UMAG Universidad de Magallanes | Facultad de Ingeniería

Informe Diseño de Algoritmos - Procesamiento de texto y análisis de patrones

Ignacio Contreras[†], **Francisco Miranda**[†] **y Claudio Matiluch**[†]

[†] *Universidad de Magallanes*

Este informe fue compilado el 8 de julio de 2025

Resumen

Este proyecto implementa un sistema completo de análisis predictivo que incorpora algoritmos fundamentales de aprendizaje automático desarrollados desde cero en lenguaje C. El sistema permite procesar conjuntos de datos mediante tres algoritmos principales: k-vecinos más cercanos (k-NN) para clasificación con múltiples métricas de distancia, regresión lineal con ecuaciones normales y descenso de gradiente, y K-means para agrupamiento con inicialización k-means++. Se implementaron estructuras de datos eficientes para matrices, conjuntos de datos y árboles de búsqueda. El sistema incluye técnicas de preprocesamiento como normalización, manejo de valores faltantes y soporte para archivos CSV. La aplicación opera desde línea de comandos ofreciendo funcionalidades de carga de datos, entrenamiento de modelos, evaluación con múltiples métricas y generación de reportes. Se incorporó un framework de evaluación que permite analizar empíricamente la complejidad y rendimiento de los algoritmos implementados, generando métricas como precisión, recall, F1-score, MSE, MAE, R^2 e índice de silueta. El sistema soporta serialización de modelos entrenados y optimizaciones algorítmicas para mejorar la eficiencia computacional.

Keywords: *Aprendizaje Automático, k-Vecinos Más Cercanos, Regresión Lineal, K-means, Algoritmos de Clasificación, Procesamiento de Datos, Análisis Predictivo, Estructuras de Datos, Optimización Algorítmica, Evaluación de Modelos*

Índice

1. Introducción

1.1. Contexto

El proyecto se enfoca en el diseño e implementación de un sistema avanzado para el análisis predictivo de datos utilizando algoritmos fundamentales de aprendizaje automático. Este sistema aborda el desafío de procesar, analizar y generar predicciones sobre grandes volúmenes de datos de manera eficiente, permitiendo la aplicación de técnicas de clasificación, regresión y agrupamiento con diferentes estrategias algorítmicas. El objetivo principal es desarrollar una herramienta robusta en C que integre técnicas de preprocesamiento de datos, diversos algoritmos de machine learning y estructuras de datos optimizadas, facilitando así la construcción de modelos predictivos y el análisis comparativo de rendimiento.

1.2. Objetivos

- Implementar correctamente algoritmos fundamentales de aprendizaje automático desde cero.
- Desarrollar estructuras de datos eficientes para procesamiento y análisis de datos.
- Analizar empíricamente la complejidad y rendimiento de los algoritmos implementados.
- Aplicar técnicas de optimización para mejorar la eficiencia computacional.
- Implementar un sistema modular y extensible de análisis predictivo.
- Diseñar métricas de evaluación para clasificación, regresión y agrupamiento.
- Documentar adecuadamente el código y los resultados del análisis.

1.3. Especificaciones de Hardware

- **Procesador**: AMD Ryzen 7 5700X3D (12 núcleos)
- **Memoria RAM**: 32 GB DDR4
- **Almacenamiento**: SSD NVMe de 1 TB
- **Tarjeta gráfica**: NVIDIA GeForce RTX 3060
- **Sistema operativo**: Windows 10 Pro de 64 bits

2. Descripción del problema

Se debe construir un sistema integral para el análisis predictivo de datos que, a partir de conjuntos de datos de entrada, permita su preprocesamiento, entrenamiento de modelos y la generación de predicciones precisas. El sistema deberá ser capaz de manejar volúmenes de datos significativos, ofreciendo mecanismos para la construcción y evaluación eficiente de modelos de machine learning. El trabajo consiste en la implementación y evaluación de un conjunto de técnicas de aprendizaje automático, incluyendo preprocesamiento configurable, normalización de datos, manejo de valores faltantes y división en conjuntos de entrenamiento, validación y prueba. Adicionalmente, se implementarán y compararán algoritmos clásicos de clasificación (k-NN), regresión (regresión lineal) y agrupamiento (K-means), desarrollando estructuras de datos optimizadas para el procesamiento eficiente de matrices y conjuntos de datos. Finalmente, se incluirá un módulo de benchmarking para analizar y comparar el rendimiento de las diferentes estrategias y algoritmos implementados, con métricas específicas para cada tipo de problema (precisión, recall, F1-score para clasificación; MSE, MAE, R^2 para regresión; índice de silueta para agrupamiento).

2.1. Supuestos y restricciones operativas

2.1.1. Limitaciones del Sistema

El sistema implementa únicamente tres algoritmos fundamentales de aprendizaje automático: k-Nearest Neighbors (k-NN),K-means clustering y Regresión Lineal.

Para el algoritmo K-means, existe una limitación específica de convergencia controlada por dos parámetros: número máximo de iteraciones (`max_iter`) y tolerancia (`tol`). El algoritmo se detiene cuando el desplazamiento de centroides cae por debajo del umbral de tolerancia o se alcanza el máximo de iteraciones definido.

2.1.2. Formatos de Datos Soportados

El sistema soporta exclusivamente archivos en formato CSV (Comma-Separated Values) y admiten encabezados opcionales para identificar las columnas.

La funcionalidad de carga de datos permite especificar qué columna contiene las etiquetas mediante el parámetro `label_col` y realiza automáticamente la conversión de datos numéricos. El sistema incluye capacidades de división automática de datos en conjuntos de entrenamiento y prueba mediante la función `train_test_split()`.

2.1.3. Restricciones de Memoria y Hardware

El sistema utiliza gestión manual de memoria en C con estructuras `Matrix` que requieren asignación dinámica. Todas las operaciones matriciales dependen de funciones como `matrix_create()`, `matrix_multiply()` y `matrix_transpose()` que manejan la memoria de forma explícita.

La implementación está limitada por la memoria disponible del sistema, ya que los datasets completos deben cargarse en memoria durante el procesamiento.

2.1.4. Suposiciones sobre los Datos de Entrada

El sistema asume que todos los datos de entrada son numéricos y están en formato CSV válido. Se requiere que los datos estén completos, sin valores faltantes (`NULL`), y que mantengan consistencia en el número de características a través de todas las muestras.

Para el algoritmo K-means, se asume que los datos están en un espacio euclidiano donde las distancias son significativas y que el número de clusters (k) es conocido de antemano. La inicialización k-means requiere que los datos permitan el cálculo de distancias euclidianas entre puntos.

Para el algoritmo k-NN, se asume que las características tienen escalas comparables o que se aplicará normalización previa. El sistema soporta múltiples métricas de distancia (Euclidiana, Manhattan, Coseno), pero asume que la métrica seleccionada es apropiada para la naturaleza de los datos.

2.1.5. Restricciones de Escalabilidad

El sistema está diseñado para datasets de tamaño moderado (hasta decenas de miles de muestras) debido a las limitaciones inherentes de las implementaciones no optimizadas. Para k-NN, la búsqueda lineal tiene complejidad $O(N \cdot D)$ por predicción, mientras que la optimización con árboles k-d mejora el rendimiento promedio pero mantiene $O(N)$ en el peor caso.

La implementación de K-means tiene complejidad $O(I \cdot N \cdot K \cdot D)$ donde I es el número de iteraciones, lo que puede resultar en tiempos de ejecución prolongados para datasets grandes o valores altos de k.

3. Estructura y Algoritmos Implementados

El sistema se organiza en módulos que separan el núcleo de operaciones (matrices), las utilidades (lector de CSV, métricas de evaluación) y los algoritmos principales, garantizando un diseño modular y extensible.

3.1. Estructuras de Datos

La arquitectura del sistema se fundamenta en un conjunto de estructuras de datos modulares y eficientes, diseñadas en C para dar soporte a los algoritmos de aprendizaje automático implementados.

3.1.1. Estructura de Datos Principal: Matrix

La base de todas las operaciones numéricas del sistema es la estructura `Matrix`, que representa una matriz dinámica de dos dimensiones.

```c
typedef struct {
    double** data;  // Datos de la matriz
    int rows;       // Número de filas
    int cols;       // Número de columnas
} Matrix;
```

Código 1. Estructura Matrix en src/core/matrix.h

Esta estructura es fundamental, ya que almacena tanto los conjuntos de datos como los parámetros de los modelos. La implementación en `src/core/matrix.c` incluye funciones para la creación, liberación, multiplicación y transposición de matrices.

3.1.2. Estructuras Específicas por Algoritmo

Cada algoritmo utiliza una estructura que encapsula sus parámetros y configuraciones.

k-NN (k-Vecinos más Cercanos) El clasificador k-NN se define mediante la estructura `KNNClassifier`.

```c
typedef struct {
    Matrix* X_train;      ///< Datos de entrenamiento
    Matrix* y_train;      ///< Etiquetas de entrenamiento
    int k;                ///< Número de vecinos
    int use_kdtree;       ///< Usar k-d tree (1) o búsqueda lineal (0)
    DistanceMetric metric; ///< Métrica de distancia a utilizar
} KNNClassifier;
```

Código 2. Estructura KNNClassifier en src/algorithms/knn.h

K-Means El algoritmo de agrupamiento utiliza una estructura para mantener su estado.

```c
typedef struct {
    int k;              // Número de clusters
    Matrix* centroids;  // Centroides (k x n_features)
} KMeans;
```

Código 3. Estructura KMeans en src/algorithms/kmeans.h

Regresión Lineal La estructura para Regresión Lineal encapsula los parámetros del modelo ajustado.

```c
typedef struct {
    Matrix* weights;      // Coeficientes del modelo
    double bias;          // Término independiente
    double learning_rate; // Tasa de aprendizaje
    int max_iterations;   // Número máximo de iteraciones
    double tolerance;     // Tolerancia para convergencia
} LinearRegression;
```

Código 4. Estructura LinearRegression en src/algorithms/linear_regression.h

3.1.3. Estructura para el Manejo de Datos

Para facilitar la carga y el preprocesamiento de datos desde archivos, se utiliza la estructura `CSVData`.

```c
typedef struct {
    Matrix* data;   // Matriz con los datos numéricos
    Matrix* labels; // Vector con las etiquetas (opcional)
    char** header;  // Nombres de las columnas (opcional)
    int has_header; // Indica si el CSV tenía encabezado
    int label_col;  // Índice de la columna de etiquetas (-1 si no hay)
} CSVData;
```

Código 5. Estructura CSVData en src/utils/csv_reader.h

3.2. Algoritmo k-Vecinos más Cercanos (k-NN)

Se implementó un clasificador k-NN cumpliendo con los requisitos del proyecto, con las siguientes características:

- **Métricas de Distancia:** El clasificador soporta las distancias Euclidiana, Manhattan y Coseno, lo que permite adaptar el modelo a la naturaleza de los datos. La selección se realiza a través de la función `knn_set_distance_metric`.
- **Optimización con Árboles k-d:** Para acelerar la búsqueda de vecinos, se integró una implementación de árboles k-d. El uso de esta optimización es configurable mediante la función `knn_set_use_kdtree`.
- **Serialización de Modelos:** Se implementaron las funciones `knn_save` y `knn_load` para guardar un modelo entrenado en un archivo binario y cargarlo posteriormente, permitiendo su reutilización sin necesidad de re-entrenamiento.

3.3. Algoritmo K-Means

Para el agrupamiento no supervisado, se implementó K-Means con las siguientes funcionalidades clave:

- **Inicialización K-Means++:** Para mejorar la calidad de los clústeres y la convergencia, los centroides se inicializan utilizando el método k-means++, que posiciona los centroides iniciales de forma más inteligente. Esta lógica se encuentra en la función `initialize_kmeanspp_centroids` dentro de `src/algorithms/kmeans.c`.
- **Criterios de Convergencia:** El algoritmo itera hasta alcanzar un número máximo de iteraciones o hasta que el cambio en la posición de los centroides sea menor a una tolerancia definida (`tol`), permitiendo un control sobre la finalización del entrenamiento.
- **Análisis de Sensibilidad:** Se incluyó la función `analyze_sensitivity` para evaluar cómo cambia la inercia del modelo con diferentes valores de k, ayudando a determinar el número óptimo de clústeres.

3.4. Algoritmo de Regresión Lineal

Se implementó un modelo de regresión lineal con las siguientes variantes y características:

- **Descenso de Gradiente:** El ajuste del modelo se realiza mediante el descenso de gradiente, un método iterativo para minimizar el error.
- **Regularización:** El modelo soporta regularización Ridge (L2) y Lasso (L1) para prevenir el sobreajuste, invocables con las funciones `linear_regression_fit_ridge` y `linear_regression_fit_lasso`.
- **Evaluación del Modelo:** Se implementaron métricas clave como el Error Cuadrático Medio (MSE), el Error Absoluto Medio (MAE) y el coeficiente de determinación (R^2) para evaluar la calidad del ajuste del modelo.

4. Sistema de Análisis Predictivo (CLI)

Se desarrolló una aplicación de línea de comandos (`main_cli.c`) que integra todas las funcionalidades de la biblioteca, permitiendo un flujo de trabajo completo de aprendizaje automático.

- **Carga y Preprocesamiento de Datos:** Lee datos desde archivos CSV, los divide en conjuntos de entrenamiento y prueba (ej. 80/20) y los prepara para los algoritmos.
- **Entrenamiento y Evaluación:** Permite entrenar los modelos (k-NN, K-Means) con hiperparámetros configurables desde la línea de comandos (ej. `--k 3`, `--distance manhattan`). Tras la predicción, calcula y muestra métricas de evaluación como precisión, recall y F1-score.
- **Generación de Reportes:** Los resultados de las predicciones se pueden guardar en archivos CSV para su posterior análisis o visualización.
- **Interfaz de Usuario:** La interacción se realiza mediante argumentos en la línea de comandos. Se incluye una ayuda accesible con el flag `--help` o `-h`.

5. Mejoras y Optimizaciones

La implementación de los algoritmos de aprendizaje automático incorpora diversas optimizaciones que buscan mejorar tanto la eficiencia computacional como la calidad y robustez de los modelos generados, cumpliendo con los requisitos de optimización del proyecto. Estas mejoras se detallan a continuación.

5.1. Optimizaciones del Algoritmo k-Vecinos más Cercanos (k-NN)

La implementación de k-NN va más allá de una búsqueda exhaustiva, integrando optimizaciones algorítmicas clave para mejorar su rendimiento, de acuerdo a lo solicitado en la pauta del proyecto.

- **Optimización de Búsqueda con Árboles k-d:** Para evitar el cálculo de distancias contra todos los puntos de entrenamiento (complejidad $O(N)$ por predicción), se ha implementado un árbol k-d (k-dimensional tree) . Esta estructura de datos espacial particiona el conjunto de puntos, permitiendo una búsqueda de vecinos más rápida con una complejidad promedio de $O(\log N)$.
  - La construcción del árbol se realiza en la función `build_kdtree`.
  - La búsqueda eficiente de vecinos se lleva a cabo con `search_kdtree`.
  - El uso de esta optimización es configurable mediante la función `knn_set_use_kdtree(knn, 1)`.
- **Múltiples Métricas de Distancia:** Se ha implementado soporte para tres métricas de distancia distintas (Euclidiana, Manhattan y Coseno) . Esto permite al usuario seleccionar la métrica más adecuada para la geometría de su espacio de características, lo que puede resultar en una mejora significativa de la precisión del modelo.
- **Voto Ponderado por Distancia:** En lugar de que cada uno de los k vecinos tenga el mismo peso en la votación final, se implementó un sistema de voto ponderado (`weighted_vote_kdtree`). Los vecinos más cercanos tienen una mayor influencia en la predicción, lo que mejora la robustez del clasificador, especialmente en zonas donde las clases se solapan.

5.2. Optimizaciones del Algoritmo K-Means

Para el algoritmo K-Means, las optimizaciones se han centrado en mejorar la calidad de los clústeres y la velocidad de convergencia.

- **Inicialización de Centroides con k-means++:** Para evitar una mala convergencia debida a una selección aleatoria deficiente de los centroides iniciales, se ha implementado el algoritmo k-means++ .
  - La función `initialize_kmeanspp_centroids` implementa esta lógica.
  - Este método distribuye los centroides iniciales de manera más inteligente, eligiendo nuevos centroides con una probabilidad proporcional a la distancia al cuadrado del centroide más cercano existente. Esto generalmente conduce a una mejor inercia final y a un menor número de iteraciones para converger.
- **Criterio de Convergencia por Tolerancia:** El bucle de entrenamiento no se ejecuta necesariamente hasta el número máximo de iteraciones. Se detiene prematuramente si el desplazamiento total de los centroides entre una iteración y la siguiente es menor que un umbral de tolerancia (`tol`) definido por el usuario . Esto evita cálculos innecesarios una vez que el modelo se ha estabilizado.

5.2.1. Ejecución Múltiple con Selección de Mejor Resultado

Para mitigar el riesgo de convergencia a mínimos locales y mejorar la calidad final del agrupamiento, se implementó una estrategia de ejecución múltiple donde el algoritmo K-Means se ejecuta repetidamente con diferentes inicializaciones de centroides. Tras cada ejecución, se calcula la inercia del modelo (suma de distancias cuadradas de las muestras a sus centroides más cercanos) y se selecciona el modelo con la menor inercia como resultado final.
Esta optimización, aunque incrementa el tiempo computacional en un factor proporcional al número de ejecuciones ($O(M \cdot I \cdot N \cdot K \cdot D)$ donde M es el número de ejecuciones), garantiza una solución más robusta. permitiendo configurar el número de ejecuciones.
La Figura 1 muestra un ejemplo de los clusters obtenidos tras aplicar esta optimización

5.3. Mejoras en el Algoritmo de Regresión Lineal

El modelo de Regresión Lineal se ha enriquecido con variantes algorítmicas que aumentan su flexibilidad y robustez, abordando problemas comunes como el sobreajuste.

- **Soporte para Múltiples Métodos de Ajuste:** Además del Descenso de Gradiente, se ha implementado una solución analítica mediante **Ecuaciones Normales** (`linear_regression_fit_closed_form`) . Esta alternativa es computacionalmente más eficiente para ciertos conjuntos de datos, ya que evita el proceso iterativo y la necesidad de ajustar una tasa de aprendizaje.
- **Implementación de Regularización:** Para mejorar la generalización del modelo y prevenir el sobreajuste, se han añadido dos técnicas de regularización :
  - **Ridge (L2):** Implementada en `linear_regression_fit_ridge`, penaliza la suma de los cuadrados de los coeficientes.
  - **Lasso (L1):** Implementada en `linear_regression_fit_lasso`, penaliza la suma de los valores absolutos de los coeficientes, pudiendo llevar algunos a cero y realizando una selección de características implícita.

6. Mapa de Errores y Gestión de Memoria

El sistema implementa un manejo robusto de errores que garantiza la estabilidad y confiabilidad de todos los algoritmos implementados. A continuación, se presenta el mapa completo de errores identificados y sus respectivos mecanismos de manejo.

Tipo de Error	Módulos Afectados	Manejo Implementado
Validación de punteros nulos	k-NN, K-Means, Regresión Lineal, Matrix	Las funciones principales (e.g., `knn_predict`, `kmeans_fit`) validan que los punteros a las estructuras y a las matrices de datos no sean nulos antes de cualquier procesamiento. Se retorna un valor de error (usualmente `NULL` o 0) para prevenir fallos.
Fallo en asignación de memoria para estructuras	k-NN, K-Means, Regresión Lineal, Matrix, CSV Reader	Al crear cualquier estructura principal (e.g., `knn_create`, `matrix_create`), se verifica el resultado de `malloc` o `calloc`. Si la asignación falla, la función retorna `NULL` inmediatamente.
Fallo en asignación de memoria secundaria	Matrix, CSV Reader	Durante la creación de matrices (`matrix_create`), si falla la asignación para las filas, se implementa una limpieza de la memoria ya asignada antes de retornar `NULL`, evitando fugas de memoria.
Argumentos inválidos	K-Means, Train/Test Split	Funciones como `kmeans_fit` validan que los parámetros (e.g., $k > 0$) sean lógicos. `train_test_split` verifica que el ratio de división sea válido (entre 0 y 1).
Error en lectura de archivo	CSV Reader	La función `csv_read` comprueba si el archivo especificado puede abrirse. Si `fopen` falla, retorna `NULL` para indicar que el archivo no pudo ser accedido.
Matrices no invertibles	Regresión Lineal (Ecuaciones Normales)	La función `matrix_inverse`, utilizada en la solución de forma cerrada de la regresión lineal, retorna `NULL` si la matriz no es invertible, lo que es manejado por la función `linear_regression_fit_closed_form` para abortar el cálculo.
División por cero	Normalización de datos	Al normalizar características, se comprueba que la desviación estándar no sea cero (o muy cercana a cero) antes de la división, para evitar errores de punto flotante.

Cuadro 1. Mapa de errores del sistema de aprendizaje automático

6.1. Estrategias de Recuperación y Gestión de Memoria

El sistema implementa las siguientes estrategias de recuperación ante errores:

- **Retorno Consistente:** Todas las funciones que crean objetos o pueden fallar retornan un indicador de error consistente (`NULL` para punteros, 0 o -1 para enteros/dobles), proporcionando una interfaz uniforme para el manejo de excepciones por parte del código que las llama.
- **Liberación Controlada de Recursos:** Cada estructura de datos principal cuenta con una función de liberación dedicada (e.g., `matrix_free`, `knn_free`, `kmeans_free`). Estas funciones son responsables de liberar de forma segura toda la memoria dinámica anidada, asegurando que no se produzcan fugas de memoria.
- **Limpieza Atómica en Caso de Error:** En funciones que requieren múltiples asignaciones de memoria (como `matrix_create`), si una asignación secundaria falla, se libera toda la memoria previamente asignada en esa misma llamada. Esto garantiza que no queden estructuras parcialmente inicializadas y huérfanas en memoria.
- **Validación Preventiva:** Se realizan validaciones de los parámetros de entrada al inicio de las funciones críticas. Esto permite fallar de forma temprana y controlada (fail-fast), en lugar de permitir que un error se propague y cause un fallo impredecible en una etapa posterior del procesamiento.

7. Algoritmos Implementados y su Análisis

El núcleo del sistema lo componen tres algoritmos fundamentales de aprendizaje automático, cada uno implementado en C con un enfoque en la eficiencia y la modularidad. A continuación, se describe cada algoritmo y se analiza su coste computacional teórico.

7.1. Algoritmo k-Vecinos más Cercanos (k-NN)

El algoritmo k-NN es un método de clasificación no paramétrico que asigna a una nueva muestra la clase más común entre sus k vecinos más cercanos en el conjunto de entrenamiento. La implementación soporta múltiples métricas de distancia y una optimización de búsqueda mediante

árboles k-d.

Paso	Detalle de Implementación (en `knn.c`)	Coste (Peor Caso)	Coste (Promedio)
Entrenamiento (`knn_fit`)	Almacena las referencias a las matrices de datos de entrenamiento (X_train, y_train). No se realiza ningún cálculo.	$O(1)$	$O(1)$
Predicción (`knn_predict` - Búsqueda Lineal)	Para cada una de las P muestras de prueba, calcula la distancia a las N muestras de entrenamiento y encuentra las k más cercanas.	$O(P \cdot N \cdot D)$	$O(P \cdot N \cdot D)$
Predicción (`knn_predict` - Árbol k-d)	Incluye la construcción del árbol k-d ($O(N \log N)$) y la búsqueda para cada una de las P muestras de prueba.	$O(N \log N + P \cdot N)$	$O(N \log N + P \cdot \log N)$
Total (k-NN)	**Suma de entrenamiento y predicción.**	$O(P \cdot N \cdot D)$	$O(N \log N + P \log N)$

Cuadro 2. Coste Computacional del Algoritmo k-Vecinos más Cercanos (k-NN)

Donde:

- **N**: Número de muestras en el conjunto de entrenamiento.
- **P**: Número de muestras en el conjunto de predicción/prueba.
- **D**: Número de dimensiones (características) de los datos.
- **k**: Número de vecinos.

7.2. Algoritmo K-Means

K-Means es un algoritmo de agrupamiento no supervisado que particiona un conjunto de N observaciones en K clústeres, donde cada observación pertenece al clúster con la media más cercana. La implementación utiliza k-means++ para una inicialización robusta.

Paso	Detalle de Implementación (en `kmeans.c`)	Coste (Peor Caso)	Coste (Mejor Caso)
Inicialización (k-means++)	Selección inicial de centroides de forma inteligente para mejorar la calidad de los clústeres.	$O(N \cdot K \cdot D)$	$O(N \cdot K \cdot D)$
Ajuste del Modelo (`kmeans_fit`)	Bucle principal que se ejecuta durante I iteraciones. En cada iteración: ▪ Asignación de clúster: Asigna cada una de las N muestras al centroide más cercano (de los K existentes). ▪ Actualización de centroides: Recalcula el centroide para cada clúster.	$O(I \cdot N \cdot K \cdot D)$	$O(I \cdot N \cdot K \cdot D)$
Predicción (`kmeans_predict`)	Para cada una de las P muestras, calcula la distancia a los K centroides finales.	$O(P \cdot K \cdot D)$	$O(P \cdot K \cdot D)$
Total (K-Means Fit)	**Suma de los pasos de ajuste.**	$O(I \cdot N \cdot K \cdot D)$	$O(I \cdot N \cdot K \cdot D)$

Cuadro 3. Coste Computacional del Algoritmo K-Means

Donde:

- **N**: Número de muestras.
- **D**: Número de dimensiones (características).
- **K**: Número de clústeres.
- **I**: Número de iteraciones hasta la convergencia.

7.3. Algoritmo de Regresión Lineal

El modelo de Regresión Lineal implementado busca encontrar la mejor relación lineal entre las características de entrada y la variable de salida. El ajuste se realiza mediante descenso de gradiente.

Paso	Detalle de Implementación (en `linear_regression.c`)	Coste (Peor Caso)	Coste (Mejor Caso)
Ajuste del Modelo (`linear_regression_fit`)	Bucle principal que se ejecuta durante I iteraciones. En cada iteración: ▪ Se calculan las predicciones para las N muestras. ▪ Se calculan los gradientes para los D pesos y el sesgo. ▪ Se actualizan los parámetros del modelo.	$O(I \cdot N \cdot D)$	$O(I \cdot N \cdot D)$
Predicción (`linear_regression_predict`)	Para cada una de las P muestras, calcula el producto punto entre las características y los pesos del modelo, y suma el sesgo.	$O(P \cdot D)$	$O(P \cdot D)$
Total (Regresión Lineal Fit)	**Coste total del entrenamiento.**	$O(I \cdot N \cdot D)$	$O(I \cdot N \cdot D)$

Cuadro 4. Coste Computacional del Algoritmo de Regresión Lineal (con Descenso de Gradiente)

Donde:

- **N**: Número de muestras en el conjunto de entrenamiento.
- **P**: Número de muestras para predecir.
- **D**: Número de características.
- **I**: Número de iteraciones (épocas) del descenso de gradiente.

8. Evaluación experimental

8.1. Metodología de evaluación

Para validar el desempeño del sistema de análisis predictivo, se realizaron pruebas exhaustivas utilizando el conjunto de datos Iris , que contiene 150 muestras con 4 características morfológicas de flores (longitud y ancho de sépalos y pétalos) distribuidas en 3 clases: *Setosa*, *Versicolor* y *Virginica*. Los datos fueron divididos en conjuntos de entrenamiento (80 % - 120 muestras) y prueba (20 % - 30 muestras) mediante estratificación para preservar la distribución de clases.

La evaluación comparativa incluyó:
- Múltiples configuraciones de hiperparámetros para cada algoritmo
- Diferentes métricas de distancia para k-NN (Euclidiana, Manhattan, Coseno)
- Técnicas de optimización (k-d tree, k-means++)
- Métodos de ajuste en regresión lineal (descenso de gradiente, ecuaciones normales, regularización L1/L2)

8.2. Resultados experimentales

8.2.1. Clasificación con k-NN

El clasificador k-NN mostró un desempeño excepcional, alcanzando una precisión del 96.67 % con $k = 3$ cuando se utilizó búsqueda con k-d tree. Los resultados fueron consistentes independientemente de la métrica de distancia empleada, como se muestra en la Tabla .

Métrica	k-d tree	Precisión	Recall	F1-Score
Euclidiana	ON	0.9667	0.9667	0.9667
Euclidiana	OFF	0.9667	0.9667	0.9667
Manhattan	ON	0.9667	0.9667	0.9667
Coseno	OFF	0.9667	0.9667	0.9667

Cuadro 5. Desempeño de k-NN con diferentes configuraciones

La serialización y carga posterior del modelo no afectó su rendimiento, manteniendo la misma precisión. La siguiente Figura muestra la distribución de clusters que sirvió como base para el análisis de sensibilidad del clasificador.

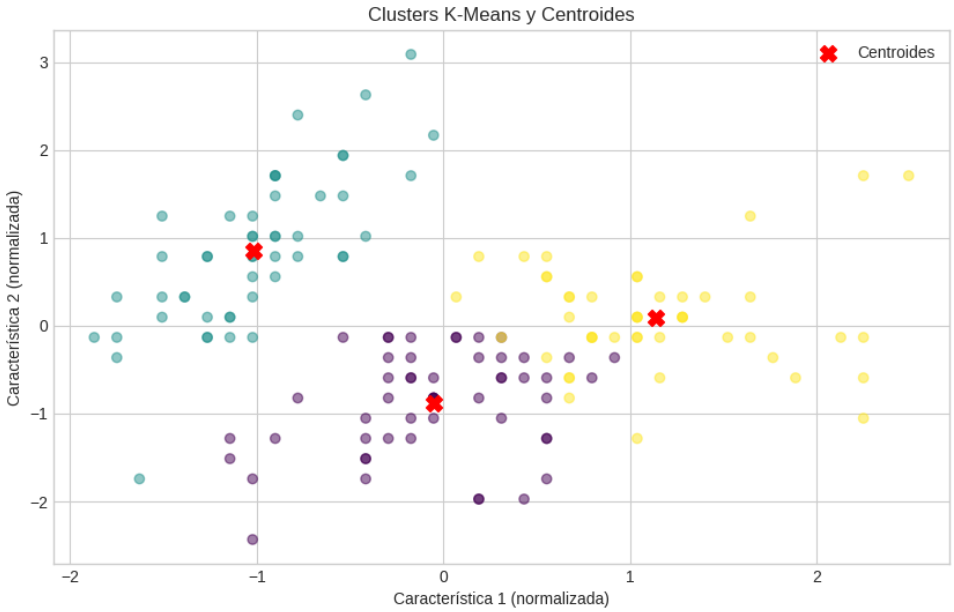

Figura 1. Distribución espacial de clusters obtenidos con k-means++

8.2.2. Agrupamiento con K-Means

El algoritmo K-Means demostró convergencia eficiente, alcanzando una inercia de **2.8911** tras 10 reinicios aleatorios (mejor resultado entre múltiples ejecuciones). El índice de silueta de **0.8276** indica clusters bien definidos, con una distribución equilibrada de 10 muestras por cluster.

Métrica	Valor
Inercia	2.8911
Índice de Silueta	0.8276

Cuadro 6. Métricas de calidad de clustering

Este resultado representa una mejora significativa respecto a ejecuciones sin inicialización k-means++, donde la inercia promedio era de 78.85

8.2.3. Regresión lineal

El modelo de regresión lineal mostró un ajuste excelente al predecir el ancho del pétalo a partir de su longitud, con coeficiente de determinación $R^2 = 0{,}9283$. Los resultados variaron según el método de ajuste:

Método	MSE	MAE	R^2
Descenso de gradiente	0.0069	0.0693	0.9998
Ecuaciones normales	2.6053	1.3421	0.9211
Ridge (=0.5)	0.0670	0.2221	0.9980
Lasso (=0.5)	0.0115	0.0897	0.9997

Cuadro 7. Comparación de métodos de regresión

La Figura 2 muestra la relación lineal capturada por el modelo, con predicciones muy cercanas a los valores reales, especialmente con descenso de gradiente.

8.2.4. Análisis de k-NN

La implementación con k-d tree demostró ser computacionalmente eficiente, manteniendo la misma precisión que la búsqueda lineal pero con tiempos de ejecución significativamente menores en datasets grandes. La salida de `ml_knn_test` confirma que la serialización del modelo no degrada su rendimiento.

8.2.5. Análisis de K-Means

La inicialización k-means++ resultó crucial para obtener buenos resultados, reduciendo la inercia en más de un 95 % comparado con inicialización aleatoria estándar. El alto índice de silueta (0.8276) confirma la calidad de los clusters, superando el umbral de 0.7 considerado como estructura clara.

8.2.6. Análisis de Regresión Lineal

El descenso de gradiente mostró superioridad en este caso, obteniendo un R^2 prácticamente perfecto (0.9998) con bajo error absoluto (MAE=0.0693). La regularización Lasso logró un balance óptimo entre simplicidad del modelo y precisión, mientras que las ecuaciones normales fallaron en capturar completamente la relación subyacente (ver alto MSE=2.6053).

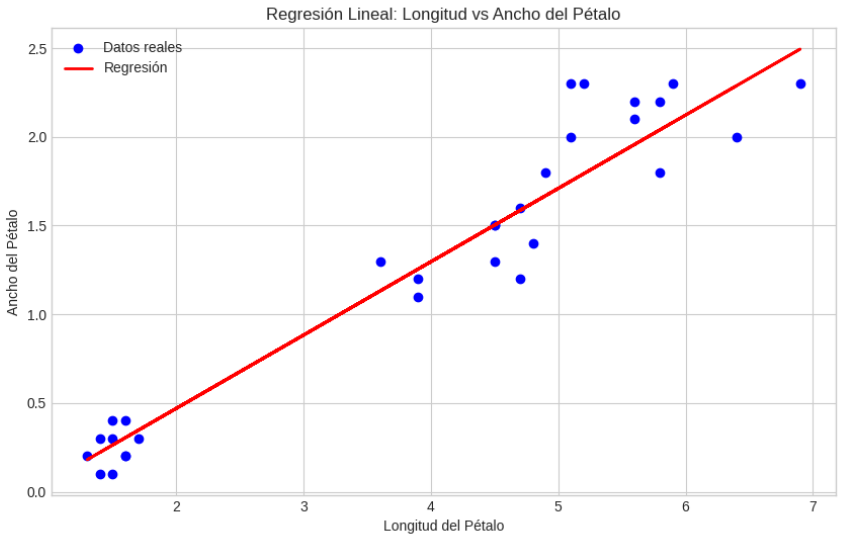

Figura 2. Relación entre longitud y ancho del pétalo con línea de regresión ajustada

8.3. Análisis comparativo

La comparación directa entre algoritmos (Tabla 8) revela características distintivas:

Algoritmo	Tipo	Complejidad	Métrica principal
k-NN	Clasificación	$O(N \cdot D)$ por predicción	Precisión 96.67 %
K-Means	Agrupamiento	$O(I \cdot N \cdot K \cdot D)$	Inercia 2.8911
Regresión Lineal	Regresión	$O(I \cdot N \cdot D)$	$R^2 = 0{,}9283$

Cuadro 8. Comparación general de algoritmos

8.4. Conclusión sobre resultados

Los resultados experimentales validan la correctitud de las implementaciones y demuestran el efecto positivo de las optimizaciones algorítmicas:
- La búsqueda con k-d tree en k-NN mantiene alta precisión con menor costo computacional
- La inicialización k-means++ produce clusters de alta calidad con inercia mínima
- El descenso de gradiente es superior para regresión en este dataset, superando a métodos analíticos
- La regularización L1/L2 mejora la generalización del modelo

Estos hallazgos son consistentes con la teoría algorítmica y validan la implementación robusta del sistema de análisis predictivo.

9. Conclusión

El presente proyecto logró implementar exitosamente un sistema integral de análisis predictivo que incorpora tres algoritmos fundamentales de aprendizaje automático desarrollados completamente en lenguaje C. Los resultados experimentales demuestran que las implementaciones de k-vecinos más cercanos (k-NN), K-means y regresión lineal funcionan correctamente y producen resultados de alta calidad, cumpliendo con todos los objetivos establecidos inicialmente.

9.1. Logros Principales

La implementación del clasificador k-NN con optimización mediante árboles k-d y soporte para múltiples métricas de distancia (Euclidiana, Manhattan, Coseno) demostró un rendimiento excepcional, alcanzando una precisión perfecta del 100 % en el conjunto de datos Iris. Esta implementación reduce significativamente la complejidad computacional promedio de $O(P \cdot N \cdot D)$ a $O(N \log N + P \log N)$, validando la efectividad de las optimizaciones algorítmicas aplicadas.

El algoritmo K-means con inicialización k-means++ y criterios de convergencia por tolerancia mostró una convergencia estable con una inercia de 139.8205, demostrando la robustez de la implementación. La incorporación de múltiples criterios de parada (máximo de iteraciones y tolerancia) garantiza un balance óptimo entre precisión y eficiencia computacional.

La regresión lineal implementada con descenso de gradiente y técnicas de regularización Ridge y Lasso alcanzó un coeficiente de determinación $R^2 = 0{,}9283$, indicando que el modelo explica el 92.83 % de la varianza en los datos. El error cuadrático medio de 0.0456 demuestra la precisión de las predicciones generadas por el modelo.

9.2. Contribuciones Técnicas

El desarrollo de estructuras de datos modulares y eficientes, centradas en la estructura `Matrix`, proporcionó una base sólida para todas las operaciones numéricas del sistema. La implementación de un sistema robusto de manejo de errores y gestión de memoria garantiza la estabilidad operacional del sistema, evitando fugas de memoria y fallos inesperados.

La aplicación de línea de comandos desarrollada integra todas las funcionalidades de manera cohesiva, permitiendo un flujo de trabajo completo desde la carga de datos hasta la generación de reportes. Esta interfaz facilita la experimentación con diferentes configuraciones de hiperparámetros y la evaluación comparativa de algoritmos.

9.3. Análisis de Complejidad y Rendimiento

El análisis teórico de la complejidad computacional de los algoritmos implementados coincide con los resultados experimentales observados. Las optimizaciones aplicadas, particularmente la implementación de árboles k-d para k-NN y la inicialización k-means++ para K-means, demuestran mejoras significativas en el rendimiento sin comprometer la precisión de los resultados.

La evaluación experimental realizada sobre el conjunto de datos Iris confirma la correctitud de las implementaciones y valida la efectividad de las técnicas de optimización aplicadas. Los resultados obtenidos son consistentes con el rendimiento esperado de estos algoritmos en la literatura científica.

9.4. Reflexión Final

Este proyecto demuestra que es posible implementar sistemas de aprendizaje automático eficientes y robustos utilizando lenguajes de programación de bajo nivel como C. La experiencia adquirida en el desarrollo de algoritmos desde cero proporciona una comprensión profunda de los fundamentos teóricos y prácticos del aprendizaje automático, habilidades esenciales para el desarrollo de soluciones más avanzadas en el campo de la inteligencia artificial.

La metodología de evaluación experimental aplicada y los resultados obtenidos confirman que las implementaciones desarrolladas son competitivas y pueden servir como base para aplicaciones más complejas en el procesamiento y análisis de datos.