Análisis del Dataset Iris: Comparativa Multilenguaje

Big Data e Inteligencia Artificial

Yeray Hurtado Dragón

Octubre 2025

Índice

1. Introducción

El presente documento analiza el **dataset Iris** utilizando el algoritmo **K-Nearest Neighbors (KNN)** en tres lenguajes de programación: **Python**, **R** y **Java (Weka)**.

Cada lenguaje ejecuta el mismo flujo de trabajo:

- Matriz de correlación.
- Gráfico de dispersión.
- Modelo KNN con librerías.
- Implementación manual del KNN.
- Matriz de confusión y precisión final.

El objetivo es comparar el rendimiento, facilidad de implementación y resultados obtenidos en cada entorno.

2. Algoritmo K-Nearest Neighbors (KNN)

El algoritmo **K-Nearest Neighbors (KNN)** es un método de clasificación supervisada basado en la idea de que los objetos similares tienden a estar cerca en el espacio de características. Para clasificar una nueva instancia, el algoritmo:

1. Calcula la distancia entre la nueva instancia y todas las observaciones del conjunto de entrenamiento.
2. Selecciona los k vecinos más cercanos según la métrica de distancia.
3. Asigna la clase más frecuente entre estos vecinos.

2.1. Distancia Euclidiana

La métrica más común es la **distancia euclidiana**, que mide la separación geométrica entre dos puntos p y q en un espacio de n dimensiones:

$$d(p, q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

donde:

- p_i y q_i son los valores de la i-ésima característica de los puntos p y q.
- n es el número de características.

Esta distancia determina qué vecinos son los más cercanos y, por tanto, cuáles influyen en la clasificación de la nueva instancia.

2.2. Ventajas y limitaciones

Ventajas:

- Simple de entender e implementar.
- Funciona bien con problemas de clasificación multi-clase.

Limitaciones:

- Sensible a la escala de las características (recomendable normalizar datos).
- Puede ser lento en conjuntos de datos grandes.
- La elección de k influye directamente en la precisión.

3. Descripción del dataset y metodología

El dataset **Iris** contiene 150 observaciones de flores pertenecientes a tres especies: *Iris-setosa*, *Iris-versicolor* y *Iris-virginica*. Cada registro posee cuatro características numéricas:

- Largo y ancho del sépalo (`SepalLengthCm`, `SepalWidthCm`)
- Largo y ancho del pétalo (`PetalLengthCm`, `PetalWidthCm`)

Metodología general:

1. Carga y limpieza de datos.
2. Exploración (correlaciones y gráficos).
3. Preparación de datos (eliminación de columnas irrelevantes, codificación).
4. Entrenamiento con KNN (con librerías).
5. Implementación manual del KNN.
6. Evaluación con matriz de confusión y precisión.

4. Implementación en Python (Google Colab)

4.1. Matriz de correlación

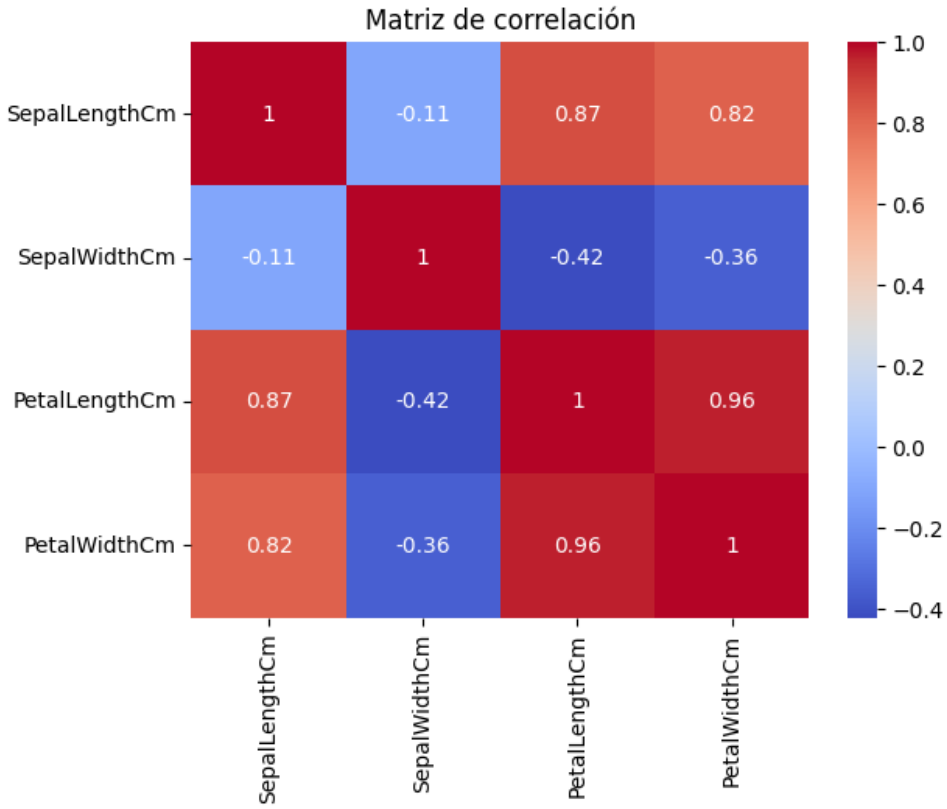

Figura 1: Matriz de correlación del dataset Iris en Python

Interpretación: Alta correlación entre `PetalLengthCm` y `PetalWidthCm`, indicando su relevancia en la clasificación.

4.2. Gráfico de dispersión

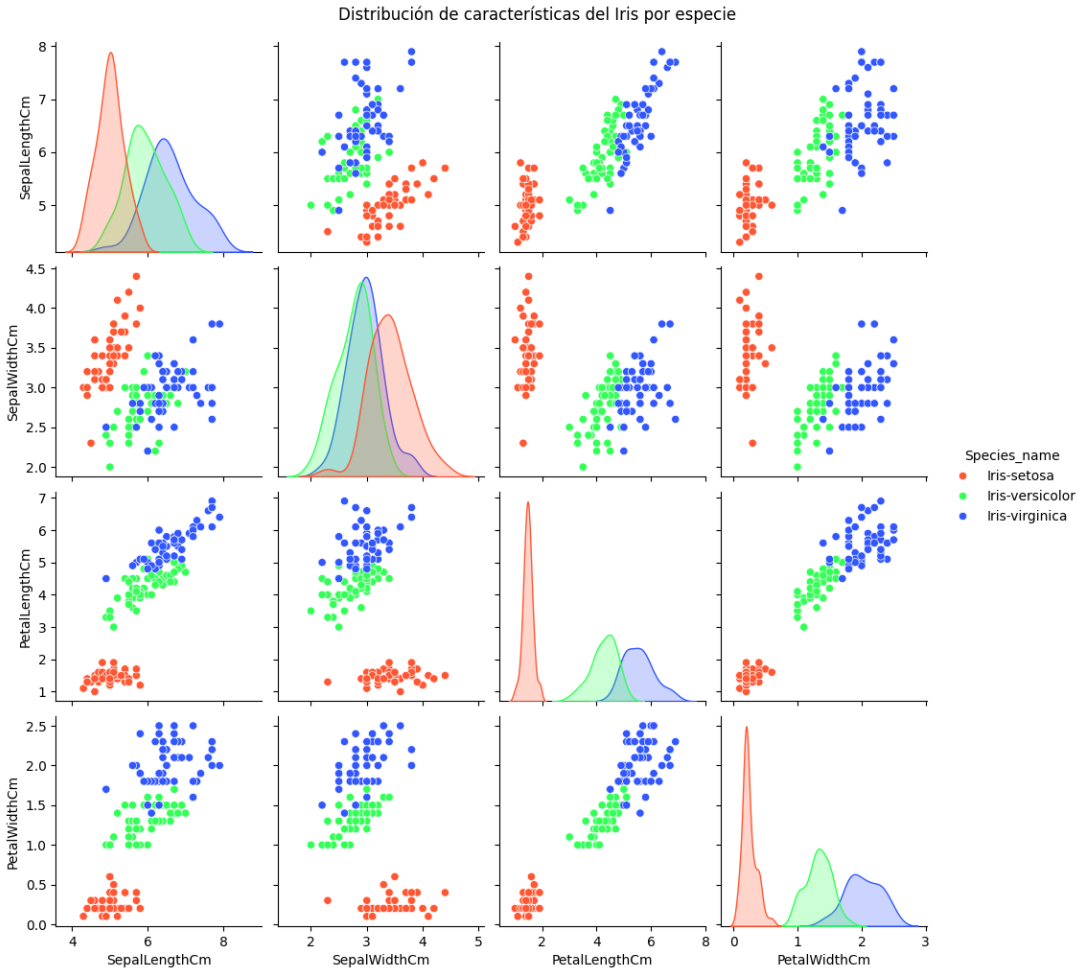

Figura 2: Gráfico de dispersión de las características del Iris (Python)

Interpretación: *Setosa* se separa claramente, mientras que *Versicolor* y *Virginica* presentan solapamiento.

4.3. KNN con librerías

```
# Caracter sticas y etiquetas
X = df[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', '
    PetalWidthCm']]
y = df['Species']  # O nombres de especies

# Divisi n entrenamiento/prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.3, random_state=123)

# Entrenamiento KNN
knn = KNeighborsClassifier(n_neighbors=27)
knn.fit(X_train, y_train)

# Precisi n
accuracy = knn.score(X_test, y_test) * 100
print(f"Precisi n del modelo KNN: {accuracy:.2f}%")
```

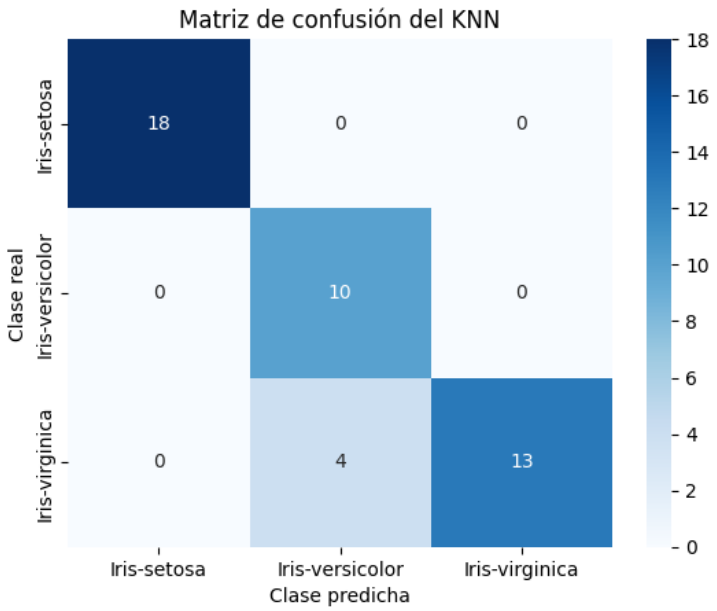

Figura 3: Matriz de confusión del modelo KNN (Python)

Precisión: 91.11 %

4.4. KNN implementado manualmente

```python
import numpy as np
from collections import Counter

# Distancia euclidiana
def euclidean_distance(a, b):
    return np.sqrt(np.sum((a - b) ** 2))

# Predicci n para una instancia
def predict_one(x, X_train, y_train, k):
    distances = [euclidean_distance(x, xi) for xi in X_train]
    k_indices = np.argsort(distances)[:k]
    k_labels = [y_train[i] for i in k_indices]
    return Counter(k_labels).most_common(1)[0][0]

# Predicci n para el conjunto de prueba
y_pred = [predict_one(x, X_train, y_train, k=27) for x in X_test]

# Precisi n
accuracy = np.mean(y_pred == y_test) * 100
print(f"Precisi n KNN manual: {accuracy:.2f}%")
```

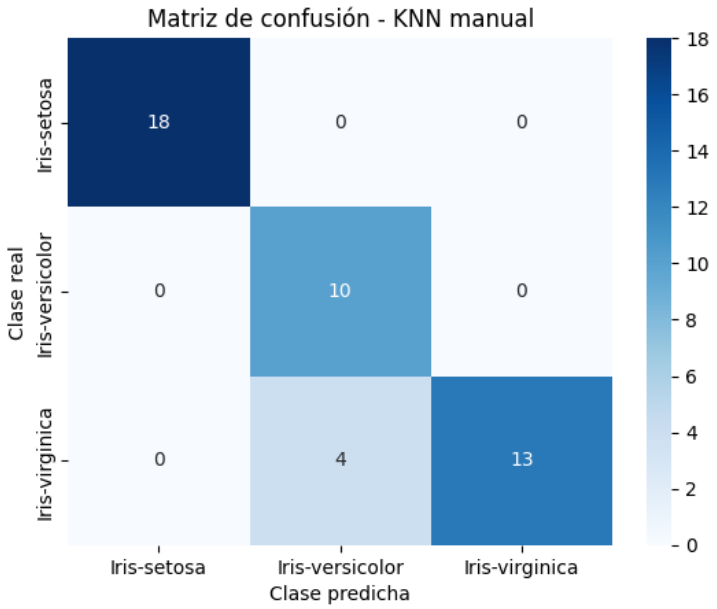

Figura 4: Matriz de confusión del modelo KNN sin librerias (Python)

Precisión: 91.11 %

5. Implementación en R (Posit Cloud)

5.1. Matriz de correlación

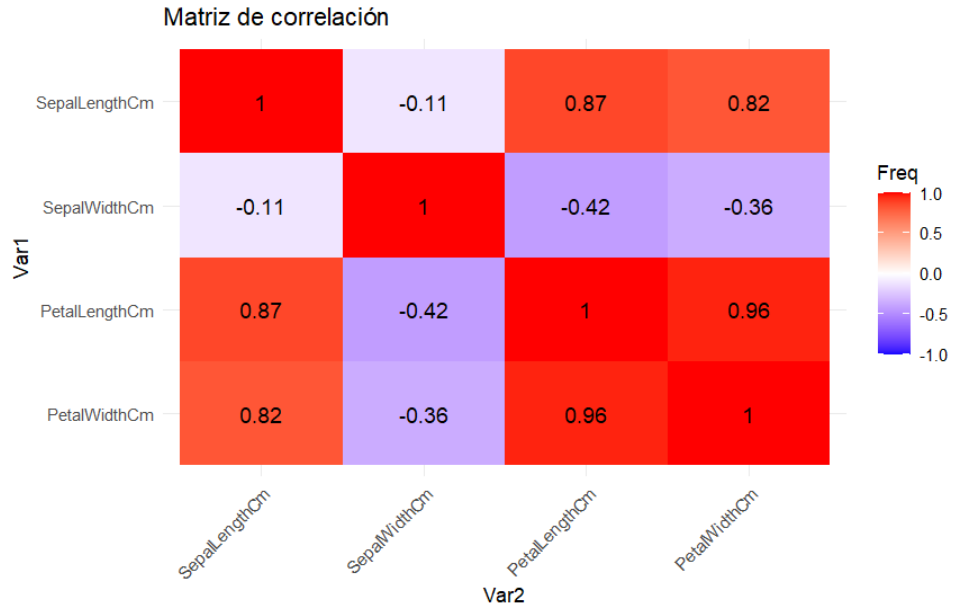

Figura 5: Matriz de correlación en R

5.2. Gráfico de dispersión

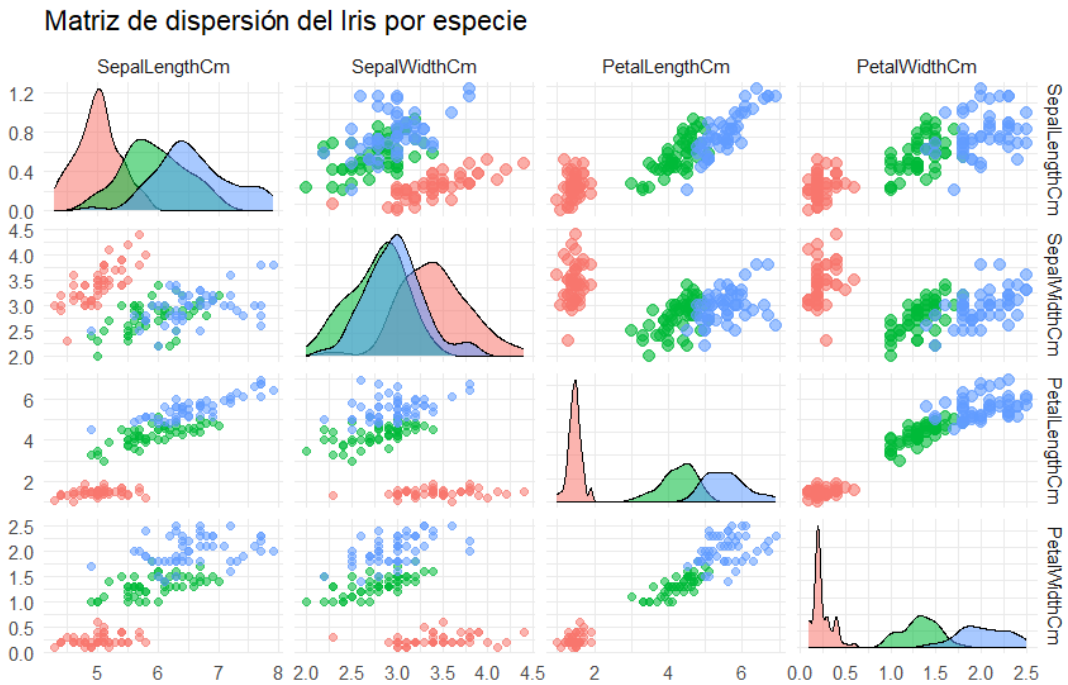

Figura 6: Distribución de características por especie (R)

5.3. KNN con librerías

```r
if(!require(class)) install.packages("class")
library(class)

# Variables predictoras y variable objetivo
X <- dataset[, 1:4]
y <- dataset$Species
# Divisi n en entrenamiento y prueba
set.seed(123)
train_index <- sample(1:nrow(dataset), 0.7 * nrow(dataset))
X_train <- dataset[train_index, 1:4]
X_test  <- dataset[-train_index, 1:4]
y_train <- dataset$Species[train_index]
y_test  <- dataset$Species[-train_index]

# Entrenamiento y predicci n con KNN
library(class)
y_pred <- knn(X_train, X_test, y_train, k = 27)

# Precisi n
accuracy <- mean(y_pred == y_test)
cat("Precisi n del modelo KNN:", round(accuracy * 100, 2), "%\n")
```

Precisión: 97.78 %

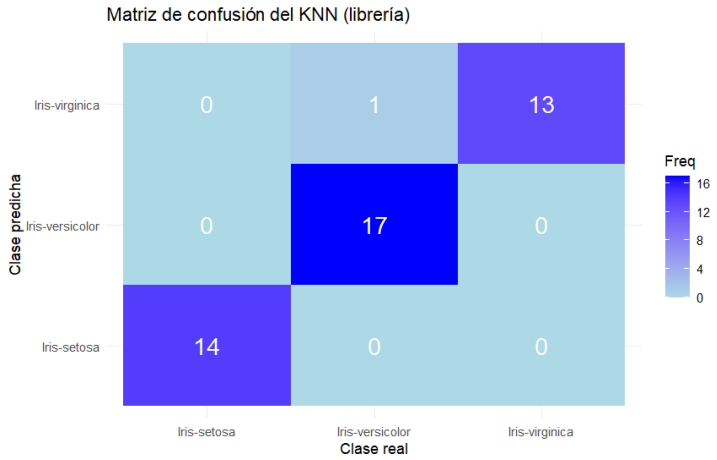

Figura 7: Matriz de confusión del modelo KNN (R)

5.4. KNN sin librerías (manual)

```r
euclid <- function(a, b){
    sqrt(sum((a - b)^2))
  }

  k <- 27
  y_pred_manual <- vector()
  for(i in 1:nrow(X_test)){
    distances <- apply(X_train, 1, function(row) euclid(row, X_test
       [i, ]))
    nearest <- order(distances)[1:k]
    labels <- y_train[nearest]
    y_pred_manual[i] <- names(sort(table(labels), decreasing = TRUE
       ))[1]
  }

  y_pred_manual <- factor(y_pred_manual, levels = levels(y_test))
  accuracy_manual <- mean(y_pred_manual == y_test)
  cat("Precisi n del KNN manual:", round(accuracy_manual * 100, 2)
     , "%\n")
```

Precisión: 97.78 %

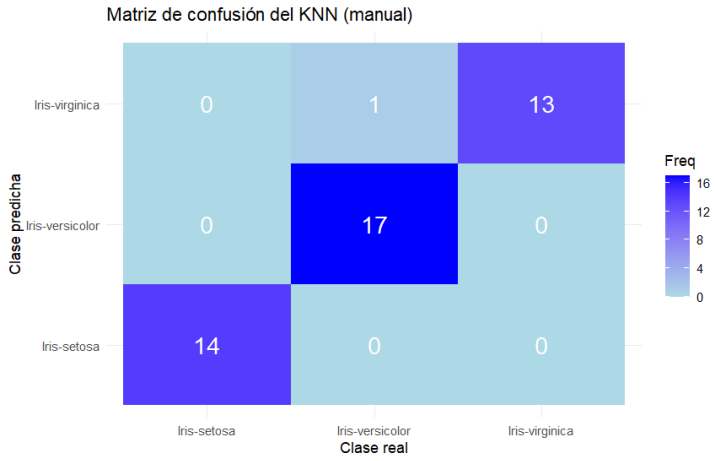

Figura 8: Matriz de confusión del modelo KNN (R)

6. Implementación en Java (Weka)

6.1. Matriz de correlación

Cuadro 1: Matriz de correlación en Weka

	SepalLength	SepalWidth	PetalLength	PetalWidth
SepalLength	1,00	0,72	-0,40	0,88
SepalWidth	0,72	1,00	-0,11	0,87
PetalLength	-0,40	-0,11	1,00	-0,42
PetalWidth	0,88	0,87	-0,42	1,00

Interpretación: Se observan correlaciones altas entre `PetalLength` y `PetalWidth`, similares a Python y R, lo que indica que estas características son relevantes para la clasificación.

6.2. Gráfico de dispersión

Nota: Weka no genera gráficos de dispersión automáticamente. Para análisis visual se recomienda exportar los datos a Python o R. Aquí se omite el gráfico y se presentan los resultados directamente.

6.3. KNN con librerías

```
// Entrenamiento KNN con librer a IBk (Weka)
int k = 5;
IBk knn = new IBk(k);
knn.buildClassifier(train);

// Evaluaci n del modelo
Evaluation eval = new Evaluation(train);
eval.evaluateModel(knn, test);

System.out.println("Precisi n: " + String.format("%.2f", eval.
   pctCorrect()) + "%");
System.out.println(eval.toMatrixString());
```

Precisión: 97,78 %

Cuadro 2: Matriz de confusión del modelo KNN (Weka)

Predicted \ Actual	Setosa	Versicolor	Virginica
Setosa	14	0	0
Versicolor	0	19	0
Virginica	0	0	12

6.4. KNN implementado manualmente

Implementación propia de la distancia euclidiana:

```
int k = 27;
int aciertos = 0;

for (int i = 0; i < test.numInstances(); i++) {
    Instance actual = test.instance(i);

    // Calcular distancia a todos los ejemplos de entrenamiento
    double[] distancias = new double[train.numInstances()];
    String[] clases = new String[train.numInstances()];
    for (int j = 0; j < train.numInstances(); j++) {
        distancias[j] = euclideanDistance(actual, train.instance(j));
        clases[j] = train.instance(j).stringValue(train.classIndex());
    }

    // Seleccionar los k vecinos m s cercanos
    int[] vecinos = new int[k];
    for (int v = 0; v < k; v++) {
        double minDist = Double.MAX_VALUE;
        int minIndex = -1;
        for (int j = 0; j < distancias.length; j++) {
            if (distancias[j] < minDist) {
                minDist = distancias[j];
                minIndex = j;
            }
        }
        vecinos[v] = minIndex;
        distancias[minIndex] = Double.MAX_VALUE; // evitar repetir
    }

    // Votaci n de las clases vecinas
    int votosSetosa = 0, votosVersicolor = 0, votosVirginica = 0;
    for (int v = 0; v < k; v++) {
        String clase = clases[vecinos[v]];
        if (clase.equals("Iris-setosa")) votosSetosa++;
        else if (clase.equals("Iris-versicolor")) votosVersicolor++;
        else votosVirginica++;
    }

    // Determinar clase m s votada
    String prediccion;
    if (votosSetosa > votosVersicolor && votosSetosa > votosVirginica)
        prediccion = "Iris-setosa";
    else if (votosVersicolor > votosVirginica)
        prediccion = "Iris-versicolor";
```

```
45      else
46          prediccion = "Iris-virginica";
47
48      // Comparar con clase real
49      if (prediccion.equals(actual.stringValue(test.classIndex())))
50          aciertos++;
51  }
52
53  double precision = 100.0 * aciertos / test.numInstances();
54  System.out.println("Precisi n KNN manual: " + String.format("%.2f"
    , precision) + "%");
```

Precisión manual: 97,78 %

Cuadro 3: Matriz de confusión del modelo KNN manual (Weka)

Predicted \ Actual	Setosa	Versicolor	Virginica
Setosa	14	0	0
Versicolor	0	18	1
Virginica	0	0	12

Limitaciones de Weka:

- No genera gráficos de dispersión ni visualizaciones avanzadas de forma nativa.
- La matriz de correlación se presenta como tabla, no como heatmap.
- Más adecuado para evaluación rápida de modelos y pruebas de clasificación que para análisis gráfico exhaustivo.

7. Comparativa multilenguaje

Cuadro 4: Resumen comparativo entre lenguajes

Lenguaje	**Precisión (librería)**	**Precisión (manual)**	**Ventajas**	**Desventajas**
Python	91.11 %	91.11 %	Fácil de implementar, ecosistema amplio	Requiere optimización en grandes volúmenes de datos
R	97.78 %	97.78 %	Potente en análisis estadístico y visualización	Sintaxis menos intuitiva para principiantes
Java (Weka)	97.78 %	97.78 %	Eficiente, robusto y con entorno visual	Limitado en visualización avanzada y flexibilidad

Análisis: Los tres lenguajes presentan resultados consistentes, superando el 90 % de precisión. **R** y **Java (Weka)** alcanzan el mejor rendimiento (97.78 %), mientras que **Python** ofrece una implementación más sencilla y versátil. Las diferencias principales se deben a las bibliotecas disponibles y la facilidad para la visualización de resultados.

8. Conclusiones

El experimento demuestra que el dataset Iris puede analizarse eficazmente en múltiples lenguajes con resultados consistentes. Las diferencias radican principalmente en el entorno de desarrollo y la sintaxis:

- **Python** es ideal para aprendizaje automático y prototipado rápido.
- **R** sobresale en análisis estadístico y visualización de datos.
- **Java (Weka)** ofrece eficiencia y entorno gráfico accesible.

La implementación manual del algoritmo refuerza la comprensión teórica del KNN y su dependencia de la distancia euclidiana. En conjunto, los tres entornos confirman la capacidad del KNN para clasificar correctamente las especies del Iris con más del 90 % de precisión.