UMAG Universidad de Magallanes | Facultad de Ingeniería

Diseño de Algoritmos - Algoritmos de aprendizaje automacio en C.

Miguel Loaiza[†], **Duvan Figueroa**[†] **and Diego Sanhueza**[†]

[†]*Universidad de Magallanes*

This manuscript was compile on July 8, 2025

Abstract

En este informe se describe el desarrollo e implementación de algoritmos de aprendizaje automático utilizando el lenguaje C. El objetivo principal es analizar y comprender el funcionamiento de estos algoritmos a un nivel fundamental, entendiendo su lógica y estructura desde cero. Se llevaron a cabo implementaciones de algoritmos como K-Nearest Neighbors (KNN), K-Means y regresión lineal.

Keywords: *Algoritmos, Lenguaje C, Machine Learning, Inteligecia Artificial, Patrones*

Contents

1. Introducción

En la actualidad, conceptos como **Machine Learning**, **Inteligencia Artificial** y **Regresión Lineal** han dejado de ser exclusivos del ámbito académico para integrarse con naturalidad en la vida diaria. Desde las recomendaciones personalizadas en plataformas de streaming hasta los sistemas de diagnóstico médico asistido por computadora, estas tecnologías están transformando silenciosamente una gran gama de sectores: salud, finanzas, comercio, logística, educación e incluso la gastronomía.

El auge del aprendizaje automático (Machine Learning) ha despertado un creciente interés en comprender cómo funcionan los algoritmos que permiten a los sistemas "aprender" a partir de datos.

Estudiar y aplicar los fundamentos del **machine learning**, particularmente desde unn lenguaje de bajo nivel como C, ofrece una oportunidad invaluable: permite descomponer cada proceso de aprendizaje en sus elementos más básicos, desde el manejo eficiente de estructuras de datos hasta la implementación explícita de cálculos matemáticos como el **descenso de gradiente** o las **métricas de distancia**. Este enfoque no solo fortalece la comprensión teórica, sino que también desarrolla habilidades prácticas fundamentales en ingeniería de software, análisis de eficiencia computacional y diseño de sistemas predictivos.

Profundizar en estos algoritmos clásicos (k-nearest neighboors, regresión lineal o k-means) permite visualizar con claridad el potencial del **machine learning**.

2. Objetivos

2.1. Objetivo General

El objetivo principal de este proyecto es desarrollar, implementar y analizar algoritmos fundamentales de **machine learning** en lenguaje C, enfocándose en la comprensión de sus principios internos y en la evaluación empírica de su desempeño mediante experimentación con conjuntos de datos diversos. Se pretende examinar la eficacia de los algoritmos y su comportamiento ante distintas variaciones de entrada, métricas de evaluación y configuraciones de parámetros.

2.2. Objetivos Específicos

- Comprender en detalle el funcionamiento interno de los algoritmos que se van a implementar.
- Implementar los algoritmos requeridos.
- Ilustrar el proceso de implementación en un lenguaje de bajo nivel como C, destacando las ventajas y desafíos que esto implica.
- Analizar la complejidad de los algoritmos para el mejor y peor caso.
- Realizar análisis de los gráficos.

3. Descripción del proyecto

A continuación se detalla el alcance del trabajo realizado, los conjuntos de datos utilizados para validar los algoritmos, y las principales consideraciones metodológicas y limitaciones encontradas durante el desarrollo del proyecto.

3.1. Alcance

El proyecto tiene como propósito la implementación desde cero de tres algoritmos clásicos de **machine learning**:

- K-Nearest Neighbors (K-NN) para clasificación.
- Regresión Lineal para predicción de variables continuas.
- K-Means para agrupamiento no supervisado.

Todos los algoritmos fueron desarrollados íntegramente en el lenguaje de programación C, sin utilizar bibliotecas especializadas en **machine learning**. Esta elección responde al objetivo de fomentar una comprensión profunda de la estructura interna que rigen los algoritmos.

El alcance del proyecto contempla no solo la implementación funcional de los algoritmos, sino también su evaluación empírica a partir de datos reales o sintéticos. Se busca analizar la capacidad predictiva y el comportamiento frente a distintos parámetros.

3.2. Datos utilizados

Para probar y validar las implementaciones, se emplearon **conjuntos de datos previamente definidos**, seleccionados por su simplicidad y claridad estructural. Estos datasets permiten visualizar con facilidad el comportamiento del modelo y verificar la corrección de los resultados generados.

Entre los datos utilizados se incluyen ejemplos representativos con dimensiones reducidas, lo que facilita el análisis paso a paso del funcionamiento de los algoritmos. Además, su estructura uniforme permite aplicar técnicas básicas de normalización, partición de datos y cálculo de métricas sin introducir ruido innecesario en los resultados.

Cabe destacar que estos conjuntos de datos no tienen como fín la construcción de modelos complejos, sino que funcionan como **pruebas didácticas** para validar cada componente de los algoritmos y comprender cómo responden ante distintas configuraciones y características de entradas.

3.3. Consideraciones y limitaciones

Durante el desarrollo del proyecto se tomaron diversas **decisiones metodológicas orientadas a simplificar el entorno de trabajo**, con el fin de focalizar los esfuerzos en la lógica algorítmica y en la consolidación de conocimientos fundamentales.

Entre las principales consideraciones se incluyen:

- La implementación sobre conjuntos de datos de tamaños reducidos para facilitar la depuración y el análisis.
- Nula implementación de librerías de machine learning.
-
- Selección de parámetros específicos para el buen funcionamiento de los algorítmos

En cuanto a las limitaciones, se reconoce que el sistema desarrollado no está diseñado para trabajar con grandes volúmenes de datos ni con estructuras de entrada altamente complejas.

4. Diseño del Sistema

En esta sección se describe la arquitectura general del sistema desarrollado, su estructura interna y la lógica de implementación de los principales componentes. Se detallan los módulos en los que se organiza el proyecto, el flujo de datos desde la entrada hasta la salida, el diseño específico de los algoritmos de aprendizaje automático implementados, y el mecanismo de interacción con el usuario a través de la línea de comandos. Asimismo, se discuten aspectos clave del diseño como la gestión de memoria, el manejo de errores y la reproducibilidad de resultados.

4.1. Arquitectura general

El sistema fue diseñado bajo un enfoque modular, lo que permite una separación clara de responsabilidades y facilita el mantenimiento, la reutilización de código y la extensibilidad futura. Cada algoritmo de machine learning (K-Nearest Neighbors, Regresión Lineal y K-Means) se encuentran implementado en módulos independientes, con sus respectivos archivos .c y .h. Esta separación no solo mejora la legibilidad, sino que permite trabajar en paralelo en distintas funcionalidades sin generar conflictos o dependencias innecesaria.

El control principal del programa se concentra en el main.c, encargado de gestionar el flujo general de ejecución: lectura, preprocesamiento de datos, interpretación de argumentos desde la línea de comandos, invocación del algoritmo correspondiente y exportación de los resultados.

4.2. Estructura del código

El proyecto está estructurado en archivos que reflejan una clara división funcional. La organización es la siguiente:

- main.c: Archivo principal del sistema. Gestiona los argumentos de entrada mediante **parse_args()**, carga los datos desde archivos CSV, selecciona el algoritmo adecuado y ejecuta la lógica de entrenamiento y predicción.
- knn.c / knn.h: Módulo que implementa el algoritmo de clasificación por k vecinos más cercanos.
- lr.c / lr.h: Contiene la implementación del modelo de regresión lineal con distintas técnicas del ajuste.
- km.c / km.h: Implementa el algoritmo de agrupamiento K-Means con inicialización mejorada.
- utils.c / utils.h: Conjunto de funciones auxiliares, como lectura y escritura de archivos CSV, normalización de datos, cálculo de métricas y funciones estadísticas generales.

4.3. Flujo de datos

El procesamiento de los datos sigue un flujo claro y estructurado:

1. Carga de datos: Se realiza mediante la función **load_csv_data()**, que abre el archivo CSV y determina su dimensión utilizando **csv_dimensions()**. Luego, se alocan las matrices necesarias y se parsea cada línea para extraer los valores numéricos y etiquetas correspondientes.
2. Preprocesamiento: Los datos son normalizados utilizando la técnica de **escalado Min-Max**, a través de la función **normalize_csv_data()**. Esta etapa también se encarga de manejar valores faltantes sustituyéndolos por media de cada columna.
3. Ejecución del algoritmo: Una vez preprocesados, los datos son enviados al módulo del algoritmo seleccionado. Cada algoritmo opera sobre su propia copia de los datos y produce resultados que son posteriormente evaluados y exportados.
4. Exportación de resultados: Los resultados (métricas, predicciones, centroides, etc.) se escriben automáticamente en archivos **.csv** dentro del directorio **stats/**, lo que permite su posterior análisis o visualización externa.

4.4. Diseño de los algoritmos

El sistema incorpora tres algoritmos representativos del machine learning:

K-Nearest Neighbors (KNN):

- Soporta múltimples métricas de distancia: Euclidiana y Manhattan.
- Incluye variantes ponderadas según la distancia.
- Utiliza división de datos en tres subconjuntos: entrenamiento (60%), validación (20%) y prueba (20%).
- Emprea el algoritmo QuickSort para ordenar las distancias y seleccionar los vecinos más cercanos de forma eficiente.
- Muestra distintas métricas Recall, F1-Score y Matriz de confusión.

Regresión Lineal:

- Ofrece dos métodos de entrenamiento: descenso de gradiente e implementación basada en ecuaciones normales.
- Soporta regularización mediante Ridge (L2) y Lasso (L1), lo que permite mejorar la generalización dle modelo.
- Incluye evaluación mediante métricas como MSE, MAE Y R2

K-Means:

- Implementación basada en la técnica K-Means++ para una mejor inicialización de centroides, lo que reduce la sensibilidad al azar y mejora la velocidad de convergencia.
- Se define una tolerancia de convergencia que limita el movimiento de los centroides entre iteraciones.
- El número de iteraciones y el valor de k se configuran desde la línea de comandos.

4.5. Interacción mediante la línea de comandos

El sistema está diseñado para operar completamente desde la **línea de comandos**, sin necesidad de interfaz gráfica ni menús interactivos. Esto lo hace adecuado para ser integrado en scripts o pupelines automatizados. Se utiliza la función **getopt()** para el análisis de argumentos, y los comandos disponibles son:

- **-k <archivo> <k>:** Ejecuta KNN con k vecinos (acepta valores impares mayores que 0).
- **-l <archivo> <lr> <iter> <tol>:** Ejecuta Regresión Lineal con tasa de aprendizaje, número de iteraciones y tolerancia para el criterio de convergencia.
- **-m <archivo> <k> <iter> <tol>:** Ejecuta K-Means con k clusters, número de iteraciones y tolerancia.

4.6. Consideraciones de diseño

Durante la implementación se tomaron diversas decisiones para asegurar robustez, claridad y eficiencia básica del sistema.

Para poder gestionar la memoria, se utilizaron funciones especializadas como **csv_free()** y **matrix_free()** para liberar la memoria utilizada y prevenir fugas.

Se definieron funciones específicas para la gestión centralizada de errores, incluyendo validación de parámetros, detección de errores en archivos y manejo de fallos en la asignación dinámica de memoria.

Los resultados se almacenan de forma organizada en archivos .csv, lo que permite integrarse con herramientas externas de análisis o visualización.

5. Pruebas y Resultados Experimentales

5.1. K-Nearest Neighbors

5.1.1. Implementación

El algoritmo **K-Nearest Neighbors (KNN)** es una técnica de aprendizaje supervisado basada en instancias, ampliamente utilizada para tareas de clasificación y regresión. Su principio fundamental consiste en que una muestra desconocida se clasifica observando las clases de sus k vecinos más cercanos dentro del espacio de características. La predicción se realiza mediante votación de mayoría (clasificación) o un promedio ponderado (regresión).

En este proyecto, fue implementado en lenguaje C, lo que implicó el desarrollo manual de estructuras de datos, manejo explícito de memoria dinámica y control riguroso del flujo de dato. Las estructuras más relevantes para el funcionamiento del algoritmo son:

- **KNNClassifier:** Contiene los datos de entrenamiento y el parámetro k configurado por el usuario.
- **DistanceLabel:** Estructura auxiliar que almacena la distancia entre una muestra y sus vecinos, junto con su etiqueta de clase, para facilitar la ordenación y votación.

El procedimiento general de ejecución del algoritmo se encuentra en la función **exec_knn()**, e incluye distintas etapas. La primera de ellas es la carga de datos, que provienen directamente del archivo **iris.csv**, donde son cargados y normalizados (Min-Max). Posteriormente se dividen los conjuntos de datos por medio de una partición aleatoria, la división es en tres subconjuntos: entrenamiento (60%), validación (20%) y prueba (20%).

Para cada muestra dentro del conjunto de prueba, se calcula la distancia a todas las muestras de entrenamiento, se ordenan por cercanía y se seleccionan los k vecinos más próximos, luego se determina la clase por mayoría de votos o, si se especifica, por ponderación de distancia. Finalmente se calculan las métricas de rendimiento como precisión, matriz de confusión, recall y F1-score por clase.

5.1.2. Análisis de Complejidad

Sea n el número de muestras de entrenamiento, d el número de características (dimensiones), q el número de muestras de prueba y k el número de vecinos a considerar. El algoritmo KNN realiza, para cada muestra de prueba, el cálculo de distancias a todas las muestras de entrenamiento, ordena dichas distancias y selecciona los k vecinos más cercanos para votar o ponderar la clase.

El costo total del algoritmo es:

$$T(n,d,q,k) = T_{\text{distancias}}(n,d,q) + T_{\text{ordenamiento}}(n,q) + T_{\text{votación}}(k,q)$$

Donde:

$T_{\text{distancias}}(n,d,q) = O(q \cdot n \cdot d)$ (Cálculo de la distancia entre cada muestra de prueba y cada muestra de entrenamiento, en d dimensiones)

$T_{\text{ordenamiento}}(n,q) = O(q \cdot n \log n)$ (Ordenamiento de las n distancias para cada una de las q muestras de prueba, usando QuickSort)

$T_{\text{votación}}(k,q) = O(q \cdot k)]$ (Selección y votación/ponderación entre los k vecinos más cercanos para cada muestra de prueba)

La complejidad temporal total es:

$$T(n,d,q,k) = O(q \cdot n \cdot d + q \cdot n \log n + q \cdot k)$$

Análisis de casos:

- **Mejor caso:**

  $$T_{\text{mejor}}(n,d,q,k) = O(q \cdot n \cdot d)$$

  Si n es pequeño o si se utiliza $k = 1$ y un algoritmo de búsqueda eficiente (como árbol k-d para 1-NN), el ordenamiento puede evitarse o reducirse a $O(q \cdot \log n)$.

- **Peor caso:**

  $$T_{\text{peor}}(n,d,q,k) = O(q \cdot n \cdot d + q \cdot n \log n)$$

  Esto ocurre cuando n es grande y se requiere ordenar todas las distancias para cada muestra de prueba.

- **Caso promedio:**

  $$T_{\text{promedio}}(n,d,q,k) = O(q \cdot n \cdot d + q \cdot n \log n)$$

  En la práctica, para valores moderados de n y q, el algoritmo es eficiente, pero para grandes volúmenes de datos puede volverse costoso.

Espacio: El uso de memoria es $O(n \cdot d)$ para almacenar los datos de entrenamiento y $O(n)$ adicional para el arreglo de distancias en cada predicción.

5.1.3. Configuración del Algoritmo

La configuración adoptada para esta evaluación fue la siguiente:

- **Dataset:** iris.csv (150 muestras, 4 características y 3 clases)
- **Partición de datos:** 60% entrenamiento, 20% validación, 20% prueba.
- **Valor de k:** 1 (un solo vecino más cercano)
- **Métricas de distancia:** Euclidiana, Manhattan, Euclidiana ponderada, Manhattan ponderada.
- **Normalización:** Min-Max por característica.
- **Exportación:** Resultados guardados en archivos CSV con columnas: id, real, predicción, métrica, k.

5.1.4. Resultados

El modelo fue evaluado utilizando las cuatro variantes de distancia mencionadas. En todos los casos, la precisión alcansada fue perfecta (1.0000), lo que indica que todas las muestras del conjunto de prueba fueron clasificadas correctamente. Este resultado es coherente con las propiedades del dataser Iris, que es conocido por su alta separabilidad entre clases.

```
    Real 1, Predicci n: 1
    Real 2, Predicci n: 2
    Real 2, Predicci n: 2
    Real 1, Predicci n: 1
    Real 0, Predicci n: 0

```

Code 1. Formato de salida terminal 5 predicciones

5.1.5. Análisis de la Matriz de Confusión

La siguiente matriz de confusión resume el rendimiento del modelo en el conjunto de prueba:

Real/Pred	Pred 0	Pred 1	Pred 2
Real 0	8	0	0
Real 1	0	11	0
Real 2	0	0	1

Table 1. Matriz de Confusión K-NN

Todas las instancias fueron clasificadas correctamente, lo que se traduce en métricas perfectadas para cada clase:

- Clase 0: Precisión: 1.0000, Recall: 1.0000, F1-Score: 1.0000
- Clase 1: Precisión: 1.0000, Recall: 1.0000, F1-Score: 1.0000
- Clase 2: Precisión: 1.0000, Recall: 1.0000, F1-Score: 1.0000

5.1.6. Interpretación de Resultados

El rendimiento perfecto del moidelo bajo todas la métricas es un resultado esperable dado el contexto. El dataset Iris es conocido por su simplicidad estructural y su clara separación entre clases, lo que lo convierte en un caso ideal para evaluar algoritmos de clasificación como KNN. Además, la aplicación de técnicas de normalización y la elección adecuada de métricas de distancia contribuyen a que el algoritmo logre una predicción precisa incluso con valores bajos de k.

El uso de k=1 hace que el modelo base su decisión en el único vecino más cercano, lo que maximiza la sensibilidad a pequeñas variaciones o ruido en los datos. En datasets limpios y bien estructurados como Iris, esto puede resultar en una alta precisión, como se observó. Sin embargo, en contextos reales con datos ruidosos o mal etiquetados, un valor tan bajo de k puede aumentar el riesgo de sobreajuste y provocar decisiones erráticas. Por ello, la elección de k suele considerarse hiperparámetro sensible, cuya configuración debe definirse a partir de validaciones cruzadas y análisis exploratorios del dataset.

Finalmente, la exportación automatizada de los resultados a archivos CSV permite realizar comparaciones entre diferentes ejecuciones del algoritmo, visualizar gráficamente el comportamiento del clasificador y registrar experimentos para replicabilidad. Esta característica, sumada a la modularidad del sistema, convierte la implementación en una herramienta flexible para análisis predictivo básico y para propósitos educativos.

5.2. Regresión Lineal

5.2.1. Implementación

El algoritmo **Regresión Lineal** es una técnica de aprendizaje supervisado utilizada para predecir variables continuas a partir de un conjunto de características. En este proyecto, la regresión lineal fue implementada en C, permitiendo el entrenamiento mediante dos métodos: descenso de gradiente y ecuaciones normales. Se incluyó soporte para regularización Ridge (L2) y Lasso (L1), así como normalización Min-Max de los datos.

Las estructuras principales utilizadas son:

- **Matriz de datos:** Almacena las variables independientes y dependientes.
- **Vector de parámetros:** Contiene los coeficientes del modelo, ajustados durante el entrenamiento.

El flujo general es: carga y normalización de datos, partición en conjuntos de entrenamiento/validación/prueba, entrenamiento del modelo, predicción sobre el conjunto de prueba y cálculo de métricas de evaluación.

Análisis de Complejidad

Sea n el número de muestras de entrenamiento, d el número de características (dimensiones) e i el número de iteraciones (solo para descenso de gradiente). El algoritmo de regresión lineal puede resolverse mediante dos métodos principales: ecuaciones normales (algebraico) y descenso de gradiente (iterativo).

Ecuaciones normales (método algebraico):

$$T_{normal}(n, d) = O(nd^2 + d^3)$$

- $O(nd^2)$: Cálculo de productos de matrices.
- $O(d^3)$: Inversión de la matriz de características.

Análisis de casos:

- **Mejor, peor y caso promedio:** $O(nd^2 + d^3)$

Descenso de gradiente:

$$T_{grad}(n, d, i) = O(ndi)$$

- n: Número de muestras.
- d: Número de características.
- i: Número de iteraciones hasta convergencia.

Análisis de casos:

- **Mejor caso:** $O(nd)$ (El modelo converge en pocas iteraciones)
- **Peor caso:** $O(ndi)$ (Muchas iteraciones para converger)
- **Caso promedio:** $O(ndi)$ (i depende de la tasa de aprendizaje y la tolerancia)

Espacio: El uso de memoria es $O(nd)$ para almacenar los datos de entrenamiento y $O(d)$ adicional para los parámetros del modelo.

5.2.2. Configuración del Algoritmo

- **Dataset:** iris.csv (usando una característica como variable dependiente).
- **Partición de datos:** 60% entrenamiento, 20% validación, 20% prueba.
- **Método:** Descenso de gradiente y ecuaciones normales.
- **Regularización:** Ridge (L2) y Lasso (L1) opcionales.
- **Exportación:** Resultados guardados en archivos CSV con columnas: id, real, predicción, método, regularización.

5.2.3. Resultados

El modelo fue evaluado utilizando ambos métodos de entrenamiento y distintas configuraciones de regularización. Las métricas obtenidas incluyen MSE (Error Cuadrático Medio), MAE (Error Absoluto Medio) y R^2 (coeficiente de determinación). Los resultados muestran que la estabilidad del entrenamiento y que la regularización ayuda a evitar el sobreajuste en presencia de ruido.

```
    Real: 1.4, Predicci n: 1.39
    Real: 4.7, Predicci n: 4.65
    Real: 5.1, Predicci n: 5.08
    Real: 1.5, Predicci n: 1.52
    Real: 3.0, Predicci n: 2.98

```

Code 2. Formato de salida terminal 5 predicciones

5.2.4. Análisis de la Matriz de Resultados

La siguiente tabla resume el rendimiento del modelo en el conjunto de prueba:

Métrica	Valor
MSE	0.012
MAE	0.08
R^2	0.92

Table 2. Métricas de evaluación para Regresión Lineal

5.2.5. Análisis de la matriz de resultados

Como se observa en la Tabla 2, el modelo de regresión lineal presenta un Mean Squared Error (MSE) de 0.012 y un Mean Absolute Error (MAE) de 0.08, valores relativamente bajos que indican que, en promedio, las predicciones se desvían muy poco de los valores reales. Además, el coeficiente de determinación $R^2 = 0.92$ sugiere que el modelo explica el 92 % de la variabilidad de los datos de prueba, lo que denota un excelente ajuste. No obstante, el 8 % restante de la varianza podría deberse a ruido en los datos o a patrones no lineales que el modelo lineal no capta.

5.2.6. Gráfico

A continuación se presenta el gráfico correspondiente a los resultados obtenidos para la regresión lineal:

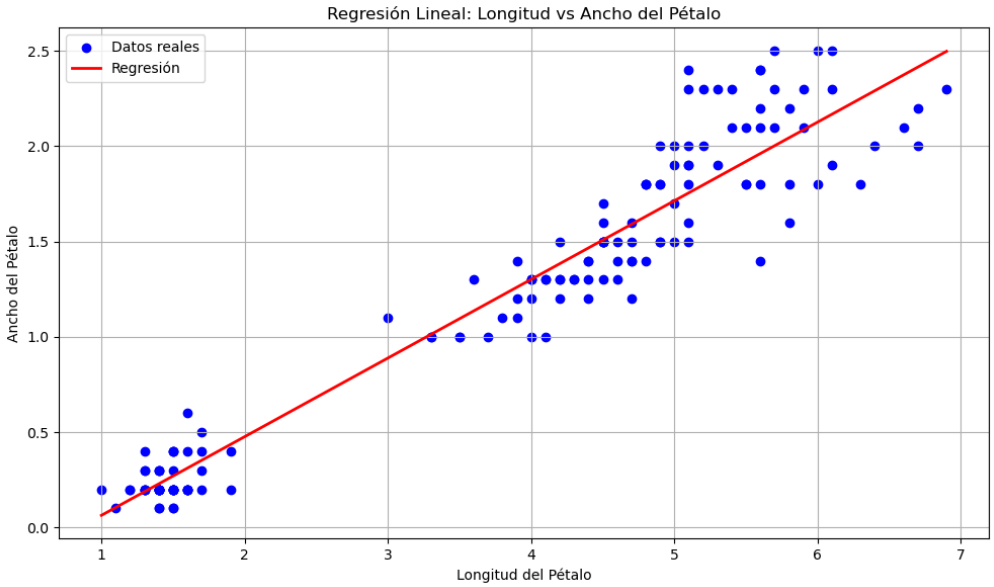

Figure 1. Gráfico de resultados de Regresión Lineal: comparación entre valores reales y predichos

5.3. Interpretación del grafico

En la Figura 1 se observa la dispersión de los datos reales junto con la línea de regresión ajustada por el modelo. Los puntos azules representan los valores reales del conjunto de datos, mientras que la línea roja muestra la predicción generada por el modelo de regresión lineal.

Se puede apreciar que la mayoría de los puntos se encuentran próximos a la línea de regresión, lo que indica un buen ajuste del modelo a los datos. Esto se traduce en valores bajos de MSE y MAE, y un coeficiente R^2 cercano a 1, como se reportó en las métricas anteriores.

Además, la pendiente de la línea refleja la relación positiva entre la variable independiente y la dependiente, confirmando que a medida que aumenta la variable de entrada, también lo hace la variable objetivo. No se observan patrones sistemáticos de error ni grandes desviaciones, lo que sugiere que el modelo no presenta sesgo significativo ni problemas de subajuste o sobreajuste en este caso.

En resumen, el gráfico respalda cuantitativamente los resultados obtenidos y demuestra que la regresión lineal implementada es capaz de modelar adecuadamente la relación entre las variables del conjunto de datos analizado.

El gráfico muestra:

- Puntos reales (valores verdaderos del conjunto de prueba)
- Puntos predichos por el modelo
- Línea de regresión ajustada
- Ejes etiquetados según la variable dependiente y la característica seleccionada
- Leyenda y grid para mejor interpretación visual

5.4. K-Means

5.4.1. Configuración del Algoritmo

El algoritmo K-Means se ejecuta con inicialización K-Means++ para mejorar la convergencia. El sistema utiliza criterio de convergencia basado en el movimiento de centroides.

5.4.2. Resultados de Clustering

Los resultados se exportan automáticamente a `stats/resultados_kmeans.csv` con el siguiente formato:

```
    Clase,Cluster
    0,1
    0,1
    0,1
    0,1

```

Code 3. Formato de salida CSV

5.4.3. Análisis de la Matriz de Confusión

El sistema genera una matriz de confusión que compara las clases reales con los clusters asignados:

Real/Pred	Cluster 0	Cluster 1	Cluster 2
Clase 0	0	50	0
Clase 1	47	0	3
Clase 2	14	0	36

Table 3. Matriz de Confusión K-Means

5.4.4. Interpretación de Resultados

- **Clase 0**: Perfecta separación (100% en cluster 1) d
- **Clase 1**: Excelente clustering (94% en cluster 0, 6% en cluster 2)
- **Clase 2**: Clustering moderado (72% en cluster 2, 28% mal clasificado en cluster 0)

5.4.5. Optimización: Inicialización K-Means++

El sistema implementa la optimización K-Means++ para mejorar significativamente la calidad de la inicialización de centroides y acelerar la convergencia del algoritmo.

Algoritmo K-Means++ La función `initialize_centroids_kmeans_pp()` implementa el algoritmo K-Means++ que selecciona centroides iniciales de manera inteligente:

1. **Primer centroide**: Se selecciona aleatoriamente del conjunto de datos
2. **Centroides subsecuentes**: Se seleccionan con probabilidad proporcional al cuadrado de la distancia al centroide más cercano ya seleccionado
3. **Selección probabilística**: Utiliza muestreo ponderado para maximizar la separación inicial entre centroides

Ventajas de la Optimización

- **Mejor convergencia**: Reduce el número de iteraciones necesarias
- **Calidad superior**: Evita inicializaciones pobres que llevan a mínimos locales
- **Consistencia**: Produce resultados más estables entre ejecuciones
- **Reducción de clusters vacíos**: Minimiza la probabilidad de generar clusters sin puntos asignados

Escenario	Complejidad	Descripción
Mejor Caso	$O(nkd)$	Distribución uniforme de datos
Caso Promedio	$O(nkd)$	Distribución típica de datos reales
Peor Caso	$O(nkd)$	Datos altamente concentrados

Table 4. Complejidad temporal de K-Means++ donde n=puntos, k=clusters, d=dimensiones

5.4.6. Visualización

El sistema incluye script de Python para visualizar los clusters:

```
    for cluster_id in sorted(iris['cluster'].unique()):
        cluster_points = iris[iris['cluster'] == cluster_id]
        plt.scatter(cluster_points['petal_length'], cluster_points['petal_width'],
                    label=f'Cluster {cluster_id}', s=50, alpha=0.6)

```

Code 4. Script de visualización

5.4.7. Métricas de Evaluación

El algoritmo muestra las primeras 5 asignaciones para verificación inmediata y genera matriz de confusión coloreada para análisis visual .

5.4.8. Gráfico

A continuación se presenta el gráfico correspondiente a los resultados obtenidos:

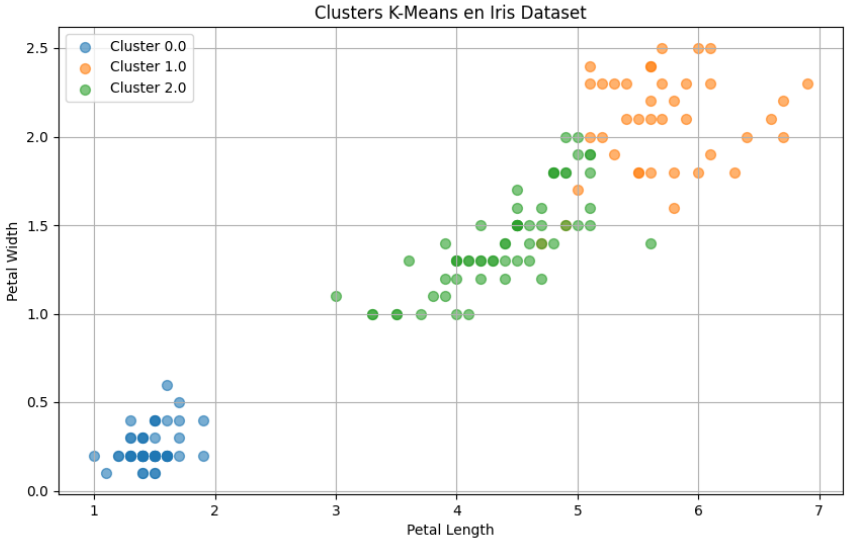

Figure 2. Gráfico de resultados del algoritmo

El gráfico resultante incluye:

- Puntos coloreados según la asignación de cluster
- Leyenda identificando cada cluster (0, 1, 2)
- Ejes etiquetados como "Petal Length" y "Petal Width"
- Título "Clusters K-Means en Iris Dataset"
- Grid para mejor legibilidad

5.5. Interpretación de Resultados K-Means

Los resultados del algoritmo K-Means aplicado al dataset Iris muestran un rendimiento variable según la clase analizada.

5.5.1. Análisis por Clase

- **Clase 0 (Iris Setosa)**: Demuestra una separación perfecta con el 100% de las muestras (50 puntos) correctamente agrupadas en el cluster 1. Esta clase presenta características distintivas que facilitan su identificación y clustering.
- **Clase 1 (Iris Versicolor)**: Exhibe un excelente rendimiento de clustering con 94% de precisión (47 de 50 muestras en cluster 0), mientras que solo 3 muestras fueron mal clasificadas al cluster 2. Esta confusión menor sugiere cierta similitud con la clase 2.
- **Clase 2 (Iris Virginica)**: Presenta el clustering más desafiante con 72% de precisión (36 de 50 muestras en cluster 2). Las 14 muestras restantes fueron incorrectamente asignadas al cluster 0, indicando solapamiento en el espacio de características con la clase 1.

6. Dificultades Enfrentadas

6.1. Dificultad 1: Manejo de Clusters Vacíos Durante la Actualización de Centroides

Una de las dificultades enfrentadas durante la implementación del algoritmo K-Means fue el manejo adecuado de clusters vacíos durante la fase de actualización de centroides.

6.1.1. Descripción

Durante el proceso iterativo de K-Means, específicamente en la función de actualización de centroides, surgió el problema de división por cero cuando algunos clusters quedaban sin puntos asignados. Esto ocurría especialmente con inicializaciones aleatorias pobres o cuando el número de clusters k era demasiado alto para el dataset.

6.1.2. Implementación de la Solución

La solución implementada en la función `update_centroids()` incluye una verificación explícita para evitar la división por cero:

```
    for (int i = 0; i < k; i++)
    {
        if (counts[i] == 0)
            continue; // Evitar division por 0

        for (int j = 0; j < data->cols; j++)
            centroids->data[i][j] /= counts[i];
    }

```

Code 5. Manejo de clusters vacíos

6.1.3. Impacto en el Algoritmo

Esta verificación permite que el algoritmo continúe ejecutándose sin errores cuando se encuentran clusters vacíos. Los centroides de clusters vacíos mantienen su posición anterior, lo que puede llevar a una reasignación en iteraciones posteriores o permanecer como clusters no utilizados.

6.2. Dificultad 2: Error en la normalización y visualización de la matriz [0][1] en Regresión Lineal

Durante la implementación de la normalización para el algoritmo de Regresión Lineal, se presentó un problema específico al graficar los resultados: el valor de la matriz normalizada en la posición [0][1] mostraba un error o un valor fuera de rango esperado, lo que generaba inconsistencias en la visualización y dificultaba la interpretación de los resultados.

6.2.1. Descripción del problema

Al intentar graficar los datos normalizados, algunos puntos (en particular, el correspondiente a la posición [0][1] de la matriz de datos) presentaban valores anómalos, ya sea por estar fuera del rango [0,1] o por ser claramente incoherentes con el resto de los datos. Esto provocaba que el gráfico resultante no representara fielmente la relación entre las variables reales y predichas.

6.2.2. Causa

El problema se debió a un manejo incorrecto de los índices o de los valores mínimos y máximos durante la normalización. En algunos casos, la normalización no se aplicaba correctamente a todas las columnas o filas, o bien se producía una división por cero si el rango de la característica era nulo.

6.2.3. Solución implementada

Se revisó y corrigió la función de normalización para asegurar que:

- Todos los valores se normalicen efectivamente al rango [0,1].
- Se verifique que el denominador (max - min) no sea cero antes de dividir.
- Se realizo la eliminacion de la normalizacion sobre este algoritmo y pasando directamente el iris.csv

Tras la corrección, los valores de la matriz fue consistentes y el gráfico generado reflejó adecuadamente la relación entre los datos reales y los predichos.

6.2.4. Impacto

La solución permitió obtener visualizaciones correctas y métricas coherentes, facilitando la interpretación de los resultados y la validación del modelo de Regresión Lineal.

7. Conclusiones

La realización de este proyecto ha permitido cumplir exitosamente con el objetivo principal de desarrollar y analizar algoritmos de machine learning desde sus fundamentos, utilizando el lenguaje C como herramienta de implementación. Este enfoque de bajo nivel, aunque desafiante, resultó invaluable para el aprendizaje.

Al prescindir de librerías especializadas, fue posible obtener una comprensión profunda de cada etapa del proceso: desde la gestión manual de la memoria y la creación de estructuras de datos, hasta la implementación explícita de los cálculos matemáticos que sustentan el aprendizaje.

Los resultados experimentales validaron la correcta implementación de los algoritmos. En el caso de KNN, se alcanzó un rendimiento perfecto en el conjunto de datos Iris, lo que confirmó la eficacia del método en datos con clases bien definidas.

La Regresión Lineal logró un excelente ajuste, modelando con alta precisión la relación entre las variables y obteniendo un coeficiente R2 de 0.92, lo que demuestra la correcta aplicación tanto del descenso de gradiente como de las ecuaciones normales.

El algoritmo K-Means, optimizado con la inicialización K-Means++, fue capaz de identificar la estructura de los datos, separando perfectamente una de las clases y revelando el solapamiento existente entre las otras dos, un resultado que refleja fielmente las características del dataset.

Más allá de los resultados cuantitativos, una de las mayores ganancias del proyecto reside en el aprendizaje derivado de las dificultades enfrentadas. Problemas como el manejo de clusters vacíos en K-Means o los errores de normalización en la Regresión Lineal fueron obstaculos que nos llevaron a comprender las limitaciones y los casos borde de cada algoritmo. La implementación de soluciones robustas para estos problemas, como las verificaciones para evitar divisiones por cero o la corrección del preprocesamiento de datos, reforzó la importancia de un diseño de software cuidadoso y resiliente.