque esta suposición rara vez se cumple en la práctica, el clasificador Naive Bayes ha demostrado ser altamente efectivo en una variedad de aplicaciones.El clasificador 1-Nearest Neighbor, conocido como 1NN, es una versión específica del algoritmo de vecinos cercanos, en la que el valor de \(K \) se establece en 1. Este clasificador también se basa en la proximidad entre puntos dentro del espacio de características, pero permite un mayor grado de flexibilidad al soportar distintas métricas de distancia, no solo la euclidiana. En el proceso de clasificación, el algoritmo busca el punto en el conjunto de entrenamiento que esté más cerca del nuevo punto de prueba y asigna la clase de este punto vecino al nuevo dato. Este modelo resulta efectivo en problemas donde los puntos de una misma clase están próximos entre sí y el comportamiento del clasificador depende de los datos más cercanos.

El método de validación Hold Out es una técnica para evaluar el desempeño de los modelos de clasificación. Este enfoque consiste en dividir el conjunto de datos en dos subconjuntos: uno para entrenamiento y otro para prueba. En el caso de Hold Out 70/30, el 70% de los datos se utiliza para entrenar el modelo, mientras que el 30% restante se emplea para probar su rendimiento. Este método permite estimar el comportamiento del modelo en datos que no ha visto antes, proporcionando una medida de su capacidad de generalización. El 10-Fold Cross-Validation es una técnica de validación más robusta y consiste en dividir el conjunto de datos en 10 partes iguales. En cada iteración, una parte se utiliza para evaluar el modelo y las otras nueve para entrenarlo, hasta que cada parte haya sido usada una vez para validación. Al final, se calcula el promedio de las 10 mediciones de rendimiento, obteniendo una estimación confiable del desempeño general del modelo. Otro método de validación es el Leave-One-Out, que es una variación del Cross-Validation, en el cual el modelo se entrena con todos los puntos de datos excepto uno, que se usa para la evaluación. Este proceso se repite para cada punto de datos en el conjunto, proporcionando una evaluación más precisa aunque costosa en términos de tiempo de cómputo.

Para evaluar el desempeño de los clasificadores, se utiliza el cálculo de la Accuracy y la matriz de confusión. La Accuracy o precisión mide el porcentaje de predicciones correctas realizadas por el modelo con relación al total de predicciones. Se calcula dividiendo el número de aciertos entre el total de casos y proporciona una medida sencilla de la efectividad general del clasificador. Sin embargo, cuando las clases están desbalanceadas, la matriz de confusión es una herramienta complementaria muy útil. Esta matriz permite desglosar las predicciones correctas e incorrectas, mostrando la cantidad de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos que el modelo ha generado. La matriz de confusión permite identificar cómo se comporta el clasificador con cada clase, ayudando a comprender si existe algún tipo de sesgo hacia una clase específica o si hay problemas de clasificación entre ciertas clases.

Desarrollo

El código implementa y evalúa dos clasificadores supervisados, uno el naive bayes y otro usando el método de vecinos más cercanos (1NN), para clasificar datos de distintos conjuntos (Iris, Vino y Dígitos). Cada clasificador utiliza tres métodos de validación: Hold Out, que divide los datos en entrenamiento y prueba; validación cruzada de 10 partes (10-Fold Cross-Validation), que evalúa el modelo con múltiples particiones de los datos; y Leave-One-Out, donde cada punto se evalúa individualmente mientras los demás entrenan el modelo. Para cada combinación de clasificador y método de validación, el código calcula la exactitud y la matriz de confusión, que muestran el rendimiento y la distribución de aciertos y errores en las predicciones del modelo.

A continuación, procedemos a explicar el código más a detalle:

Importación de Librerías

```
1  import numpy as np
2  from sklearn.model_selection import train_test_split, KFold, LeaveOneOut
3  from sklearn.neighbors import KNeighborsClassifier
4  from sklearn.metrics import accuracy_score, confusion_matrix
5  from sklearn.datasets import load_iris, load_wine, load_digits
6
```

Ilustración 1

Este bloque importa las librerías necesarias. “numpy” se utiliza para operaciones matemáticas y de matrices. “sklearn.model_selection” contiene los métodos de validación “train_test_split”, “KFold” y “LeaveOneOut”, mientras que “KNeighborsClassifier” permite implementar el clasificador 1NN. “accuracy_score” y “confusion_matrix” son métricas para evaluar el desempeño de los modelos, y “load_iris”, “load_wine”, “load_digits” proporcionan los conjuntos de datos.

Función para el Clasificador Naive bayes

```
# Función para implementar el clasificador k-NN con k configurable
def clasificador_knn(X_entrenamiento, y_entrenamiento, X_prueba, k=3):
    clasificador = KNeighborsClassifier(n_neighbors=k)
    clasificador.fit(X_entrenamiento, y_entrenamiento)
    return clasificador.predict(X_prueba)
```

Ilustración 2

Esta función implementa el clasificador Euclidiano. Para cada punto en “X_prueba”, calcula las distancias a todos los puntos en “X_entrenamiento”, encuentra el índice de la distancia más corta (el vecino más cercano) y asigna la clase de este vecino a “y_predicho”. El resultado es un arreglo con las clases predichas para cada punto de prueba.

Función para el Clasificador 1NN

```python
# Función para implementar el clasificador 1NN
def clasificador_1nn(X_entrenamiento, y_entrenamiento, X_prueba):
    clasificador = KNeighborsClassifier(n_neighbors=1)
    clasificador.fit(X_entrenamiento, y_entrenamiento)
    return clasificador.predict(X_prueba)
```

Ilustración 3

Esta función implementa el clasificador 1-Nearest Neighbor (1NN) usando "KNeighborsClassifier" de "scikit-learn". Configura "n_neighbors=1" para seleccionar el vecino más cercano, entrena el modelo con "X_entrenamiento" y "y_entrenamiento" y luego predice las clases de "X_prueba".

Función para el clasificador K-NN con k variable

```python
# Función para implementar el clasificador k-NN con k configurable
def clasificador_knn(X_entrenamiento, y_entrenamiento, X_prueba, k=3):
    clasificador = KNeighborsClassifier(n_neighbors=k)
    clasificador.fit(X_entrenamiento, y_entrenamiento)
    return clasificador.predict(X_prueba)
```

Ilustración 4

Función para Evaluar el Modelo

Esta función evalúa el modelo usando tres métodos de validación distintos ("Hold Out", "10-Fold Cross-Validation" y "Leave-One-Out"). Dependiendo de "metodo_validacion", divide los datos y utiliza el clasificador para predecir las clases de los datos de prueba, calculando la exactitud (porcentaje de aciertos) y la matriz de confusión. La matriz de confusión acumula los errores y aciertos por clase, y se actualiza en cada iteración de la validación.

```python
# Función para evaluar el desempeño del modelo
def evaluar_modelo(X, y, clasificador, metodo_validacion):
    n_clases = len(np.unique(y))
    exactitud = 0
    matriz_confusion = np.zeros((n_clases, n_clases), dtype=int)

    if metodo_validacion == 'hold_out':
        X_entrenamiento, X_prueba, y_entrenamiento, y_prueba = train_test_split(X, y, test_size=0.3, random_state=42)
        y_predicho = clasificador(X_entrenamiento, y_entrenamiento, X_prueba)
        exactitud = accuracy_score(y_prueba, y_predicho)
        matriz_confusion = confusion_matrix(y_prueba, y_predicho, labels=np.arange(n_clases))
```

Ilustración 5

El método Hold Out es el más simple de los tres. Aquí, el conjunto de datos se divide en dos partes: un subconjunto de entrenamiento y otro de prueba. En la configuración de Hold Out 70/30, el 70% de los datos se utiliza para entrenar el modelo, mientras que el 30% restante sirve para evaluar su rendimiento. Este método es rápido y fácil de implementar, aunque su precisión puede variar en

función de cómo se haga la partición, ya que el rendimiento depende de los datos que se incluyan en cada subconjunto.

```
elif metodo_validacion == '10_fold':
    validacion_kf = KFold(n_splits=10, shuffle=True, random_state=42)
    exactitudes = []
    for indice_entrenamiento, indice_prueba in validacion_kf.split(X):
        X_entrenamiento, X_prueba = X[indice_entrenamiento], X[indice_prueba]
        y_entrenamiento, y_prueba = y[indice_entrenamiento], y[indice_prueba]
        y_predicho = clasificador(X_entrenamiento, y_entrenamiento, X_prueba)
        exactitudes.append(accuracy_score(y_prueba, y_predicho))
        matriz_confusion += confusion_matrix(y_prueba, y_predicho, labels=np.arange(n_clases))
    exactitud = np.mean(exactitudes)
```

Ilustración 6

El 10-Fold Cross-Validation, o validación cruzada de 10 particiones, es un método que proporciona una evaluación más completa y estable del modelo. En este caso, el conjunto de datos se divide en 10 partes o "folds". En cada iteración, el modelo se entrena con nueve de estas partes y se evalúa con la décima, que actúa como conjunto de prueba. Esto se repite diez veces, alternando la parte que se utiliza para la prueba hasta que cada fold haya sido usado una vez para evaluar el modelo. Al final, se calcula el promedio de los resultados de las diez iteraciones, obteniendo una medida más confiable y menos dependiente de una única partición de los datos.

```
elif metodo_validacion == 'leave_one_out':
    validacion_loo = LeaveOneOut()
    exactitudes = []
    for indice_entrenamiento, indice_prueba in validacion_loo.split(X):
        X_entrenamiento, X_prueba = X[indice_entrenamiento], X[indice_prueba]
        y_entrenamiento, y_prueba = y[indice_entrenamiento], y[indice_prueba]
        y_predicho = clasificador(X_entrenamiento, y_entrenamiento, X_prueba)
        exactitudes.append(accuracy_score(y_prueba, y_predicho))
        matriz_confusion += confusion_matrix(y_prueba, y_predicho, labels=np.arange(n_clases))
    exactitud = np.mean(exactitudes)

return exactitud, matriz_confusion
```

Imagen 7

El método Leave-One-Out es una forma extrema de validación cruzada. Aquí, se toma un solo punto de datos como conjunto de prueba y se usa el resto para entrenar el modelo. Este proceso se repite para cada punto de datos, de modo que cada dato actúa una vez como conjunto de prueba mientras que todos los demás lo entrenan. Aunque este método proporciona una evaluación precisa, ya que explora todas las posibles divisiones del conjunto de datos, es computacionalmente costoso, especialmente con grandes volúmenes de datos, debido al alto número de iteraciones necesarias.

6. Lista de Conjuntos de Datos para Evaluar

```
# Lista de conjuntos de datos para evaluar
conjuntos_datos = {
    "Iris": load_iris(),
    "Wine": load_wine(),
    "Digits": load_digits()
}
```

Imagen 8

Aquí se cargan tres conjuntos de datos prediseñados (“Iris”, “Wine”, y “Digits”) de “sklearn”. Cada uno contiene características y etiquetas de diferentes tipos de datos, proporcionando variedad para evaluar el rendimiento de los clasificadores.

Evaluación de los Clasificadores

```
# Evaluar clasificadores en cada conjunto de datos y método de validación
for nombre, conjunto in conjuntos_datos.items():
    X = conjunto.data
    y = conjunto.target
    print(f"\nEvaluación para el conjunto de datos: {nombre}")

    # Clasificador Euclidiano
    print("\nClasificador Euclidiano")
    for metodo in ['hold_out', '10_fold', 'leave_one_out']:
        exactitud, matriz_confusion = evaluar_modelo(X, y, clasificador_euclidiano, metodo)
        print(f"{metodo} -> Exactitud: {exactitud:.2f}, Matriz de confusión:\n{matriz_confusion}")

    # Clasificador 1-NN
    print("\nClasificador 1-NN")
    for metodo in ['hold_out', '10_fold', 'leave_one_out']:
        exactitud, matriz_confusion = evaluar_modelo(X, y, clasificador_1nn, metodo)
        print(f"{metodo} -> Exactitud: {exactitud:.2f}, Matriz de confusión:\n{matriz_confusion}")
```

Imagen 9

Aquí el código evalúa el rendimiento de dos clasificadores (el clasificador Euclidiano y el clasificador 1NN) en tres conjuntos de datos (“Iris”, “Wine” y “Digits”). La evaluación de cada clasificador se realiza bajo diferentes métodos de validación (“Hold Out”, “10-Fold Cross-Validation” y “LeaveOne-Out”), proporcionando así una visión completa de su desempeño en distintos escenarios.

La evaluación comienza recorriendo cada conjunto de datos disponible en el diccionario “conjuntos_datos”. Para cada conjunto, se extraen las características (almacenadas en “X”) y las etiquetas o clases correspondientes (almacenadas en “y”). A continuación, el código muestra un mensaje indicando el inicio de la evaluación para el conjunto de datos actual.

Luego, el código procede a evaluar cada clasificador. Primero se analiza el clasificador Euclidiano: para cada método de validación (definido como “'hold_out'“, “'10_fold'“, y “'leave_one_out'“), la función “evaluar_modelo” calcula dos métricas clave: la exactitud (o “accuracy”) y la matriz de confusión. Estas métricas se obtienen haciendo uso de la lógica de cada método de validación. La exactitud refleja el porcentaje de predicciones correctas del modelo, mientras que la matriz de confusión permite observar los aciertos y errores detallados del clasificador en cada clase del conjunto de datos, mostrando la cantidad de verdaderos positivos, verdaderos negativos, falsos positivos y

falsos negativos. Tras calcular estas métricas, el código las imprime, permitiendo comparar cómo se comporta el clasificador Euclidiano con cada método de validación.

Después, se realiza el mismo procedimiento con el clasificador 1NN. Aquí, el código llama a "evaluar_modelo" para cada método de validación, calculando nuevamente la exactitud y la matriz de confusión, que se imprimen después de cada evaluación. Esta información permite elegir el clasificador y el método de validación que mejor se ajusten al problema específico en función de la precisión requerida y la naturaleza del conjunto de datos.

Pruebas

```
Evaluación para el conjunto de datos: Iris

Clasificador 1-NN
hold_out -> Exactitud: 1.00, Matriz de confusión:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
10_fold -> Exactitud: 0.96, Matriz de confusión:
[[50  0  0]
 [ 0 47  3]
 [ 0  3 47]]
leave_one_out -> Exactitud: 0.96, Matriz de confusión:
[[50  0  0]
 [ 0 47  3]
 [ 0  3 47]]

Clasificador Naive Bayes
hold_out -> Exactitud: 0.98, Matriz de confusión:
[[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]
10_fold -> Exactitud: 0.96, Matriz de confusión:
[[50  0  0]
 [ 0 47  3]
 [ 0  3 47]]
leave_one_out -> Exactitud: 0.95, Matriz de confusión:
[[50  0  0]
 [ 0 47  3]
 [ 0  4 46]]

Clasificador k-NN (k=3)
hold_out -> Exactitud: 1.00, Matriz de confusión:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
10_fold -> Exactitud: 0.97, Matriz de confusión:
[[50  0  0]
 [ 0 47  3]
 [ 0  2 48]]
leave_one_out -> Exactitud: 0.96, Matriz de confusión:
[[50  0  0]
 [ 0 47  3]
 [ 0  3 47]]

Clasificador k-NN (k=5)
hold_out -> Exactitud: 1.00, Matriz de confusión:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
10_fold -> Exactitud: 0.97, Matriz de confusión:
[[50  0  0]
 [ 0 47  3]
 [ 0  1 49]]
leave_one_out -> Exactitud: 0.97, Matriz de confusión:
[[50  0  0]
 [ 0 47  3]
 [ 0  2 48]]
```

Imagen 10

En el conjunto de datos *Iris*, se evaluaron cuatro clasificadores utilizando tres métodos de validación: *hold-out*, *10-fold cross-validation* y *leave-one-out cross-validation*. Los resultados muestran un desempeño notablemente alto en todos los modelos, con pequeñas diferencias entre ellos dependiendo del método de validación empleado.

El clasificador 1-NN alcanzó una exactitud perfecta en el método *hold-out*, clasificando correctamente todas las instancias en el conjunto de prueba. Este resultado demuestra la capacidad del modelo para ajustarse bien a los datos en este escenario. Sin embargo, en las validaciones *10-fold* y *leave-one-out*, la exactitud se redujo ligeramente al 96%. En estos casos, los errores se concentraron en la confusión entre las clases 2 y 3, donde se clasificaron incorrectamente 3 instancias de la clase 2 como clase 3 y viceversa. Este comportamiento puede explicarse por la naturaleza local del modelo 1-NN, que se ve afectado por las pequeñas variaciones en los datos de entrenamiento y prueba.

Por su parte, el clasificador Naive Bayes tuvo un rendimiento destacado, aunque ligeramente menor en algunos casos comparado con los métodos basados en vecinos. En el método *hold-out*, logró una exactitud del 98%, con solo un error en la clasificación de una instancia de la clase 2 como clase 3. En *10-fold*, obtuvo una exactitud similar al 1-NN con un 96%, mientras que en *leave-one-out* alcanzó un 95%. Este desempeño refleja que, aunque Naive Bayes se basa en la suposición de independencia

entre características, lo que no siempre es completamente válido en este conjunto de datos, sigue siendo un modelo robusto y eficiente para este problema.

El clasificador k-NN con k=3 mostró un desempeño sobresaliente. Al igual que 1-NN, alcanzó una exactitud perfecta del 100% en el método *hold-out*. En las validaciones *10-fold* y *leave-one-out*, logró una exactitud del 97% y 96%, respectivamente. Las matrices de confusión indican que las confusiones se concentraron en las clases 2 y 3, aunque con menos errores en comparación con 1-NN. Esto sugiere que utilizar tres vecinos en lugar de uno proporciona una mayor estabilidad frente a pequeñas variaciones en los datos.

El clasificador k-NN con k=5 también logró una exactitud del 100% en el método *hold-out*, lo que muestra su capacidad para manejar correctamente la separación entre clases en este conjunto de datos. En *10-fold*, alcanzó el 97%, igualando el desempeño del modelo con k=3. Sin embargo, en *leave-one-out*, logró una ligera mejora, con una exactitud del 97%. Las matrices de confusión reflejan una reducción en los errores de clasificación entre las clases 2 y 3 en comparación con los modelos anteriores, lo que indica que el uso de un mayor número de vecinos contribuye a reducir la sensibilidad del modelo frente a valores atípicos o casos particulares.

En general, los resultados reflejan que todos los clasificadores son altamente efectivos para el conjunto de datos *Iris*, alcanzando niveles de exactitud muy altos en los tres métodos de validación. Los clasificadores k-NN con k=3 y k=5 demostraron ser los más estables y precisos, especialmente en las validaciones cruzadas. Por otro lado, Naive Bayes, aunque presenta ligeras desventajas frente a los modelos basados en vecinos, sigue siendo una opción competitiva, eficiente y robusta. Esto confirma que el conjunto de datos *Iris* es ideal para clasificaciones supervisadas, dado su buena separación entre clases.

```
Evaluación para el conjunto de datos: Wine

Clasificador 1-NN
hold_out -> Exactitud: 0.80, Matriz de confusión:
[[17  0  2]
 [ 3 16  2]
 [ 1  3 10]]
10_fold -> Exactitud: 0.73, Matriz de confusión:
[[52  3  4]
 [ 5 52 14]
 [ 3 19 26]]
leave_one_out -> Exactitud: 0.77, Matriz de confusión:
[[52  3  4]
 [ 5 54 12]
 [ 3 14 31]]

Clasificador Naive Bayes
hold_out -> Exactitud: 1.00, Matriz de confusión:
[[19  0  0]
 [ 0 21  0]
 [ 0  0 14]]
10_fold -> Exactitud: 0.98, Matriz de confusión:
[[57  2  0]
 [ 0 69  2]
 [ 0  0 48]]
leave_one_out -> Exactitud: 0.98, Matriz de confusión:
[[57  2  0]
 [ 0 69  2]
 [ 0  0 48]]

Clasificador k-NN (k=3)
hold_out -> Exactitud: 0.74, Matriz de confusión:
[[17  0  2]
 [ 1 15  5]
 [ 1  5  8]]
10_fold -> Exactitud: 0.70, Matriz de confusión:
[[51  3  5]
 [ 7 48 16]
 [ 8 14 26]]
leave_one_out -> Exactitud: 0.72, Matriz de confusión:
[[51  3  5]
 [ 7 49 15]
 [ 7 12 29]]

Clasificador k-NN (k=5)
hold_out -> Exactitud: 0.74, Matriz de confusión:
[[17  0  2]
 [ 1 15  5]
 [ 1  5  8]]
10_fold -> Exactitud: 0.66, Matriz de confusión:
[[53  1  5]
 [ 5 47 19]
 [ 6 24 18]]
leave_one_out -> Exactitud: 0.70, Matriz de confusión:
[[52  1  6]
 [ 6 49 16]
 [ 6 19 23]]
```

Imagen 11

En el conjunto de datos *Wine*, se evaluaron cuatro clasificadores utilizando los métodos de validación *hold-out*, *10-fold cross-validation* y *leave-one-out cross-validation*. Los resultados muestran un desempeño variable entre los modelos, con diferencias más marcadas en comparación al conjunto de datos *Iris*, lo que refleja la mayor complejidad de este conjunto.

El clasificador 1-NN mostró un desempeño moderado, alcanzando una exactitud del 80% en el método *hold-out*. La matriz de confusión revela que los errores se distribuyeron entre las clases, particularmente con instancias de las clases 2 y 3 que fueron confundidas entre sí. En las validaciones cruzadas, su rendimiento disminuyó, obteniendo una exactitud del 73% en *10-fold* y 77% en *leave-one-out*. Los errores en estos casos también se concentraron en confusiones entre las clases 2 y 3, lo que sugiere que el clasificador tiene dificultades para diferenciar estas clases debido a posibles solapamientos en sus características.

Por otro lado, el clasificador Naive Bayes tuvo un rendimiento sobresaliente en todos los métodos de validación. En el método *hold-out*, logró una exactitud perfecta del 100%, clasificando correctamente todas las instancias en el conjunto de prueba. En *10-fold* y *leave-one-out*, obtuvo una exactitud del 98%, con apenas unos pocos errores. Las matrices de confusión indican que las confusiones fueron mínimas, reflejando que Naive Bayes se adapta muy bien a este conjunto de datos. Este excelente desempeño puede atribuirse a que las características de este conjunto parecen ajustarse bien a la suposición de independencia entre variables que realiza este modelo.

El clasificador k-NN con k=3 presentó un desempeño inferior en comparación con Naive Bayes y 1-NN. En el método *hold-out*, alcanzó una exactitud del 74%, con errores significativos en la clasificación de instancias de las clases 2 y 3. En las validaciones cruzadas, su desempeño fue aún menor, con una exactitud del 70% en *10-fold* y 72% en *leave-one-out*. Las matrices de confusión revelan un patrón consistente de confusiones entre las clases 2 y 3, lo que indica que el modelo no logra captar adecuadamente las diferencias entre estas clases cuando se consideran tres vecinos.

El clasificador k-NN con k=5 tuvo resultados similares al modelo con k=3, con una exactitud del 74% en el método *hold-out*. Sin embargo, su desempeño disminuyó notablemente en las validaciones cruzadas, con una exactitud del 66% en *10-fold* y 70% en *leave-one-out*. Las confusiones entre las clases 2 y 3 fueron incluso más pronunciadas que en el caso de k=3, lo que sugiere que considerar más vecinos introduce ruido adicional que afecta negativamente la capacidad del modelo para discriminar entre clases.

En general, los resultados muestran que el clasificador Naive Bayes es claramente el más adecuado para el conjunto de datos *Wine*, logrando un desempeño excepcional en todos los métodos de validación. Los clasificadores basados en vecinos (1-NN y k-NN) presentan limitaciones significativas, especialmente en la diferenciación entre las clases 2 y 3, lo que afecta su desempeño global. Esto puede deberse a la naturaleza del conjunto de datos, que podría presentar una distribución compleja o características solapadas entre estas clases.

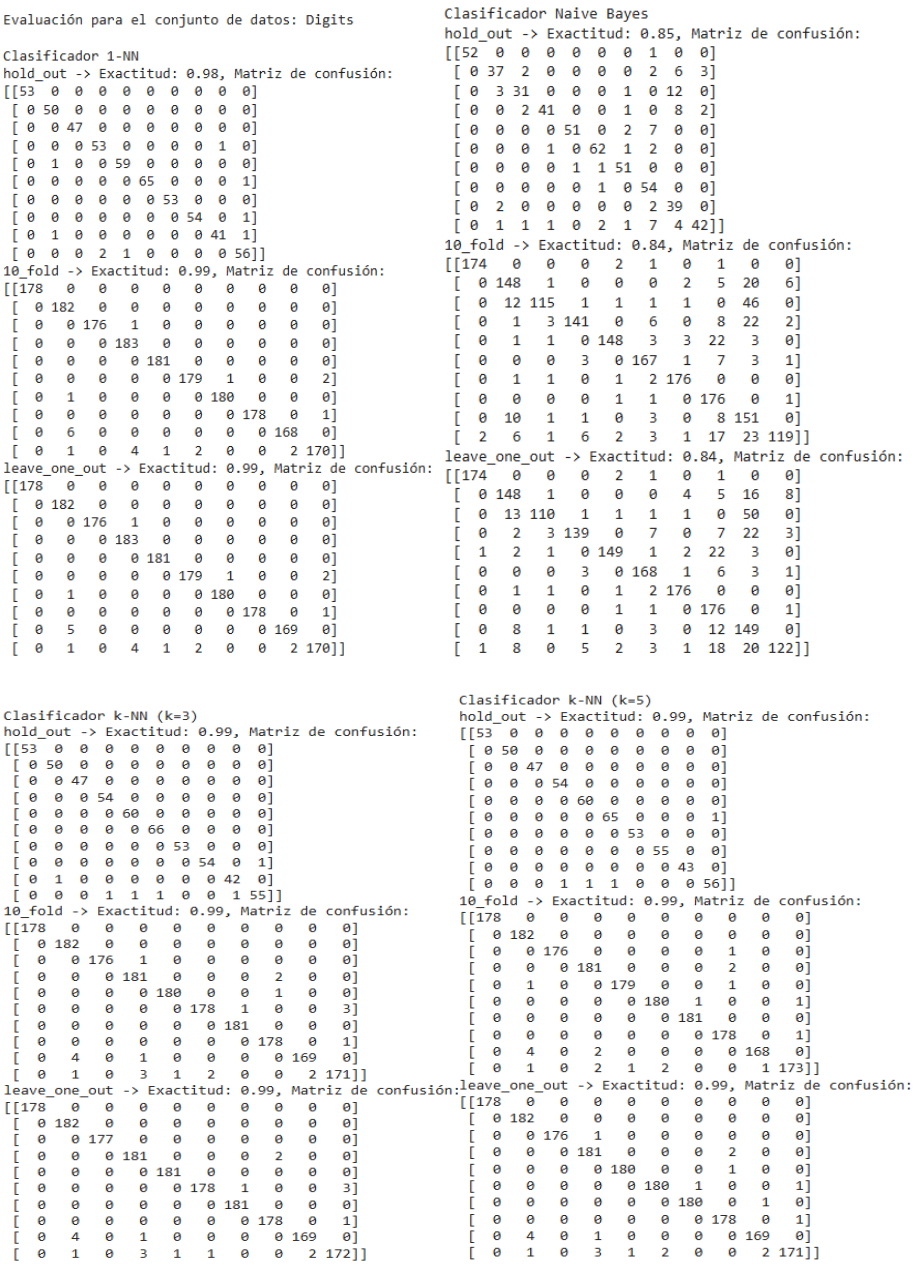

Imagen 12

En el conjunto de datos Digits, se evaluaron diferentes clasificadores bajo tres métodos de validación: *hold-out*, *10-fold cross-validation* y *leave-one-out*. Los resultados obtenidos muestran diferencias significativas en la precisión y en las matrices de confusión según el modelo utilizado.

El clasificador 1-NN obtuvo un excelente desempeño, con una exactitud del 98% en *hold-out*, y del 99% tanto en *10-fold cross-validation* como en *leave-one-out*. La matriz de confusión evidencia un alto nivel

de precisión, con muy pocos errores de clasificación en todas las validaciones. Este rendimiento confirma que el modelo es efectivo para separar correctamente las clases, incluso en los escenarios de validación más exigentes.

El clasificador Naive Bayes, en comparación, tuvo un desempeño más modesto, con una exactitud del 85% en *hold-out* y del 84% tanto en *10-fold cross-validation* como en *leave-one-out*. Las matrices de confusión indican errores de clasificación más frecuentes, especialmente en clases que presentan distribuciones similares, como la confusión entre dígitos vecinos. Este comportamiento refleja las limitaciones del supuesto de independencia entre características del modelo Naive Bayes, que en este caso no captura adecuadamente la complejidad de los datos.

Por otro lado, los clasificadores k-NN con valores de k=3k=3k=3 y k=5k=5k=5 mostraron un desempeño muy similar al de 1-NN, con exactitudes del 99% en todos los métodos de validación. Las matrices de confusión muestran que estos modelos tienen una capacidad notable para identificar correctamente las clases, con errores prácticamente inexistentes. Esto sugiere que la elección de un kkk mayor introduce una leve robustez frente a ruidos en los datos, sin sacrificar precisión.

En general, los resultados demuestran que los modelos basados en vecinos más cercanos (1-NN, k-NN) son los más adecuados para el conjunto de datos Digits, debido a su alto rendimiento en exactitud y consistencia entre los métodos de validación. Mientras tanto, Naive Bayes, aunque eficiente en términos de velocidad y simplicidad, no es capaz de competir con los métodos basados en distancias para este problema.

Conclusiones

En general, los clasificadores muestran diferencias en su desempeño según las características de cada conjunto de datos. Para el conjunto **Digits**, los modelos basados en vecinos más cercanos (1-NN y k-NN) sobresalieron por su alta precisión y consistencia, mientras que Naive Bayes fue menos eficaz debido a sus supuestos de independencia entre características. En el conjunto **Iris**, tanto k-NN como Naive Bayes lograron buenos resultados, mostrando que ambos pueden manejar datos bien estructurados, aunque k-NN demostró mayor robustez. Finalmente, en el conjunto **Wine**, los clasificadores también tuvieron un desempeño destacado, siendo k-NN nuevamente el más preciso, mientras que Naive Bayes presentó limitaciones frente a clases más complejas. En resumen, k-NN demostró ser el modelo más confiable en todos los conjuntos, mientras que Naive Bayes fue menos competitivo en problemas con relaciones complejas entre características.