



ugr | Universidad
de **Granada**

TRABAJO FIN DE GRADO
INGENIERÍA INFORMÁTICA

Estudio y Análisis de Metaheurísticas modernas para el problema de Selección de Características

Autor
Miguel García López

Directores
Daniel Molina Cabrera



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, Enero de 2024



Título del proyecto

Subtítulo del proyecto.

Autor

Nombre Apellido1 Apellido2 (alumno)

Directores

Nombre Apellido1 Apellido2 (tutor1)

Nombre Apellido1 Apellido2 (tutor2)

Estudio y Análisis de Metaheurísticas modernas para el problema de Selección de Características

MIGUEL GARCÍA LÓPEZ

Palabras clave: Metaheurística, selección de características, binario, optimización, población, fitness, aprendizaje automático

Resumen

En el ámbito del *Machine Learning*, algunos algoritmos, como KNN o SVM, muestran excelentes resultados, pero a menudo requieren un procesamiento previo para identificar las características más relevantes. Esto se debe a que, ya sea por la amplitud y complejidad del problema, por falta de conocimiento o contexto al recolectar los datos, es posible aglomerar información innecesaria, dando lugar a conjuntos de datos inmensos. Incluso los algoritmos que internamente realizan esta identificación pueden beneficiarse de un procesamiento adicional. Este preprocesamiento, conocido como selección de características (*feature selection*), se considera un problema complejo de optimización combinatoria.

Existen varios métodos comunes para la selección de características: filtrado, envoltura (Wrapper), métodos basados en árboles, Análisis de Componentes Principales (PCA) y selección L1 (Lasso Regression).

Estos métodos pueden utilizarse individualmente o en combinación para mejorar la selección de características y el rendimiento de los modelos de *Machine Learning*. En este documento se hablarán de métodos metaheurísticos para la resolución de este problema.

Las metaheurísticas son algoritmos diseñados para resolver problemas de optimización complejos cuando los recursos son limitados. Aunque inicialmente se desarrollaron para abordar principalmente problemas combinatorios, en la actualidad se están proponiendo y aplicando cada vez más en problemas que implican variables continuas o reales. En respuesta a la creciente demanda en el campo de la selección de características, se han adaptado versiones especializadas de estas metaheurísticas para abordar este tipo de problemas combinatorios. Sin embargo, a pesar del número creciente de propuestas en este campo, las comparaciones objetivas entre ellas son limitadas. Aunque existen revisiones bibliográficas, muchas de ellas carecen de comparaciones adecuadas debido a la importancia y actualidad del problema de selección de

características. Por lo tanto, hay una necesidad de estudios que proporcionen una evaluación comparativa más rigurosa y exhaustiva de las diferentes propuestas en este ámbito.

Existe por tanto una necesidad de estudios que proporcionen una evaluación comparativa más rigurosa y exhaustiva de las diferentes propuestas en este ámbito, planteando un doble estudio que use tanto adaptaciones binarias como reales con algoritmos modernos.

En este trabajo de carácter científico, se llevará a cabo una revisión bibliográfica de diversas metaheurísticas recientes para abordar el problema de selección de características. Se estudiarán e implementarán aquellas consideradas más prometedoras, con el objetivo de construir un repertorio amplio y variado de propuestas. Posteriormente, se realizará un estudio comparativo exhaustivo utilizando diversos algoritmos de machine learning y conjuntos de datos representativos. Finalmente, se llevará a cabo un análisis crítico utilizando diversas métricas y valoraciones, como la tasa de acierto y el tiempo de ejecución, entre otras.

Study and Analysis of Modern Metaheuristics for the Feature Selection Problem

MIGUEL GARCÍA LÓPEZ

Keywords: Metaheuristic, feature selection, binary, optimization, population, fitness, machine learning

Abstract

In the field of Machine Learning, some algorithms, such as KNN or SVM, show excellent results, but often require preprocessing to identify the most relevant features. This is because, whether due to the broad and complex nature of the problem, lack of knowledge, or lack of context when collecting data, it is possible to aggregate unnecessary information, resulting in immense datasets. Even algorithms that internally perform this identification can benefit from additional processing. This preprocessing, known as feature selection, is considered a complex combinatorial optimization problem.

There are several common methods for feature selection: filtering, wrapping (Wrapper), tree-based methods, Principal Component Analysis (PCA), and L1 selection (Lasso Regression).

These methods can be used individually or in combination to improve feature selection and the performance of *Machine Learning* models. This document will discuss metaheuristic methods for solving this problem.

Metaheuristics are algorithms designed to solve complex optimization problems when resources are limited. Initially developed mainly for combinatorial problems, they are increasingly proposed and applied to problems involving continuous or real variables. In response to the growing demand in feature selection, specialized versions of these metaheuristics have been adapted to tackle combinatorial problems. However, despite the increasing number of proposals in this field, objective comparisons between them are limited. Although there are literature reviews, many lack adequate comparisons due to the importance and timeliness of the feature selection problem. Therefore, there is a need for studies providing a more rigorous and exhaustive comparative evaluation of different proposals in this area.

Therefore, there is a need for studies that provide a more rigorous and comprehensive comparative evaluation of the different proposals in this field, proposing a dual study that uses both binary and real adaptations with

modern algorithms.

In this scientific work, a literature review of various recent metaheuristics for feature selection will be conducted. The most promising ones will be studied and implemented to build a broad and varied repertoire of proposals. Subsequently, a comprehensive comparative study will be conducted using various machine learning algorithms and representative datasets. Finally, a critical analysis will be carried out using various metrics and assessments, such as accuracy rate and execution time, among others.

Yo, **Miguel García López**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77159865E, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Miguel García López

Granada a 29 de Enero de 2024.

D. **Daniel Molina Cabrera**, Profesor del Departamento Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Estudio y Análisis de Metaheurísticas modernas para el problema de Selección de Características*, ha sido realizado bajo su supervisión por **Miguel García López**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 29 de Enero de 2024.

El director:

Daniel Molina Cabrera

Agradecimientos

Poner aquí agradecimientos...

Índice general

1. Introducción	11
1.1. Motivación	11
1.2. Objetivos	13
2. Planificación	15
3. Fundamentos Teóricos	21
3.1. Optimización	21
3.1.1. Definición general de un problema de optimización . .	21
3.1.2. Función objetivo y función fitness	22
3.1.3. Óptimos globales o locales	22
3.2. Selección de características	23
3.2.1. Necesidad y motivo	23
3.2.2. Concepto	25
3.2.3. Maldición de la dimensionalidad	26
3.3. Metaheurísticas	26
3.3.1. Exploración vs explotación	27
3.4. Teorema No Free Lunch	28
3.5. Aprendizaje automático	29
3.5.1. Aprendizaje supervisado	29
3.5.2. Aprendizaje no supervisado	30
3.5.3. SVM	30
3.5.4. K-nearest neighbors	31
4. Revisión de la literatura	33
4.1. Introducción a los primeros algoritmos poblacionales	35
4.2. Propuestas metaheurísticas modernas	37
4.3. Versiones binarias	40
5. Descripción de los algoritmos	43
5.1. Algoritmos genéticos	43
5.1.1. Introducción	43
5.1.2. Funcionamiento y Operadores	44
5.2. Algoritmos de colonias de hormigas	46

5.2.1. Introducción	46
5.2.2. Funcionamiento y Operadores	47
5.3. Optimización por enjambre de partículas	48
5.3.1. Introducción	48
5.3.2. Funcionamiento y Operadores	48
5.4. Evolución diferencial	49
5.4.1. Funcionamiento y Operadores	50
5.5. Optimización por colonia de abejas artificial	53
5.5.1. Introducción	53
5.5.2. Funcionamiento y Operadores	53
5.6. Algoritmo de optimización del saltamontes	55
5.6.1. Introducción	55
5.6.2. Funcionamiento y Operadores	55
5.7. Algoritmo de la libélula	57
5.7.1. Introducción	57
5.7.2. Funcionamiento y Operadores	57
5.8. Algoritmo de optimización de ballenas	59
5.8.1. Introducción	59
5.8.2. Funcionamiento y Operadores	60
5.9. Algoritmo del murciélagos	61
5.9.1. Introducción	61
5.9.2. Funcionamiento y Operadores	62
5.10. Optimización de los lobos grises	63
5.10.1. Introducción	63
5.10.2. Funcionamiento y Operadores	64
5.11. Algoritmo de la luciérnaga	66
5.11.1. Introducción	66
5.11.2. Funcionamiento y Operadores	66
5.12. Búsqueda Cuckoo	67
5.12.1. Introducción	67
5.12.2. Funcionamiento y Operadores	67
6. Diseño experimental	69
7. Análisis y resultados	75
7.1. Binario	76
7.1.1. General	76
7.1.2. Variabilidad	76
7.1.3. Fitness	79
7.1.4. Clasificación y reducción	83
7.1.5. Tiempo de ejecución	90
7.1.6. Convergencia	90
7.2. Continuo	92
7.2.1. General	92

ÍNDICE GENERAL **3**

7.2.2. Variabilidad	94
8. Bibliografía	97
A.	109

Índice de figuras

2.1. Diagrama de Gantt inicial	17
2.2. Diagrama de Gantt final	18
3.1. Puntos globales y locales	23
3.2. Correlación entre error del modelo y N	24
3.3. Tendencia de incremento de dimensionalidad en varios espacios de características	26
3.4. Clasificación metaheurísticas	27
3.5. Aprendizaje supervisado	29
3.6. Aprendizaje no supervisado	30
3.7. Margen en SVM	31
4.1. Popularidad de feature selection sobre los años	34
4.2. Popularidad de feature selection + metaheuristics sobre los años	34
4.3. PBT en Redes Neuronales	38
4.4. Popularidad metaheurísticas modernas	39
4.5. Funciones de transferencia	41
5.1. Funcionamiento de un algoritmo genético	44
5.2. One point crossover	45
5.3. Blend crossover	45
5.4. Mutación en DE	51
5.5. Crossover en DE	52
5.6. Comportamiento de ABCO	54
5.7. Convergencia en GOA	56
5.8. Vecindario de libélulas	58
5.9. Mecanismo de encogimiento del cerco	60
5.10. Mecanismo de actualización de posición en espiral	61
5.11. Rodeo de la presa en 2D y 3D	65
7.1. Rankings para fitness - binario	77
7.2. <i>Boxplots</i> para kNN - binario	78
7.3. Rankings para reducción de características	86
7.4. Reducción de características en bWOA y bGWO	89

7.5.	Convergencia de todas las metaheurísticas en waveform5000 - knn - binario	91
7.6.	Convergencia de todas las metaheurísticas en wdbc - knn - binario	91
7.7.	Rankings para fitness - real	93
7.8.	<i>Boxplots</i> para knn - real	95
A.1.	<i>Boxplot</i> waveform5000 - knn - real	109
A.2.	<i>Boxplot</i> yeast - knn - real	110
A.3.	<i>Boxplot</i> spectf-heart - knn - real	110
A.4.	<i>Boxplot</i> dermatology - knn - real	111
A.5.	<i>Boxplot</i> wdbc - knn - real	111
A.6.	<i>Boxplot</i> breast-cancer - knn - real	112
A.7.	<i>Boxplot</i> wine - knn - real	112
A.8.	<i>Boxplot</i> spambase-460 - knn - real	113
A.9.	<i>Boxplot</i> parkinsons - knn - real	113
A.10.	<i>Boxplot</i> sonar - knn - real	114
A.11.	<i>Boxplot</i> zoo - knn - real	114
A.12.	<i>Boxplot</i> diabetes - knn - real	115
A.13.	<i>Boxplot</i> iris - knn - real	115
A.14.	<i>Boxplot</i> ecoli - knn - real	116
A.15.	<i>Boxplot</i> waveform5000 - svc - real	116
A.16.	<i>Boxplot</i> yeast - svc - real	117
A.17.	<i>Boxplot</i> spectf-heart - svc - real	117
A.18.	<i>Boxplot</i> dermatology - svc - real	118
A.19.	<i>Boxplot</i> wdbc - svc - real	118
A.20.	<i>Boxplot</i> breast-cancer - svc - real	119
A.21.	<i>Boxplot</i> wine - svc - real	119
A.22.	<i>Boxplot</i> spambase-460 - svc - real	120
A.23.	<i>Boxplot</i> parkinsons - svc - real	120
A.24.	<i>Boxplot</i> sonar - svc - real	121
A.25.	<i>Boxplot</i> zoo - svc - real	121
A.26.	<i>Boxplot</i> diabetes - svc - real	122
A.27.	<i>Boxplot</i> iris - svc - real	122
A.28.	<i>Boxplot</i> ecoli - svc - real	123
A.29.	<i>Boxplot</i> waveform5000 - knn - binary	123
A.30.	<i>Boxplot</i> yeast - knn - binary	124
A.31.	<i>Boxplot</i> spectf-heart - knn - binary	124
A.32.	<i>Boxplot</i> dermatology - knn - binary	125
A.33.	<i>Boxplot</i> wdbc - knn - binary	125
A.34.	<i>Boxplot</i> breast-cancer - knn - binary	126
A.35.	<i>Boxplot</i> wine - knn - binary	126
A.36.	<i>Boxplot</i> spambase-460 - knn - binary	127
A.37.	<i>Boxplot</i> parkinsons - knn - binary	127

A.38. <i>Boxplot</i> sonar - knn - binary	128
A.39. <i>Boxplot</i> zoo - knn - binary	128
A.40. <i>Boxplot</i> diabetes - knn - binary	129
A.41. <i>Boxplot</i> iris - knn - binary	129
A.42. <i>Boxplot</i> ecoli - knn - binary	130
A.43. <i>Boxplot</i> waveform5000 - svc - binary	130
A.44. <i>Boxplot</i> yeast - svc - binary	131
A.45. <i>Boxplot</i> spectf-heart - svc - binary	131
A.46. <i>Boxplot</i> dermatology - svc - binary	132
A.47. <i>Boxplot</i> wdbc - svc - binary	132
A.48. <i>Boxplot</i> breast-cancer - svc - binary	133
A.49. <i>Boxplot</i> wine - svc - binary	133
A.50. <i>Boxplot</i> spambase-460 - svc - binary	134
A.51. <i>Boxplot</i> parkinsons - svc - binary	134
A.52. <i>Boxplot</i> sonar - svc - binary	135
A.53. <i>Boxplot</i> zoo - svc - binary	135
A.54. <i>Boxplot</i> diabetes - svc - binary	136
A.55. <i>Boxplot</i> iris - svc - binary	136
A.56. <i>Boxplot</i> ecoli - svc - binary	153
A.57.Convergencia con svc parte 1 - binario	154
A.58.Convergencia con svc parte 2 - binario	155
A.59.Convergencia con knn parte 1 - binario	156
A.60.Convergencia con knn parte 2 - binario	157

Índice de tablas

2.1. Tabla de duración de cada tarea	17
2.2. Costos servidor Google Cloud - Part 1	18
2.3. Costos servidor Google Cloud - Part 2	19
2.4. Costo estimado del proyecto	19
4.1. Algoritmos modernos	35
4.2. Algoritmos clásicos	35
6.1. Información de los conjuntos de datos ordenada por número de características	69
6.2. Parámetros de diferentes algoritmos de optimización	72
7.1. Algoritmos con más veces mejor resultados - knn y svc - binario	80
7.2. P-valores del bGWO vs el resto - knn - binario	80
7.3. P-valores del bGWO vs el resto - svc - binario	81
7.4. P-valores de ACO vs el resto - knn - binario	84
7.5. P-valores de ACO vs el resto - svc - binario	85
7.6. P-valores tras corrección entre bGWO y bWOA - knn	88
7.7. P-valores tras corrección entre bGWO y bWOA - svc	88
A.1. P-valores en <i>fitness</i> - knn - binario	137
A.2. P-valores en <i>fitness</i> - svc - binario	138
A.3. Ranking de los algoritmos en <i>fitness</i> - knn - binario	139
A.4. Ranking de los algoritmos en <i>fitness</i> - svc - binario	140
A.5. Media de <i>fitness</i> de los algoritmos - knn - binario	141
A.6. Media de <i>fitness</i> de los algoritmos - svc - binario	142
A.7. P-valores en <i>accuracy</i> - knn - binario	143
A.8. P-valores en <i>accuracy</i> - svc - binario	144
A.9. Ranking de los algoritmos en <i>accuracy</i> - knn - binario	145
A.10. Ranking de los algoritmos en <i>accuracy</i> - svc - binario	146
A.11. P-valores en ratio de selección - knn - binario	147
A.12. P-valores en ratio de selección - svc - binario	148
A.13. Ranking de los algoritmos en selección de características - knn - binario	149

A.14.Ranking de los algoritmos en seleccion de caracteristicas - svc - binario	150
A.15.Porcentajes de características seleccionadas y precisión en claseficación para cada algoritmo binario	151
A.16.Media de tiempos - knn - binario	152

Capítulo 1

Introducción

1.1. Motivación

El problema de la selección de características se define como el proceso de seleccionar un subconjunto de características relevantes. Una característica es una propiedad individual medible de un fenómeno concreto. Este problema es considerado un problema **NP duro** [1, 2]. La reducción de dimensionalidad, y con ello de características, suele ser necesario a la hora de crear un modelo predictivo por medio del aprendizaje automático ya que muchas de las características dentro de un conjunto de datos pueden no llegar a ser relevantes para solucionar aquellos problemas que se intentan solucionar, ya sea por que no aporta información, porque puede ser agrupada junto a otras tantas en una sola propiedad o incluso porque hay ruido en los datos, lo cual puede ser inevitable [3].

Gracias a la reducción de características es posible mejorar tanto la capacidad de generalización como la precisión del modelo predictivo gracias a la reducción de *ruido*.

Además de la simplificación del modelo, que conduce a una reducción del ruido, la selección de características es un preprocesamiento necesario por varias razones:

1. Interpretabilidad: La presencia de características irrelevantes puede complicar innecesariamente la interpretación y el rendimiento de los modelos de aprendizaje automático. La selección de un subconjunto relevante de características puede simplificar el modelo resultante, haciéndolo más comprensible y fácilmente interpretable.
2. Mejora de la eficiencia computacional: La reducción de la dimensionalidad

dad puede conducir a un ahorro significativo en términos de tiempo y recursos computacionales necesarios para el entrenamiento y la evaluación de modelos. Al eliminar características irrelevantes, se reduce la complejidad del problema y se acelera el proceso de aprendizaje. Cabe mencionar que el coste computacional de pre-procesamiento que constituye la selección de características es sí extenso, pero merece la pena de cara a posteriores usos en la construcción de modelos. Se hace una vez y se usa siempre.

3. Reduce la maldición de la dimensionalidad: Cuando la dimensionalidad se incrementa en un problema, el volumen del espacio también lo hace, y esto ocurre de manera exponencial. De forma que para obtener un resultado seguro/fiable, la cantidad de datos necesitados debe verse incrementada de manera también exponencial [4].

Este proyecto se centra en el uso de metaheurísticas para la resolución de este problema. Las metaheurísticas son algoritmos de optimización cuyo uso principal recae en tareas cuyo resultado es difícil de obtener por medios convencionales. Ciertos problemas pueden no tener una solución algorítmica obvia o simplemente ser demasiado complejos en cuanto al tiempo de resolución de los mismos. En ambos casos, las metaheurísticas son capaces de ofrecer soluciones muy buenas y en un tiempo admisible por medio de procesos de búsqueda inspirados en múltiples ámbitos (física, biología, comportamiento social, etc).

Para la selección de características existen multitud de métodos que tratan de aplacar este problema. Algunos de los más famosos son los método de filtrado, el análisis de componentes principales (PCA) o incluso distintos tipos de regresiones como *Lasso*. En este documento se estudian métodos de envoltura o *Wrapper*, los cuales hacen uso del entrenamiento y evaluación de modelos de *Machine Learning* para evaluar distintos conjuntos de características.

El reciente interés del problema de la selección de características en el ámbito de las metaheurísticas en los últimos años es más que evidente. Puede comprobarse como en los últimos años hay una tendencia en la publicación de artículos presentando nuevos métodos metaheurísticos, mejores con respecto a los clásicos o incluso comparativas y análisis entre distintos algoritmos.

Esta crecimiento viene acompañado, sin embargo, de comparaciones que distan de ser objetivas por varios motivos [5]. Entre varios artículos se comparan algoritmos del mismo tipo con soluciones y resultados muy variables entre sí a pesar de mismas configuraciones a la hora de experimentar, artículos sin código referenciado, de forma que sea más fácil interpretar los resultados o duplicarlos, y algoritmos novedosos presentados por su autor o autores que superaban al resto en alguna métrica concreta sin llegar a la rigurosidad

adecuada.

Por lo expuesto, la motivación principal de este trabajo es la de proveer información todo lo objetiva posible por medio de un análisis comparativo entre los algoritmos optimizatorios metaheurísticos más populares y más citados junto con los algoritmos más robustos y clásicos en el campo de la optimización pseudo estocástica. Se plantea una serie de estudios y comparaciones entre los algoritmos, haciendo uso de sus versiones en codificación binaria y su versiones en codificación continua o real. Se realizarán análisis sobre los resultados de forma que se obtengan respuestas a una serie de preguntas. ¿Son buenos en *Feature Selection* los algoritmos continuos que también lo son en binario?, ¿qué algoritmos modernos son más prometedores?, ¿los clásicos siguen siendo una opción viable frente a los modernos?

1.2. Objetivos

Objetivo General:

Realizar una comparación exhaustiva y objetiva de diversas metaheurísticas utilizadas en la selección de características, con el propósito de proporcionar una visión integral y evaluativa sobre su eficacia y aplicabilidad en diferentes contextos de análisis de datos.

Objetivos Específicos:

1. Evaluar el desempeño de las metaheurísticas más relevantes en el ámbito de la selección de características, analizando métricas clave como precisión, estabilidad de las soluciones y eficiencia computacional. Se emplearán conjuntos de datos de referencia y metodologías de validación cruzada para garantizar la robustez de los resultados.
2. Investigar la transferibilidad de las técnicas diseñadas para dominios continuos y binarios en el contexto de la selección de características. Se analizará si las metaheurísticas efectivas en un dominio son igualmente eficaces cuando se aplican a otro, identificando posibles ventajas y limitaciones de cada enfoque.
3. Identificar las fortalezas y debilidades de cada metaheurística según el tipo de representación de las características. Se realizará un análisis detallado del comportamiento de las técnicas en problemas de selección de características con diferentes tipos de datos, destacando su rendimiento relativo y sus áreas de aplicación más adecuadas.
4. Proporcionar recomendaciones prácticas basadas en los resultados obtenidos, con el objetivo de orientar a practicantes y académicos en

la selección y aplicación de metaheurísticas en problemas reales de selección de características.

5. Evaluar los resultados de las metaheurísticas en problemas de selección de característica usando distintos como algoritmos de aprendizaje los métodos *kNN* y *SVM*. Se realizará una comparativa a nivel de eficiencia en tiempo, estabilidad y calidad de los resultados.

Capítulo 2

Planificación

El proyecto se ha llevado a cabo durante un plazo de 8 meses, con una carga de trabajo adecuada a la larga duración del proyecto, siempre teniendo en cuenta el tiempo de referencia con respecto a los créditos para intentar encajar la planificación en ese intervalo temporal. Un trabajo de fin de grado consta de 12 créditos ECTS, donde se estima que cada crédito debe valer unas 25 horas de trabajo aproximadamente. Teniendo en cuenta estos datos, se calcula que la duración del TFG no debería ser superior a 300 horas, teóricamente hablando. En la práctica se han visto sobrepasadas, no por mucho, para poder realizar un proyecto de calidad. Ha de tenerse en cuenta también que el alumno trabaja 25 horas semanales y debe superar algunas asignaturas además de su proyecto final para terminar la carrera. Por lo tanto, el proyecto se planifica con una duración extendida en el tiempo, pero con una carga de trabajo semanal menos intensa.

Se planifica una duración de 8 meses aproximadamente, como ya se ha mencionado. Se utilizará un diagrama de Gantt [6] para describir la planificación del proyecto, de manera que se realizarán tareas en un orden cronológico, pero con superposición entre algunas. Varias tareas probablemente requerirán iteraciones posteriores, ya que es probable que se mejore y perfeccione el proyecto a lo largo de su ciclo de vida. Además hay otras tareas, como la investigación de metaheurísticas y la implementación de sus versiones binarias, que claramente se superponen, ya que tiene más sentido ir desarrollando cada una según se termina de estudiar para después comenzar con el siguiente algoritmo.

Las fases del ciclo de vida son:

- **Investigación inicial** Esto incluye investigar sobre conceptos básicos ya aprendidos, en forma de repaso sobre conceptos generales de aprendizaje automático, tipos de metaheurísticas, tipos de codificación, optimización de funciones, test estadísticos y conceptos básicos, código

Python y librerías asociadas, instalación de estas a partir de un entorno virtual, configuración del entorno de trabajo e investigación sobre el problema de selección de características.

- **Diseño del software:** Planificación de la estructura general del código, uso de patrones de diseño que puedan ser de utilidad de cara a al mantenimiento del software a lo largo del desarrollo, concepto de modularización inicial del código (estructura del proyecto), uso de entornos virtuales.
- **Investigación metaheurísticas:** Realización de un estudio más exhaustivo acerca de las metaheurísticas a implementar y sus diferentes versiones binarias. Esto incluye un listado de 12 metaheurísticas, cuya elección será justificada en posteriores secciones, siendo estas:
 - Firefly Algorithm
 - Whale Optimization Algorithm
 - Bat Swarm Optimizer
 - Grey Wolf Optimizer
 - Dragonfly Algorithm
 - Grasshopper Algorithm
 - Cuckoo Search
 - Differential Algorithm
 - Ant Colony Optimization
 - Artificial Bee Colony Optimization
 - Particle Swarm Optimization
 - Genetic Algorithm

De cada una de ellas se investigará su inspiración, funcionamiento, implementación y versiones binarias, normalmente asociadas al problema de selección de características.

- **Implementación del software:** Una vez claros los requisitos programáticos quedan establecidos, se implementará el software base. Esto incluye código en Python para la generación de gráficas, manejo de datasets en formato *arff*, codificación de los algoritmos metaheurísticos en versión binaria, implementación de función objetivo (*fitness*) y parametrización del programa para distintas pruebas.
- **Pruebas y refactorizado:** En esta etapa se llevarán a cabo pruebas exhaustivas para verificar la robustez y eficacia de los diferentes algoritmos implementados. Además, se considerará la refactorización del código si es necesario, con el fin de mejorar su estructura, claridad y mantenibilidad.

- **Análisis de resultados:** En esta fase se recopilarán datos de la ejecución de los algoritmos en sus diferentes versiones, así como entre ellos, utilizando los conjuntos de datos seleccionados para el proyecto. Esta recopilación de métricas permitirá una evaluación del rendimiento y la eficacia de cada algoritmo en comparación con los demás, así como su comportamiento en diferentes conjuntos de datos.
- **Documentación:** En esta etapa final se generará una documentación del proyecto que incluirá de forma general la descripción del problema, los objetivos, planificación, implementación, resultados y pruebas.

Tarea	Duración (horas)
Investigación inicial	20
Diseño del software	19
Investigación metaheurísticas	85
Implementación del software	76
Pruebas y refactorizado	25
Ánalisis de resultados	30
Documentación	91

Tabla 2.1: Tabla de duración de cada tarea

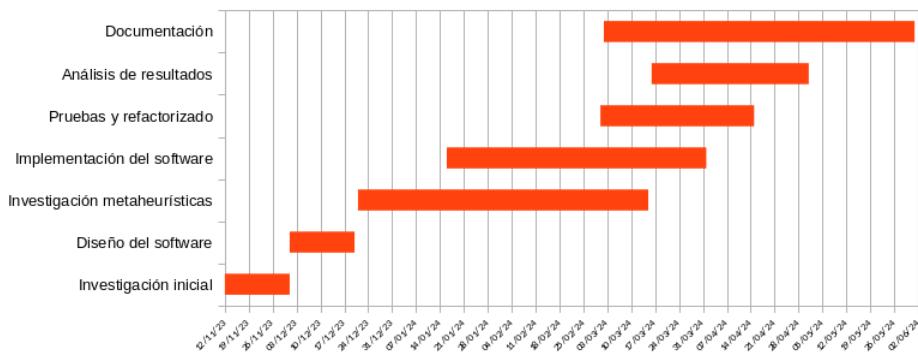


Figura 2.1: Diagrama de Gantt inicial

La planificación inicial tiene en cuenta un curso ideal del ciclo de vida del proyecto, siendo las etapas más extensas la de creación del software e implementación de la documentación.

La planificación final del proyecto se ha modificado significativamente debido a una serie de contratiempos y obstáculos surgidos durante su desarrollo, así como la influencia de numerosos eventos externos que han afectado a su cronograma. En particular, se han experimentado retrasos y bloqueos que han incidido en la duración prevista del proyecto. Esto ha llevado a una

re-evaluación de la estrategia de planificación original. Entre otras cosas, la implementación del software se ha visto alargada debido a numerosas correcciones.

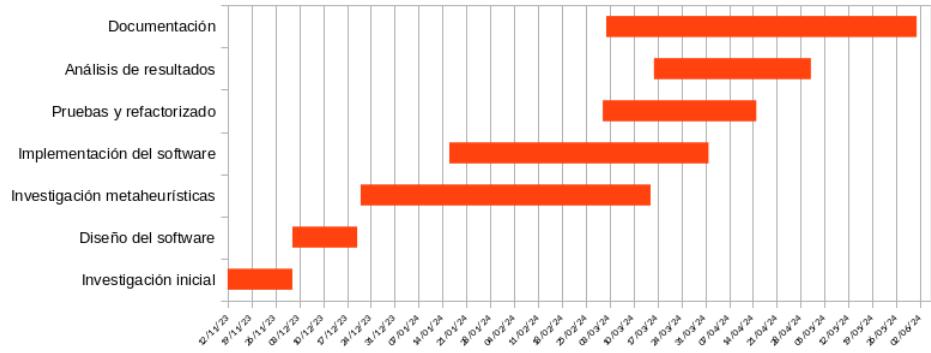


Figura 2.2: Diagrama de Gantt final

El coste estimado del proyecto se divide en varios sub-costes:

- **Sueldo:** Considerando un precio de proyecto que deba cubrir gastos y compensar, se estima un salario por investigador de 25€/hora. El proyecto ha durado aproximadamente 8 meses. Pese a ello, las horas trabajadas equivalen a un total de aproximadamente 400 horas. Por tanto, el salario final total es de 10.000€.
- **Ordenador portátil:** El estudiante consta con un portátil HP Pavilion Laptop 14-ec0xxx como herramienta principal de trabajo, con un precio de 1000€. Ha sido usado durante unos 8 meses aproximadamente. Si se tiene en cuenta que la vida media de un portátil es de aproximadamente 4 – 5 años [7] entonces el costo del ordenador sería de 133.3€.
- **Servidor** Para la experimentación del proyecto se ha hecho uso del servidor de investigación ofrecido por el tutor [8], por lo que el coste real ha sido de cero. Pese a ello, se estima cuento costaría un servicio parecido en un servidor *Cloud*. Se estima el precio por mes y total dada una estimación de los servicios de *Google Cloud - Compute Engine*.

Name	Quantity	Region	Service ID
Compute optimized Core running in London	150.0	europe-west2	6F81-5844-456A
Compute optimized Ram running in London	600.0	europe-west2	6F81-5844-456A
Storage PD Capacity in London	0.2	europe-west2	6F81-5844-456A

Tabla 2.2: Costos servidor Google Cloud - Part 1

En la primera tabla, se detallan tres servicios distintos implementados en la región de Londres. Cada servicio se identifica por su nombre, cantidad

SKU	Total Price (USD)
271A-7F2A-C5C5	6.56775
0BD6-E233-D705	3.5202
BF1A-6647-009D	0.0096
Total Price:	10.09755

Tabla 2.3: Costos servidor Google Cloud - Part 2

(indicando la cantidad de recursos utilizados), región de implementación y un identificador único del servicio. Lo importante es que se ha calculado un servicio con 30 vCPUs para paralelizar los trabajos.

La segunda tabla complementa la primera proporcionando detalles adicionales sobre los servicios en términos de SKU (Stock Keeping Unit, una identificación única para un producto) y el precio total en dólares estadounidenses asociado con cada uno de ellos. El total, hecho el cambio de divisa a euros, es de 9.48€ al mes, es decir, 75.84€ en total (calculado para 8 meses). Dado este desglose, se calcula el coste final del proyecto:

Item	Costo (€)
Salario	10.000
Ordenador portátil	133.3
Servidor CPU - GC Compute Engine	75.84
Total	10.209,14

Tabla 2.4: Costo estimado del proyecto

Capítulo 3

Fundamentos Teóricos

En este capítulo se describirán aquellos conceptos teóricos fundamentales para comprender el trabajo realizado en este proyecto.

3.1. Optimización

La optimización es un campo de estudio que trata, mediante el uso de las adecuadas herramientas matemáticas, de maximizar o minimizar una función objetivo. Esto significa, obtener la mejor solución posible para un problema dado dentro de un conjunto de alternativas y normalmente sujeto a una serie de restricciones que hacen de una solución satisfacible. Es un área interdisciplinaria que aborda desde campos tales como la Economía, Ingeniería, Biología y muchas otras tantas disciplinas.

La optimización es una disciplina arraigada en la naturaleza humana. Este impulso innato hacia la optimización ha llevado al desarrollo de diversas metodologías y técnicas a lo largo de la historia, desde los rudimentarios métodos de prueba y error hasta los sofisticados algoritmos de optimización computacional utilizados en la actualidad.

3.1.1. Definición general de un problema de optimización

Para poder convertir un problema abstracto en un problema de optimización concreto, con el que se pueda trabajar, es necesario establecer ciertos elementos fundamentales que lo definan de manera precisa y clara. En general, un problema de optimización se expresa de la siguiente forma [9]:

Función objetivo a minimizar $f(x)$ (3.1a)

Sujeto a

s restricciones de desigualdad $g_i(x) \leq 0, \quad j = 1, 2, \dots, s$ (3.1b)

w restricciones de igualdad $h_j(x) = 0, \quad j = 1, 2, \dots, w$ (3.1c)

Donde el número de variables es dado por $x_i, \quad i = 1, 2, \dots, n$

La definición general de un problema de optimización proporciona una estructura sólida para abordar el problema abstracto. Establece la función objetivo que se busca minimizar o maximizar, junto con las restricciones que deben cumplirse. Estas restricciones pueden ser tanto desigualdades como igualdades, y todas juntas definen el **espacio de búsqueda** del problema.

3.1.2. Función objetivo y función fitness

Ambos términos, aunque a menudo se utilizan como sinónimos, desempeñan roles distintos en el ámbito de la optimización. La función *objetivo*, como su nombre indica, establece el objetivo a alcanzar en la resolución del problema. Esta función cuantifica el rendimiento de las soluciones encontradas en relación con el objetivo específico del problema, que puede ser maximizar ganancias, minimizar distancia, entre otros objetivos. Por otro lado, la función de *fitness* solo tiene que evaluar la idoneidad de una solución dentro de una población de soluciones. Es decir, determina la calidad relativa de la solución respecto a otras alternativas.

Por otro lado, la función objetivo puede ser positiva o negativa, dependiendo de si se busca maximizar o minimizar el objetivo. Además, la función *fitness* también puede llegar a ser una aproximación de la función objetivo, pero no necesariamente coinciden exactamente.

Resumiendo, la función *fitness* es un tipo particular de función objetivo que se utiliza como métrica de rendimiento [10].

3.1.3. Óptimos globales o locales

Se conocen como punto de óptimo global (mínimo o máximo) la solución (o vector hablando en términos matemáticos) cuyo valor para la función objetivo es el más grande en todo el espacio de soluciones posible, es decir, el espacio de búsqueda. Los óptimos locales en cambio son varios, no solo uno como es el global. Son soluciones máximas o mínimas dentro de una región de soluciones.

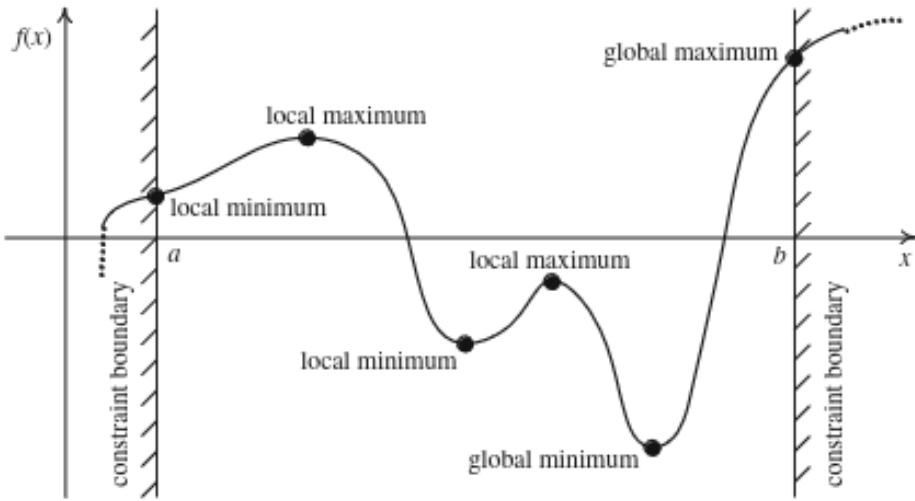


Figura 3.1: En esta figura extraída de [9] puede observarse de manera intuitiva la diferencia entre punto global y puntos locales. El máximo global es el valor más alto para $f(x)$ en el espacio, mientras que un máximo global solo representa el valor más alto dentro de un “vecindario”

Sea $f(x)$ una función a maximizar y x^* una solución óptima. Un objetivo $G(x)$ está en su máximo global sí y solo si [9]:

$$f(x^*) \geq f(x) \quad \forall x \quad (3.2)$$

En cambio el objetivo está en un máximo local en el punto x^* si:

$$\begin{aligned} f(x^*) &\geq f(x) \quad \forall x \\ &\text{dentro de un vecindario de } x^* \end{aligned} \quad [9] \quad (3.3)$$

3.2. Selección de características

La selección de características es un ejemplo de problema *NP-Hard* y uno de los problemas más importantes en el mundo de la inteligencia artificial, más concretamente en el **machine learning** o aprendizaje automático.

3.2.1. Necesidad y motivo

El aprendizaje automático se basa en los datos como fuente de aprendizaje. Es indispensable tener un buen conjunto de datos para poder obtener un modelo robusto y preciso. Por norma general o como se diría en inglés “as a

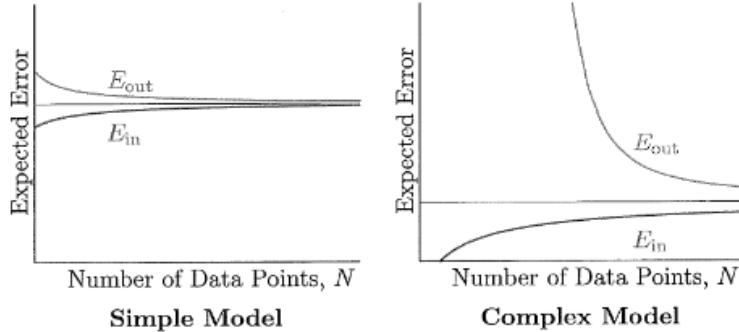


Figura 3.2: Esta figura extraída de [3] visualiza la relación directa entre calidad del modelo (menos error esperado) y número de datos que se le proporciona al algoritmo de aprendizaje.

rule of thumb”, a más grande el conjunto de datos, mayor calidad del modelo. De hecho hay una correlación inmediata con esta sentencia.

Es inmediato pensar que, a mayor cantidad de datos, mejor calidad del modelo, y así suele ser. Sin embargo, también existe una “normal general” que relaciona la complejidad del modelo y su capacidad de generalización. Es cierto que un modelo muy complejo (muchos parámetros) es capaz de ajustar mejor funciones más complejas, pero dentro de un conjunto con una serie de modelos suficientemente complejos, suele ser mejor idea elegir el más simple. Hay una serie clara de ventajas para ello:

1. Menor sensibilidad al sobre-ajuste.
2. Mayor interpretabilidad del modelo y sus resultados.
3. Mayor eficiencia y por tanto menor tiempo de ejecución y menor peso en memoria.

De hecho, como puede observarse en la figura 3.2, el modelo más complejo tiene un error esperado mucho mayor que el simple. El E_{out} o error fuera de la muestra (error de generalización) es mucho más abrupto, es decir, generaliza peor.

Por supuesto, con suficientes datos, con un N (número de datos) suficientemente grande, la tendencia del error es a la baja [3, 11]. Ocurre, sin embargo, que la recolección, limpieza y transformación de datos es una tarea compleja, por ello es mejor ceñirse, de nuevo, a el modelo más pequeño que obtenga una solución con suficiente calidad.

Como se ha mencionado anteriormente, la selección de características ayuda a la reducción del ruido. Siendo f la función objetivo a predecir, H^n el conjunto de hipótesis o conjunto de modelos de dimensión n posibles, $h^*(x)$ el mejor modelo aprendido y x una variable de entrada. El ruido conocido como ruido estocástico es aquel que atiende a una variación aleatoria que puede surgir de diversos factores, como mediciones imprecisas de señales o la falta de precisión en sensores. Por otro lado, el ruido determinista está directamente relacionado con la complejidad de un modelo. Su presencia aumenta la probabilidad de sobre-ajuste. El ruido determinista puede explicarse como la parte de la función f que el conjunto de hipótesis H^n no puede capturar, es decir, $f(x) - h^*(x)$. Este tipo de ruido se considera así porque la función (modelo) no es lo suficientemente compleja como para comprender esa parte. Este ruido depende de H^n y permanece constante para un valor dado de x [3].

La reducción de características ayuda a manejar ambos tipos de ruido [3, 12] al simplificar el modelo, lo que puede reducir el impacto del ruido estocástico y disminuir la complejidad del modelo, lo que a su vez puede ayudar a mitigar el ruido determinista al mejorar la capacidad del modelo para capturar las características relevantes y descartar las irrelevantes. Esto puede conducir a una mejor capacidad de generalización y a una reducción del sobre-ajuste.

3.2.2. Concepto

El funcionamiento y motivo es explicado ya en la sección de motivación en 1.1. Por evitar redundancias se procede a explicar el concepto de manera abreviada y puntuizando en los apartados más importantes.

La selección de características es una conocida y necesaria técnica de preprocesamiento de datos para la construcción de un modelo de aprendizaje. Su función es escoger un subconjunto óptimo de características dentro del conjunto inicial, de forma que la complejidad del modelo se reduzca.

Este problema es del tipo *NP-Hard*, como ya se ha mencionado previamente en 1.1 y explicado en ???. La selección de características ayuda a una serie de factores como son la interpretabilidad, mejora de la eficiencia temporal y espacial del modelo y sobre todo con la **maldición de la dimensionalidad** [13, 14].

3.2.3. Maldición de la dimensionalidad

El término fue acuñado por Bellman en 1961 [15]. Este fenómeno ocurre cuando la dimensionalidad de los datos es muy grande. En un espacio de características de alta dimensión, es común que los datos estén muy dispersos, lo que significa que hay muy pocos datos en comparación con la cantidad posible de características. Esto dificulta que los modelos representen correctamente todo el espacio de características [16].

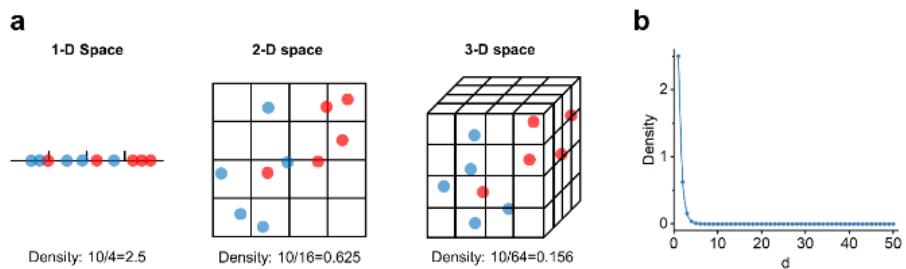


Figura 3.3: Esta figura extraída de [16] muestra la tendencia en la densidad a medida que se incrementa la dimensionalidad del espacio. En (a) se muestra la densidad con diez puntos de datos en un espacio 1D, en (b) se muestra la tendencia al incrementar las dimensiones.

Además, en espacios de alta dimensión, la medición de distancias puede volverse menos precisa. Esto se debe a que las distancias entre puntos de datos diferentes tienden a converger a un mismo valor a medida que aumenta la dimensionalidad. Esto significa que las medidas de distancia ya no son tan útiles para medir la similitud entre datos [13, 16]. En el trabajo de Beyer y colaboradores [17] se observó como esto ocurría y por tanto un escaneo lineal (recorrer los puntos uno a uno) resultaba en altas dimensiones más práctico que otras técnicas complejas.

3.3. Metaheurísticas

Dentro de la optimización hay muchos tipos de métodos, dentro de los pseudoaleatorios pueden encontrarse las metaheurísticas. Estas son algoritmos basados en una abstracción de mayor nivel de la **heurísticas**. Mientras que las heurísticas se apoyan en el conocimiento específico del campo en el que se encuentra el problema, y están restringidas a su dominio, las metaheurísticas son aplicables a todo tipo de problemas, independientemente de su área de optimización [18]. Es cierto que hay algoritmos que son más convenientes para ciertos problemas que otros, pero su aplicación es generalizada. Normalmente, las metaheurísticas son diseñadas a siguiendo una inspiración en la naturaleza,

ya sea en fenómenos físicos o en el comportamiento animal. Ejemplo de estos son algoritmos como el *Búsqueda Cuckoo*, *Enfriamiento Simulado* o incluso *Algoritmos Genéticos*.

Las metaheurísticas son especialmente útiles en problemas cuya resolución no es factible debido a altos costos computacionales, ya sea porque es posible analíticamente pero computacionalmente costoso, o porque el problema no es abordable mediante algoritmos convencionales. Son capaces de encontrar óptimos locales lo suficientemente aceptables, soluciones no óptimas, pero si muy buenas [18].

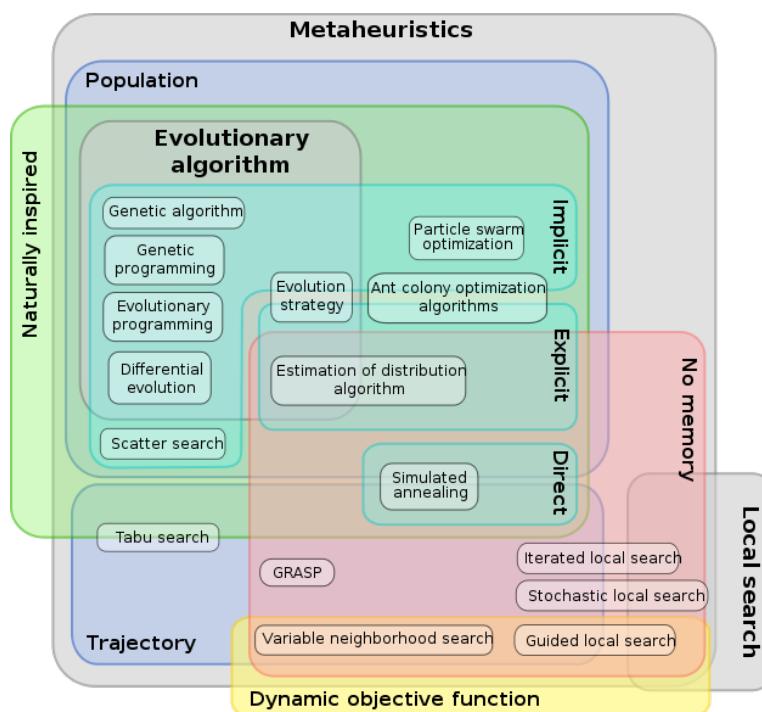


Figura 3.4: En esta figura de los autores Johann “nojhan” Dréo y Caner Candan, se clasifican las distintas metaheurísticas por inspiración.

3.3.1. Exploración vs explotación

Las metaheurísticas, y algoritmos basados en técnicas pseudoaleatorias, deben tener un balance entre sus factores explorativos y explotativos. De no ser así, los algoritmos tendrían tendencias a la convergencia temprana, dejando de lado mucho espacio por explorar y por ende soluciones posiblemente mejores (estancamiento en óptimos locales) o serían muy lentos en la convergencia hacia una solución.

La **exploración** se refiere a la habilidad del algoritmo de buscar nuevas y diversas regiones en el espacio de búsqueda/solución. Es una característica de la búsqueda global, que también puede ser llamada *diversificación*. En cambio, la **explotación** es la habilidad de la búsqueda de explotar las mejores soluciones encontradas hasta el momento y mejorárlas localmente, dentro de un “vecindario” [19].

No existe un equilibrio *de facto* entre exploración y explotación en las metaheurísticas. Aún no se ha alcanzado una respuesta definitiva a esta cuestión. Desde una perspectiva de sistemas, una metaheurística puede entenderse como un sistema dinámico compuesto por numerosos “individuos” que interactúan entre sí. Estos individuos representan las distintas soluciones o posiciones en el espacio de búsqueda que la metaheurística explora [20]. Las interacciones entre estos individuos son las que conforman comportamientos explorativos o explotativos y dependen del problema a solucionar y del propio algoritmo su equilibrio.

3.4. Teorema No Free Lunch

El Teorema de “No Free Lunch” (NFL) establece que, en promedio, ningún algoritmo de búsqueda puede superar a otros algoritmos en la búsqueda de todas las funciones objetivo posibles. En otras palabras, no existe un algoritmo universalmente óptimo que pueda dominar en todos los problemas de búsqueda. Esto implica que, si un algoritmo es efectivo para un conjunto particular de problemas, es probable que no lo sea para otros [21].

Relacionar este teorema con las metaheurísticas implica reconocer que no hay una única metaheurística que sea la mejor para todos los problemas de optimización. Cada problema puede tener características únicas que lo hacen más o menos adecuado para ciertas metaheurísticas. Por lo tanto, en lugar de buscar una solución universal, las metaheurísticas se centran en explorar y explotar diferentes áreas del espacio de búsqueda para encontrar soluciones aceptables o incluso óptimas para problemas específicos.

Pese a ello, las suposiciones de este tipo de teoremas, como conjuntos de datos extraídos de una distribución uniforme sobre todos los conjuntos de datos posibles, están completamente desalineadas con el mundo real, donde los datos suelen ser altamente estructurados y no están uniformemente muestreados [22, 23].

De esta forma y faltando evidencia concluyente al respecto, es interesante mencionar el **NFL**, pero sin llegar a negar la posibilidad de nuevas y más precisas interpretaciones.

3.5. Aprendizaje automático

El aprendizaje automático es una sub-rama de estudio de la inteligencia artificial o **IA**, la cual aglomera una serie de métodos que pueden, de manera automática, detectar patrones en conjuntos masivos de datos para predecir datos futuros [24]. El aprendizaje automático o *machine learning* en inglés, es usado en una amplia variedad de campos por su utilidad trasversal. Algunos de ellos son la agricultura, marketing, videojuegos, meteorología, física, etc.

Los algoritmos de aprendizaje automático son capaces de encontrar patrones en los datos, como ya se ha mencionado. Por ello, es necesario nutrir a estos algoritmos con datos de calidad. Un modelo de aprendizaje automático será tan bueno como los datos que se le puedan proveer, no más. De esta forma la recolección de datos y su procesamiento se convierten en una prioridad a la hora de crear modelos.

Hay muchas formas de estructurar los datos y muchos tipos de algoritmos acorde a estos “inputs”. En este documento se tratará con información en formato tabular, es decir, información en tablas con filas y columnas, donde cada fila representa un registro de información y cada columna una característica asociada. Esta última es la que determinará la complejidad del modelo.

3.5.1. Aprendizaje supervisado

Este subtipo de aprendizaje automático se caracteriza por tener un conjunto de datos sobre los que se entrena y una salida esperada (etiquetas) para cada punto de los datos [25]. Es bastante costoso porque etiquetar cada dato es una tarea laboriosa y poco escalable.

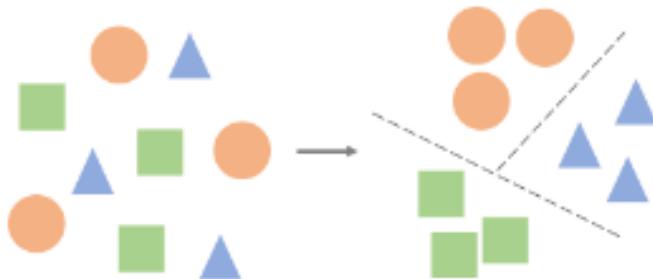


Figura 3.5: Figura extraída de [25] que muestra un ejemplo de aprendizaje supervisado.

3.5.2. Aprendizaje no supervisado

Cuando los datos solo vienen en forma de entrada y no se tiene ninguna salida (no hay etiquetas) entonces se trata de un aprendizaje no supervisado. Los algoritmos de este tipo se basan en la diferenciación de los datos mediante los patrones subyacentes que puedan encontrar. Son comunes en el aprendizaje no supervisado los algoritmos de *clustering* [25].

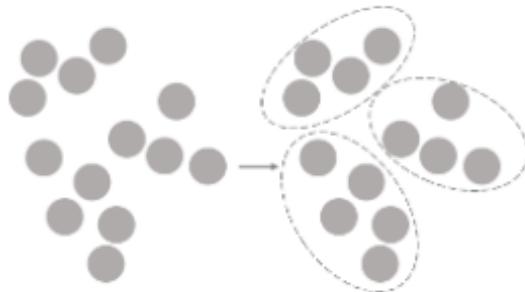


Figura 3.6: Figura extraída de [25] que muestra un ejemplo de aprendizaje no supervisado. Se trata de dividir en clústers sin información de etiquetas.

3.5.3. SVM

Las máquinas de vectores de soportes o *SVM* fueron introducidas por primera vez en [26]. En este documento será uno de los algoritmos de clasificación usados en la función *fitness* para cuantificar la calidad de los pesos aprendidos por los algoritmos optimizadores.

Las máquinas de vectores de soporte son algoritmos cuya principal característica se basa en la creación de un hiperplano o conjunto de hiperplanos en un espacio n -dimensional [27]. Este hiperplano o hiperplanos es elegido de manera que maximice el margen entre los puntos de datos de todas las clases a predecir. El margen es la distancia entre el hiperplano y los puntos de datos más cercanos de cada clase, estos puntos se denominan vectores de soporte. De esta manera se consigue minimizar el error de generalización [28].

La idea más intuitiva es, que al haber más espacio entre los puntos de distintas clases, hay más posibilidad de que los puntos no vistos, los puntos a predecir, caigan en zonas correctas.

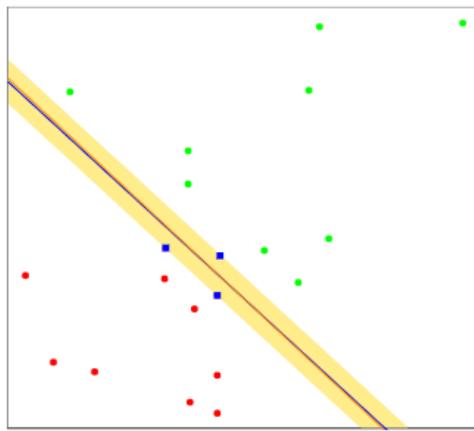


Figura 3.7: Figura extraída de [28] en la que se muestra de forma gráfica en un espacio 2-dimensional el margen óptimo de separación de dos clases.

3.5.4. K-nearest neighbors

El algoritmo de k vecinos más cercanos fue presentado en [29] por Evelyn Fix y Joseph Hodges, y más tarde ampliado en [30] por Thomas Cover.

Puede ser utilizado para regresión o clasificación, pero su uso suele darse más en este último problema. Este algoritmo ofrece una premisa sencilla, la clase de un punto p dependerá de cuál sea la clase mayoritaria dentro del subconjunto de k vecinos más cercanos [31].

El funcionamiento del algoritmo se puede resumir en los siguientes pasos:

1. **Selección del valor de k :** El usuario debe determinar el número de vecinos a considerar. Este número puede variar dependiendo del problema y los datos específicos.
2. **Cálculo de distancias:** Para un punto nuevo, se calculan las distancias a todos los puntos en el conjunto de datos de entrenamiento. Las distancias comúnmente se calculan utilizando métricas como la distancia euclídea, aunque otras métricas (como la distancia de Manhattan o la distancia de Minkowski) pueden ser utilizadas dependiendo del contexto y las características de los datos.
3. **Identificación de los k vecinos más cercanos:** Una vez calculadas las distancias, se seleccionan los k puntos del conjunto de datos de entrenamiento que estén más cerca del punto nuevo.
4. **Asignación de clase (para clasificación):** La clase del nuevo punto se determina por la mayoría de votos entre los k vecinos seleccionados.

En otras palabras, se asigna la clase más común entre estos vecinos.

5. **Predicción de valor (para regresión):** En el caso de la regresión, el valor predicho para el nuevo punto es el promedio (o alguna otra función agregada) de los valores de los k vecinos seleccionados.

Valores mayores de k incrementan la varianza, pero decrementan el sesgo (equilibrio sesgo-varianza [3]). De forma inversa, a menor k mayor sesgo y menor varianza.

Otra forma de verlo es que si $k = 1$ (el valor mínimo), el modelo entrenado será muy complejo, pues cada punto puede variar de clase por mínimo que sea el cambia en su entorno. Es un modelo que tiende a sobre-ajustar. Con $k = n$ el modelo es lo más simple posible, al tener en cuenta absolutamente todos los puntos, la clasificación será siempre la de la clase más repetida.

Capítulo 4

Revisión de la literatura

La selección de características es un problema cuya popularidad ha ido en aumento con el paso de los años. No es algo casual, pues la cantidad de datos recogidos para tareas de aprendizaje automático y áreas derivadas, como el aprendizaje profundo, ha ido en aumento de manera casi exponencial.

En este trabajo, se lleva a cabo una investigación y análisis comparativo entre varios métodos de la familia **wrapper** o métodos de envoltura. Existen multitud de estrategias [12] que intentan dar solución a este problema. Los métodos de búsqueda más famosos son los de filtrado (**filter**), los cuáles seleccionan las características más discriminativas según la naturaleza de los datos [12]. Por lo general, estos métodos realizan la selección de características antes de las tareas de clasificación y agrupamiento. Ejemplos de algoritmos de filtrado son *reliefF* [32] o F-statistic [33].

Los métodos **wrapper**, en cambio, utilizan el algoritmo de aprendizaje usado post-procesamiento para evaluar las características y seleccionar así las más útiles [12].

Los algoritmos clasificatorios de aprendizaje utilizados en este trabajo son *SVM* [26] y *kNN* [29, 30], siendo las máquinas de vectores de soporte un método robusto y eficiente y los vecinos más cercanos un método simple, interpretable y muy eficaz. Se analizará el resultado entre ambos clasificadores entre otros muchos análisis comparativos.

Se puede observar en 4.1 la creciente popularidad de los métodos de selección de características en el buscador de **Scopus**. La tendencia parece indicar que la tendencia científica puede incluso crecer aún más alrededor del problema de selección de características. Y es que pese a los grandes avances en *hardware*, parte del techo con el que se topan los algoritmos de aprendizaje sigue siendo los enormes tiempos de entrenamiento. También ha de mencionarse la tendencia, prácticamente igual en cuanto a escala de crecimiento, en la popularidad sobre los años del problema de selección de características, pero

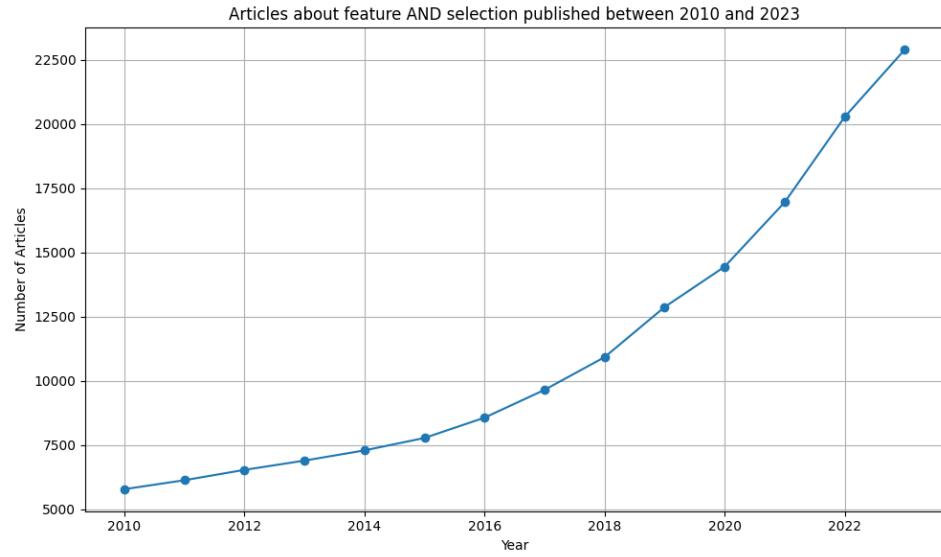


Figura 4.1: En esta figura se muestra el número de artículos publicados relacionados con la selección de características. Se ha usado el buscador Scopus para estos resultados.

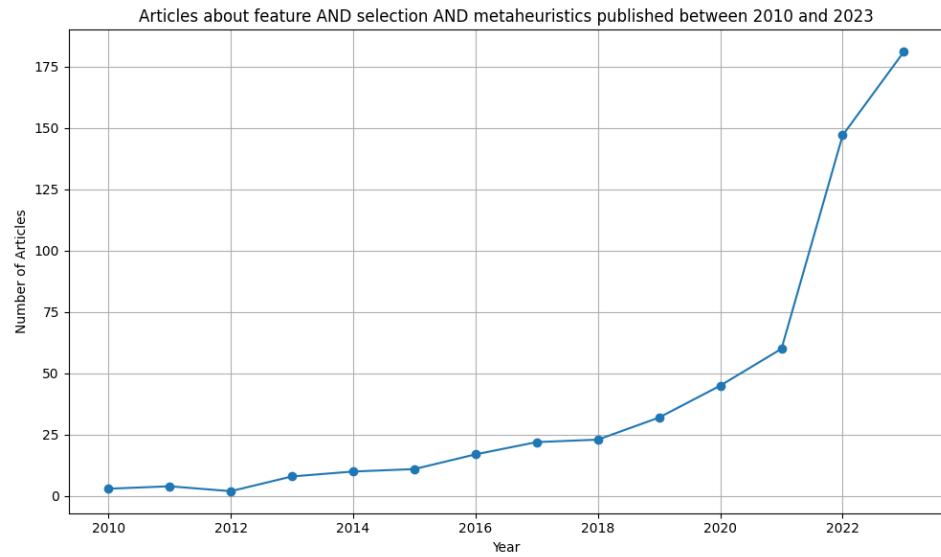


Figura 4.2: De igual forma que en la figura 4.1 las mateheuristicas han ido muy ligadas a la resolución de este problema. Se ha usado el buscador Scopus para estos resultados.

abordado con técnicas metaheurísticas 4.2.

Y es que las metaheurísticas son algoritmos muy convenientes en la resolución de este tipo de problemas. Los algoritmos analíticos pueden enfrentar dificultades cuando el conjunto de datos tiene un número muy elevado de características, debido a que la cantidad de combinaciones posibles puede hacer que el problema sea inabordable en términos computacionales. Debido a ello, se suelen utilizar algoritmos más rápidos cuya solución no es óptima en términos globales, pero si suficientemente buena.

Muchos algoritmos metaheurísticos han sido propuestos a lo largo de los años. Se han escogido los más destacables en cuanto a resultados y más mencionados entre todos ellos, dando una selección de algoritmos metaheurísticos modernos de, en principio, muy alta calidad. No solo son seleccionados algoritmos modernos, sino que también se han escogido una serie de algoritmos más “clásicos”, pero cuya aplicación es más extendida y con resultados que, de forma empírica, han demostrado ser más que buenos a lo largo de décadas de uso.

Algoritmos	Algoritmos
GWO [34]	PSO [41]
FA [35]	GA [42]
GOA [36]	ACO [43]
WOA [37]	DE [44]
DA [38]	ABCO [45]
CS [39]	
BA [40]	

Tabla 4.2: Algoritmos clásicos

Tabla 4.1: Algoritmos modernos

Los algoritmos usados en este proyecto pertenecen todos a la sub-categoría conocida como algoritmos poblacionales, esto es, algoritmos que parten de un conjunto de soluciones inicializadas de forma aleatoria y representadas en forma de vectores llamadas población. Este tipo de algoritmos “evolucionan” durante un número de generaciones hasta alcanzar un máximo número de generación o alcanzar un miembro de la población cuya solución sea lo suficientemente buena [46].

4.1. Introducción a los primeros algoritmos poblacionales

Uno de los primeros algoritmos basados en poblaciones y evolutivos es el algoritmo genético. Los *algoritmos genéticos* o **GAs** ganaron popularidad en los años 70, particularmente en 1975 con el libro de John Holland [42].

Este tipo de metaheurísticas son diseñadas basándose en la selección natural. Un conjunto de fenotipos (soluciones) son evolucionadas durante generaciones para emular el cruce entre especies (cruce de soluciones mediante un intercambio común de cromosomas) dando lugar a nuevos individuos con características de ambos padres. Poco a poco este tipo de algoritmos fueron desarrollando nuevas características y aunque en un principio fueron diseñados para resolver problemas discretos, también tienen versiones que optimizan problemas continuos [10].

Uno de los primeros algoritmos metaheurísticos basados en poblaciones es el *sistema de hormigas* de Dorigo, que fue presentado en 1991 [47]. Esta metaheurística fue aplicada a la resolución del problema del vendedor ambulante con resultados muy prometedores, pues resolvía el problema de forma muy eficiente con una solución sub-óptima, pero suficientemente buena. A partir de este punto, surgieron numerosas mejoras y versiones de los algoritmos basados en colonias de hormigas.

Más tarde se presentó el conocido algoritmo de *optimización por enjambre de partículas*, nombrado como **PSO** por sus siglas en inglés y presentado por Kennedy y Eberhart en 1995 [41]. El algoritmo **PSO** surgió a partir de la observación del comportamiento social de las aves y peces. La idea principal del **PSO** es que cada solución candidata (o “partícula”) en el espacio de búsqueda se mueve a través del espacio en función de su propia mejor posición conocida y la mejor posición conocida de cualquier partícula en el enjambre, en lugar de depender de operadores de búsqueda convencionales. El algoritmo comenzaba con un conjunto de soluciones candidatas aleatorias, denominadas partículas, que procedían a moverse a través del espacio de búsqueda para encontrar la solución óptima. Cada partícula tenía una posición y una velocidad asociadas.

En 1999 se presentó el conocido **ACO** u *optimización de colonia de hormigas* [43]. Este algoritmo refinada la fórmula anteriormente usada en **AS** usando un grafo como representación del conjunto de soluciones, la actualización y rastro de feromonas como heurística principal para escoger el camino en el grafo y una codificación discreta de las características.

La *evolución diferencial* fue un algoritmo de optimización propuesto por primera vez por Rainer Storn y Kenneth Price en el año 1997 [44]. Surgió como una alternativa a los algoritmos genéticos existentes en ese momento, ya que se inspiraba en su diseño, pero siendo inicialmente diseñado para resolver problemas continuos, a diferencia de los **GAs**.

La idea detrás de la evolución diferencial se basaba en la exploración de un espacio de búsqueda mediante la manipulación de vectores de parámetros. A diferencia de los *GAs*, que utilizaban operadores genéticos como la cruza y la mutación, la **DE** se centraba principalmente en la mutación basada en la

diferencia entre vectores de la población. Inicialmente, el algoritmo atrajo la atención debido a su simplicidad y eficiencia en la optimización de funciones complejas en espacios de búsqueda continuos.

El algoritmo de optimización basado en colonias de abejas es considerado otro gran clásico de las metaheurísticas poblacionales bio-inspiradas, pese a ello, es el más nuevo de los ya mencionados. El algoritmo de colonias de abejas artificiales o **ABCO** fue propuesto por primera vez por Dervis Karaboga en 2005 [45]. Surgió como una técnica inspirada en el comportamiento de las colonias de abejas para resolver problemas de optimización y se basaba en imitar el comportamiento de las abejas en la búsqueda de alimentos. En la naturaleza, las abejas utilizan la danza de reclutamiento para comunicar la ubicación de las fuentes de alimento a otras abejas en la colmena. Karaboga adaptó este proceso en un algoritmo de optimización que utiliza abejas artificiales para explorar y explotar un espacio de búsqueda.

4.2. Propuestas metaheurísticas modernas

Con el paso de los años se han ido refinando y probando nuevos tipos de algoritmos. Nuevas inspiraciones, basadas en el comportamiento animal, en teoría social, en teoría física o matemática. También se han creado nuevas versiones de algoritmos clásicos para resolver ciertos problemas que el algoritmo tenía de base o abordar otros nuevos. De cualquier manera, el número de nuevas metaheurísticas no ha dejado de crecer desde su origen, pues sus usos son muy extensos. Desde la selección de características hasta métodos de búsqueda de hiper-parámetros muy eficientes y con muy buenos resultados [48].

Dada una exhaustiva investigación, que incluye múltiples búsquedas en los últimos años en el motor de búsqueda de Scopus y una extensa lectura de varios estudios (“surveys”) [49-51] enfocados en metaheurísticas aplicadas a la **selección de características** se ha hecho una lista de los más usados, citados y con mejores resultados.

El más “antiguo”, dentro de la novedad, de los siete algoritmos aquí citados, es el de la *búsqueda cuco* o **CS** dadas sus siglas en inglés. Este algoritmo fue presentado a finales de 2009. En el artículo original [39] comparaba su rendimiento con un clásico ya mencionado en este capítulo, el **PSO**.

Esto no es novedad, todos los algoritmos más recientes hacen gala de sus buenos resultados en comparación a algoritmos clásicos como son el **ABCO**, **DE** o **PSO** entre otros muchos.

La búsqueda cuco se inspiraba en el comportamiento parasitario de los pájaros de cuyo nombre de apropió. Estos colocan sus huevos en los nidos de otras

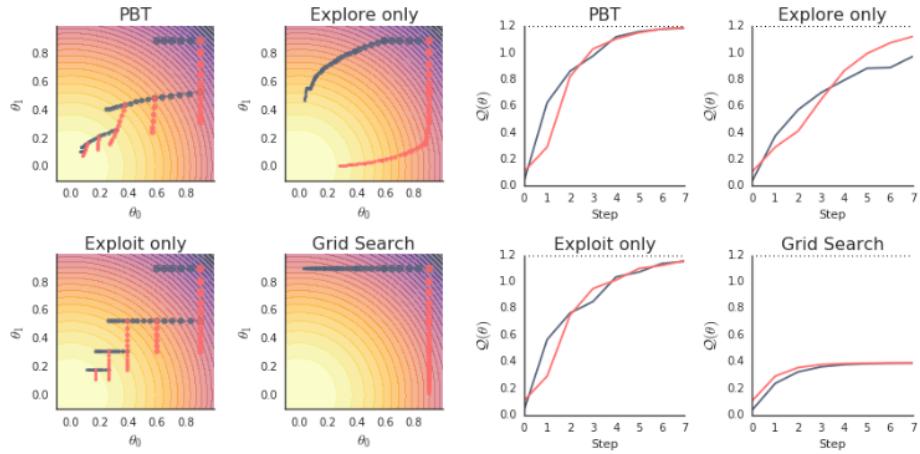


Figura 4.3: En esta figura se muestra varios métodos usados en la búsqueda de hiper-parámetros frente al método basado en poblaciones **PBT**. La zona más blanca representa el mínimo, ya que se trata de minimizar un error de pérdida a la vez que se maximiza una métrica concreta. Population-based training (**PBT**) es una técnica en el ámbito del aprendizaje automático y el entrenamiento de modelos que combina conceptos de algoritmos genéticos y métodos de ajuste de hiper-parámetros para mejorar el rendimiento y la eficiencia del entrenamiento de modelos. [48]

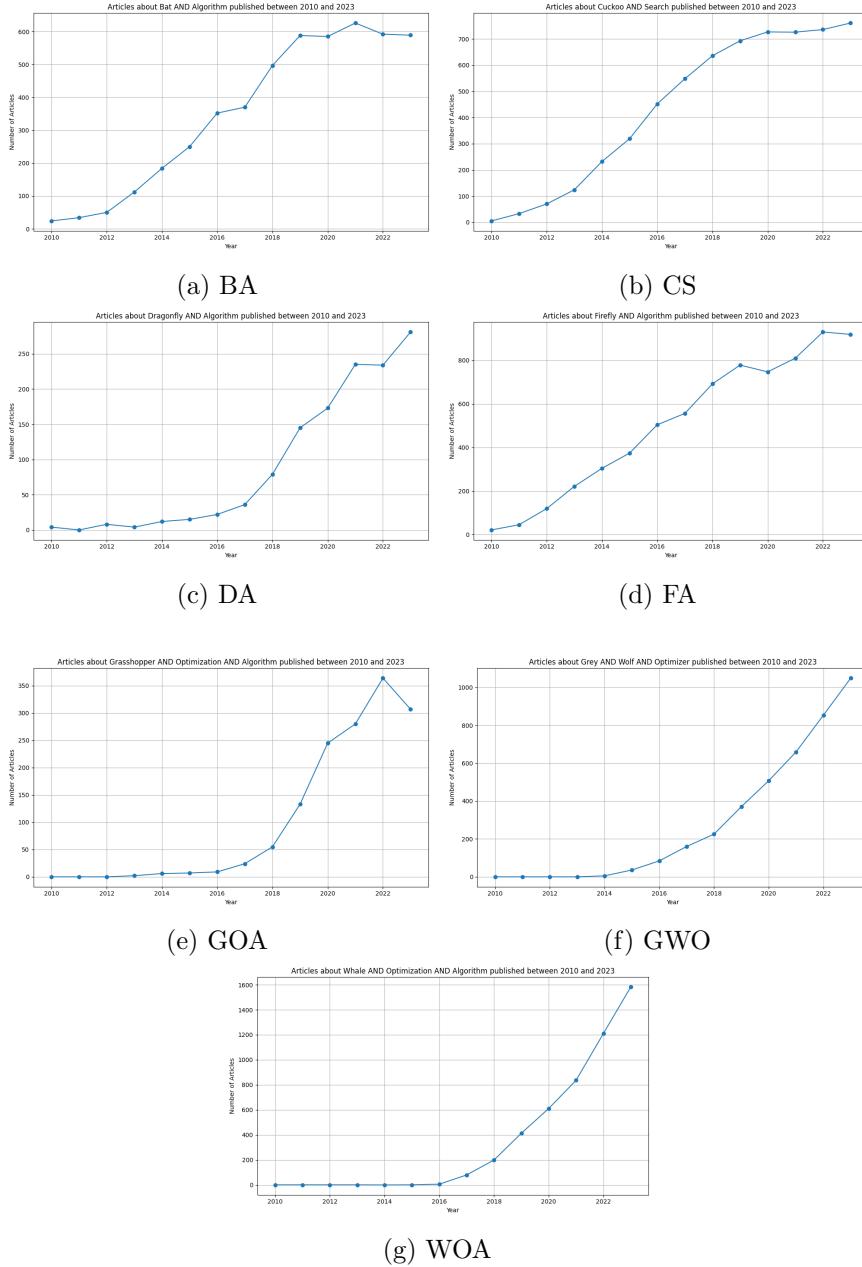


Figura 4.4: Número de búsquedas en Scopus de los diferentes algoritmos metaheurísticos seleccionados. Se puede ver una clara tendencia alcista en la popularidad de estos pese a su relativa novedad en el campo de la optimización.

aves y además retiran los huevos ajenos para incrementar las posibilidades de incubación de los suyos. Esto pequeña premisa se simula matemáticamente por los agentes de la población y se combina con el uso de funciones que simulan vuelos Lévy (pues se ha observado que el patrón de vuelos de muchas aves e insectos tienden a encajar en ese tipo de distribución [52]).

El *algoritmo de los lobos grises* o **GWO** [34] y el *algoritmo de optimización de la ballena* o **WOA** [37] comparten similitudes en la forma que actúan sus operadores. Estos se basan en la búsqueda de una presa y en el rodeo de la misma, incorporando siempre una variable aleatoria que va dejando espacio, según avanzan las iteraciones, a la convergencia de la solución.

Otros, como el **DA** [38] y el **GOA** [36], comparten la idea de la atracción y la repulsión para equilibrar la balanza entre exploración y explotación. En el caso del algoritmo de la libélula es atracción por presas (soluciones muy prometedoras) y repulsión hacia enemigos (peores soluciones encontradas), mientras que el algoritmo de optimización del saltamontes la atracción y la repulsión están relacionadas directamente con el resto de soluciones (agentes). Se repelen entre sí para favorecer la exploración del espacio y a medida que se avanza en las iteraciones o se encuentran soluciones suficientemente buenas, se incrementa la atracción entre saltamontes para dar lugar a la explotación.

Estas nuevas metaheurísticas aportaron nuevas ideas muy novedosas e interesantes. Otros algoritmos incluidos en la recopilación de este proyecto y que han sido muy mencionadas y usadas son las de **BA** [40] y **FA** [35]. **BA** (algoritmo del murciélagos) es inspirada por la forma en la que se guían (y cazan) los murciélagos, la emisión de pulsos de ecolocalización. **FA** (algoritmo de las luciérnagas), en cambio, se basa en la emisión de luz para atraer presas, atraer parejas y repeler enemigos.

4.3. Versiones binarias

Esta selección de siete metaheurísticas es, según la investigación realizada, la más representativa de la actualidad en cuanto al problema de selección de características se refiere. Su diseño permite fácilmente la modificación del algoritmo para adaptarlos al problema de selección de características, el cual requiere de una codificación discreta de los vectores soluciones. Algunas de las propuestas más novedosas para el problema de selección de características son las de **bGWO** [53-55], **bWOA** [56, 57], **bDA** [58-61], **bGOA** [62-64], **bBA** [65-67], **bCS** [68] y **bFA** [69-72]. Por su parte, los algoritmos clásicos también tienen múltiples versiones “binarias” o discretas, véase **bPSO** [73-76], **bABCO** [77-80] o **bDE** [79, 81-83]. Nótese que los algoritmos genéticos no tienen como tal una versión binaria como el resto de algoritmos, pues estos

fueron pensados originalmente para resolver problemas discretos. Gracias a que sus operadores son muy flexibles es posible también que acepten una codificación real y que resuelvan ese tipo de problemas con un rendimiento muy alto. En cuanto a los algoritmos de hormigas, tampoco necesitan una modificación especial, pues fueron creados también para resolver problemas discretos. Pese a que **ACO** tenga versiones del algoritmo con codificación continua [84], en este proyecto se ha preferido obviarlas, pues se considera que alteran demasiado el diseño del algoritmo original como para considerarlo una versión en vez de una metaheurística nueva.

Puede observarse que la técnica más empleada para pasar de un algoritmo con codificación continua a codificación binaria es la del uso de *funciones de transferencia* [49, 73, 85]. De este tipo de funciones hay varias familias, pero las más usadas sin lugar a duda son las funciones del grupo “s-shaped” y “v-shaped”, nombre que proviene de la forma en “s” y “v” que tienen este tipo de funciones representadas en una gráfica. La premisa es sencilla, la característica x_i tendrá más probabilidad de ser 0 o 1 dependiendo de su valor tras pasar por la función de transferencia. Se representa por tanto el recorrido desde los valores para la $y \in [0, 1]$ como una probabilidad dada una entrada x .

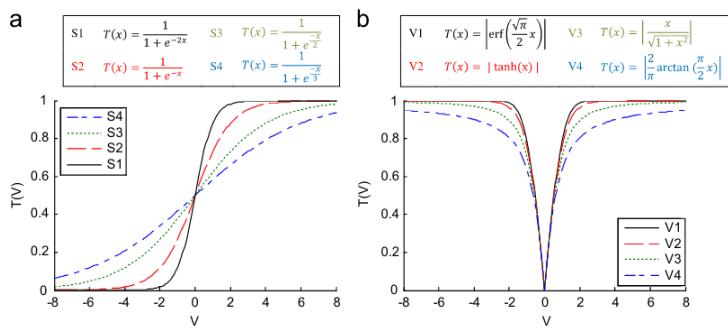


Figura 4.5: En esta figura extraída de [73] se muestran funciones del tipo **S-shaped** (a) y **V-Shaped** (b).

Capítulo 5

Descripción de los algoritmos

Para aclarar ciertas cuestiones referentes a las soluciones en este tipo de algoritmos estocásticos de tipo evolutivo, si no se especifica un tipo de representación se da por hecho que las soluciones se representan como vectores en un espacio n -dimensional. Un conjunto de individuos/vectores dan como resultado una población. La población es una matriz de $N \times M$, donde N es el número de filas o número de individuos en la población y M es el número de características del problema (M dimensiones). Algunos algoritmos admiten codificación binaria y otros continua, ese tipo de cuestiones si serán especificadas.

Por lo general, excepto para los algoritmos **GA** y **ACO**, se utilizarán funciones de transferencia a la hora de modificar los algoritmos para su versión binaria, tal y como se ha mencionado en el repaso bibliográfico.

5.1. Algoritmos genéticos

5.1.1. Introducción

Los algoritmos genéticos están inspirados en la selección natural y se utilizan tanto en problemas de optimización con restricciones como sin ellas. Es una metaheurística que modifica una población de soluciones individuales de manera repetida, seleccionando soluciones “padre” que dan lugar a la siguiente generación de soluciones en la siguiente iteración del algoritmo. En su forma más básica, un algoritmo genético opera sobre una población de soluciones potenciales a un problema dado. Cada solución potencial, a menudo llamada individuo o cromosoma, está representada como una cadena de símbolos, que puede ser binaria, numérica o simbólica [86].

5.1.2. Funcionamiento y Operadores

Las soluciones o **fenotipos** están compuestas por una serie de propiedades, características, cromosomas o **genotipos**, que pueden ser mutados y alterados. La representación más común es la binaria, pero otras codificaciones son posibles.

El proceso típico incluye [87]:

- Inicialización aleatoria de la población.
- Evaluación del *fitness* de cada individuo.
- Selección de padres basada en el *fitness*.
- Creación de una nueva generación mediante cruces y mutaciones.
- Elitismo: selección de los mejores individuos [88].

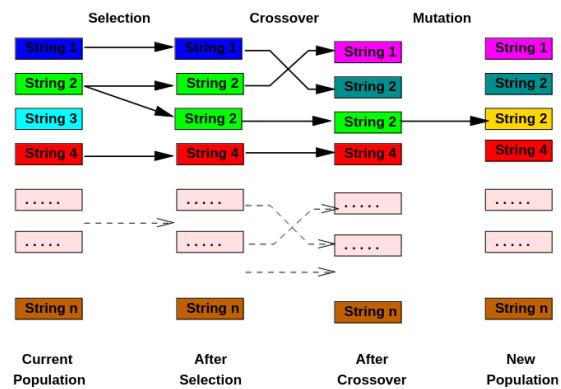


Figura 5.1: Figura obtenida de [87]

El algoritmo termina cuando se alcanza cierta tolerancia de error o número máximo de iteraciones.

Selección

La selección se puede llevar a cabo con varios métodos, uno de los más utilizados en selección por torneo.

En la selección por torneo, los individuos compiten entre sí en grupos pequeños. Cada grupo (torneo) tiene varios competidores (individuos). El competidor con la mejor aptitud (**fitness**) dentro del grupo tiene más probabilidad de ganar. Los ganadores son seleccionados para “reproducirse”.

Este proceso se repite hasta que se completa el *pool* de apareamiento con los ganadores de los torneos [89]. La probabilidad de cada individuo en base a su **fitness** se calcula de la siguiente manera:

$$P(\text{parent}_i) = \frac{\text{fitness}(\text{parent}_i)}{\sum_{j=1}^N \text{fitness}(\text{parent}_j)} \quad (5.1)$$

Para las versiones binaria y continua del algoritmo se utilizan diferentes operadores de mutación y cross-over.

Cruce binario

One-point crossover: Se elige al azar un punto en los cromosomas de ambos progenitores y se designa como "punto de cruce". Los bits a la derecha de ese punto se intercambian entre los dos cromosomas parentales. El resultado son dos descendientes, cada uno con información genética de ambos progenitores [90].

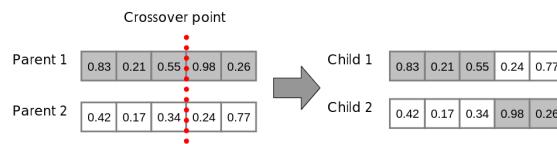


Figura 5.2: One-point crossover [91]

Cruce continuo

Blend crossover: Dado dos números reales para cada uno de los genes de los padres al hijo se le asignará un número aleatorio entre ese rango de gen para cada gen que conforme al vector cromosómico [91].

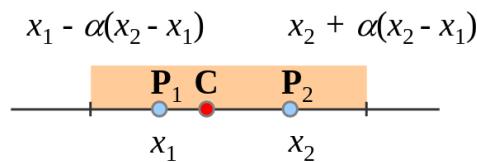


Figura 5.3: Blend crossover [91]

Mutación binaria

Su propósito es mantener la diversidad genética de los cromosomas de una población de un algoritmo evolutivo. El operador más básico y clásico consiste en cambiar un bit arbitrario de un genotipo o solución de un algoritmo genético binario a su estado inverso dada una probabilidad de mutación [88].

Mutación continua

Esta mutación se utiliza especialmente en problemas en los que se busca una exploración más amplia del espacio de búsqueda, ya que la distribución de Cauchy tiene colas pesadas y puede generar valores alejados del centro con mayor probabilidad en comparación con una distribución normal:

$$m(x_i) = \begin{cases} (2r)^{\frac{1}{\eta+1}} - 1 & \text{if } p \leq 0.5 \\ 1 - (2r)^{\frac{1}{\eta+1}} & \text{if } p > 0.5 \end{cases} \quad (5.2)$$

Esta función de mutación está diseñada para generar una exploración más amplia del espacio de búsqueda, con la posibilidad de generar valores más alejados del centro de la distribución. La variable r es un número a aleatorio y η es una variable de control.

Elitismo

El elitismo garantiza que las soluciones de alta calidad sobrevivan en las generaciones futuras, ayudando en la explotación del espacio de búsqueda. La cantidad de individuos élite seleccionados debe ser elegida con precisión [88].

5.2. Algoritmos de colonias de hormigas

5.2.1. Introducción

El algoritmo de colonia de hormigas o **ACO** es una técnica probabilística que trata de resolver problemas computacionales que pueden ser reducidos a la búsqueda de caminos óptimos a través de grafos. Se basa en el comportamiento de las hormigas reales y su comunicación vía feromonas. Las hormigas vagan por el mundo en búsqueda de comida de forma aleatoria. Cuando estas encuentran comida, dejan trazas de feromonas en su camino, de esta forma, si otras hormigas encuentran ese rastro, se hace más probable que recorran ese camino y dejen de vagar de forma aleatoria. Sin embargo, las trazas de feromonas se evaporan con el tiempo, perdiendo su fuerza de atracción sobre otras hormigas. De esta forma, si el camino es muy largo, la hormiga tardará

mucho en recorrerlo y la traza de feromonas tendrá más tiempo a evaporarse. De manera análoga, a más corto es el camino, más rápido se recorre y más densidad de feromonas acumula [92].

5.2.2. Funcionamiento y Operadores

Para poder adaptar el algoritmo ACO a un problema es necesario reducir ese problema a la búsqueda del camino más corto dentro de un grafo ponderado. De esta forma, la **representación** en este tipo de algoritmo debe ser la de un grafo. Cada característica del problema original se representa como un nodo n . Los caminos que conectan a los nodos (e) representan la elección del subconjunto de características. El camino deberá ser lo más corto posible maximizando a su vez el **accuracy** [92].

Regla de transición probabilística

$$P_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_l \tau_{il}^\alpha \eta_{il}^\beta} & \text{Si } l \text{ y } k \text{ son nodos admisibles} \\ 0 & \text{De lo contrario} \end{cases} \quad (5.3)$$

Donde:

- $P_{ij}^k(t)$ denota la probabilidad de transición de un nodo de i a j en la k -hormiga (agente) en el instante de tiempo t .
- τ_{ij} es la cantidad de traza de feromona en la arista (i, j) en el momento t . η_{ij} es la heurística de deseabilidad o visibilidad de la arista.
- β y α son dos parámetros que controlan la importancia relativa del valor de la feromona vs la información de la heurística.

Evaporación de feromona

Después de que todas las hormigas hayan terminado su camino, la evaporación de feromonas comienza. El contenido de feromonas del camino (i, j) en el instante $t + 1$ es:

$$\tau_{ij}(\text{new}) = (1 - p)\tau_{ij}(t) + \sum_{k=1}^m \Delta\tau_{ij}^k(t) + \Delta\tau_{ij}^g(t) \quad (5.4)$$

Donde:

- $\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{F^k} & \text{Si la hormiga } k \text{ pasa por la arista } (i,j) \text{ en } T^k \\ 0 & \text{De lo contrario} \end{cases}$

- $p \in (0, 1]$ es el ritmo de evaporación.
- m es el número de hormigas.
- $\Delta\tau_{ij}^k$ y $\Delta\tau_{ij}^g$ son respectivamente, la cantidad de feromonas colocadas en la arista (i, j) por la hormiga k y la cantidad de feromonas depositadas por la mejor hormiga g en el instante t sobre la arista (i, j) .
- Q es una constante y F^k es el valor de coste de la solución encontrada por la hormiga k en el tour T^k , es decir, F^k es el *fitness* de la hormiga k .

Se utiliza el sistema *max-min*, solo la mejor hormiga puede depositar feromona.

En el algoritmo original, no había heurística de deseabilidad o visibilidad (η), solo información en términos de feromona [43].

5.3. Optimización por enjambre de partículas

5.3.1. Introducción

El algoritmo conocido como optimización por enjambre de partículas o en inglés *particle swarm optimization* fue concebido en 1995 por James Kennedy y Russel Eberhart [41]. En este algoritmo, un conjunto de soluciones candidatas, llamadas partículas, se mueven en un espacio de búsqueda multidimensional. Cada partícula ajusta su posición de acuerdo con su experiencia personal y la experiencia del grupo.

Para la versión binaria se utilizan funciones de transferencia.

5.3.2. Funcionamiento y Operadores

Las soluciones son descritas como individuos dentro de un enjambre (conjunto total de posibles soluciones) que tienen una posición y una velocidad en todo momento. La posición de estos individuos representa la solución en sí.

Velocidad

$$v_i(t+1) = v_i(t) + c_1 \cdot \text{rand}() \cdot (pbest_i - x_i(t)) + c_2 \cdot \text{rand}() \cdot (gbest - x_i(t)) \quad (5.5)$$

Donde:

- $v_i(t + 1)$ es la velocidad de la partícula i en la siguiente iteración.
- $v_i(t)$ es la velocidad de la partícula i en la iteración actual.
- c_1 y c_2 son factores de aceleración que controlan la influencia de las mejores posiciones.
- $\text{rand}()$ es un número aleatorio en el rango $[0, 1]$.
- $pbest_i$ es la mejor posición histórica de la partícula i .
- $x_i(t)$ es la posición actual de la partícula i en la iteración t .
- $gbest$ es la mejor posición global del grupo.

Posición

$$x_i(t + 1) = x_i(t) + v_i(t + 1) \quad (5.6)$$

Estas ecuaciones se utilizan para guiar el movimiento de las partículas a través del espacio de búsqueda hacia las mejores soluciones, adaptándose tanto a la mejor solución encontrada por la propia partícula ($pbest$) como a la mejor solución encontrada por cualquier partícula en el enjambre ($gbest$) [41]. Con el tiempo, las partículas tienden a converger hacia las mejores soluciones conocidas.

5.4. Evolución diferencial

La evolución diferencial surgió como un algoritmo de optimización para problemas no diferenciables y no lineales, con pocas variables de control, robusto, paralelizable y muy eficaz. El algoritmo de evolución diferencial o *differential evolution* en inglés fue diseñado para resolver problemas continuos, a diferencia del propósito original de su “hermano gemelo” el algoritmo genético, cuya finalidad inicial era resolver problemas binarios. Este algoritmo fue presentado en 1996 por Storn y Price en [44].

En la evolución diferencial, la diferencia de vectores es un concepto clave que se utiliza en el operador de mutación. Esta se calcula restando dos vectores de la población. Gracias a esta operación puede calcularse la distancia entre vectores y esto conlleva una serie de ventajas [44]:

- Las posiciones de los individuos proporcionan información ya que si los individuos están bien distribuidos (y si la población es lo suficientemente grande), la población inicial será una buena representación de todo el espacio de búsqueda.

- Las distancias entre estos individuos serán inversamente proporcionales al tamaño de la población.
- A medida que avanza la búsqueda y los individuos comienzan a converger hacia un óptimo local, las distancias entre los individuos comenzarán a disminuir.

Para la versión binaria se utilizan funciones de transferencia.

5.4.1. Funcionamiento y Operadores

La representación suele ser la misma que en otros algoritmos metaheurísticos. DE fue diseñado para optimizar un problema continuo, por lo que la población debería ser un conjunto de números reales.

Mutación

La operación de mutación produce un vector, conocido como vector de prueba o en inglés *trial vector*. Para ello se hace uso de un vector objetivo y una diferencia de vectores ponderada. La mutación elige a un parente x_i , generando un vector de prueba u_i siguiendo los siguientes pasos:

1. Se selecciona un vector objetivo x_{i_1} tal que $i \neq i_1$.
2. Se selecciona aleatoriamente dos individuos x_{i_2} y x_{i_3} de la población tal que $i \neq i_1 \neq i_2 \neq i_3$ y i_2, i_3 sean seleccionados con una probabilidad uniforme. Es necesario que todos los individuos tengan la misma probabilidad de selección.

Finalmente, se calcula el vector de prueba:

$$u_i = x_{i_1} + \beta(x_{i_2} - x_{i_3}) \quad (5.7)$$

Donde β es un factor de escalado que controla la amplificación de la variación diferencial (mutación más diversa/grande) [44].

Una ventaja del uso de diferencias de vectores en la mutación es que ayuda a mantener la forma de la distribución en lugar de centrarse en el centro, como hacen muchos operadores de cruce. Además, al tener una media de la distribución cercana a cero, se minimiza la deriva genética [93]. Este fenómeno puede causar cambios en la composición genética de la población a lo largo de las generaciones, especialmente en poblaciones pequeñas. Por lo tanto, mantener una media cercana a cero ayuda a prevenir sesgos en la deriva genética.

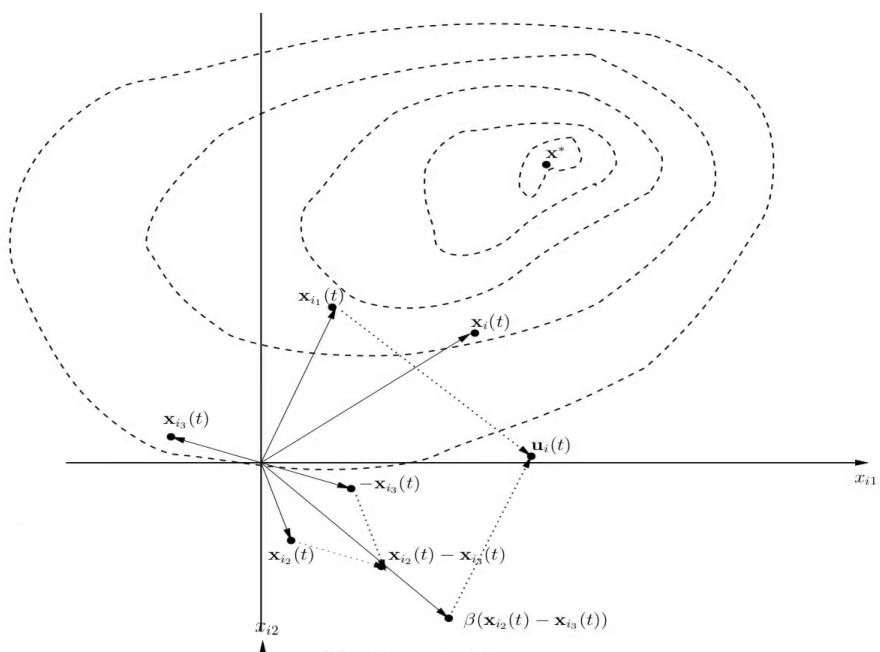


Figura 5.4: Esta figura ha sido seleccionada de [93]. Muestra de manera gráfica en un espacio bidimensional como funciona el operador de mutación y cómo se crea el “trial vector”.

Cruce

El cruce implementa una recombinación del vector de prueba u_i y el vector padre x_i para producir un vector hijo x'_i .

$$x'_{ij} = \begin{cases} u_{ij} & \text{Si } j \in \mathcal{J} \\ x_{ij} & \text{De lo contrario} \end{cases} \quad (5.8)$$

Donde \mathcal{J} es un conjunto de puntos de perturbación. Estos índices pueden ser calculados de muchas formas, por ejemplo, de manera aleatoria.

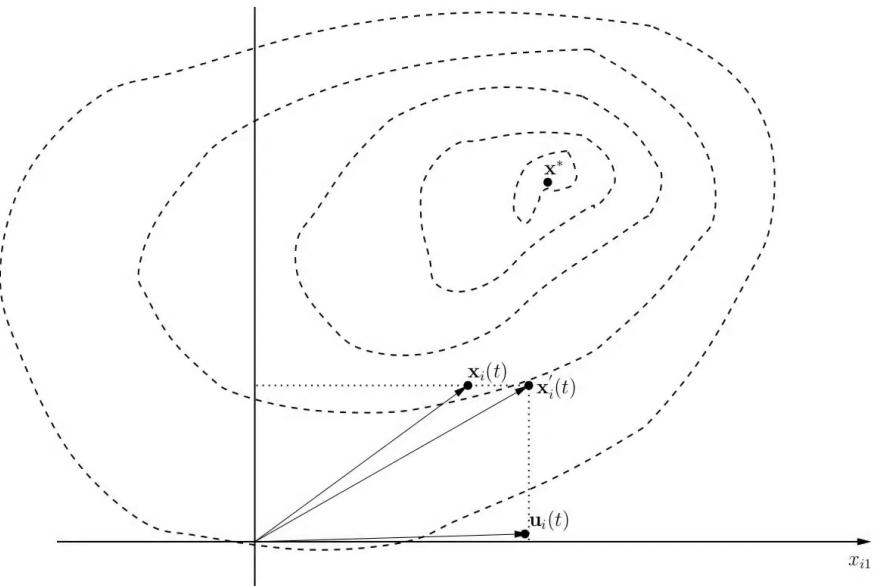


Figura 5.5: Esta figura ha sido seleccionada de [93]. Muestra de manera gráfica en un espacio bidimensional como funciona el operador de cruce al recombinar los genes/características de los vectores padre y “trial”.

Selección

Si el vector hijo, al evaluarlo, tiene un valor *fitness* mejor que el vector padre, entonces se sustituye el vector x_i por el vector x'_i . De otra forma, el vector padre continuará en la población para la siguiente generación [44].

5.5. Optimización por colonia de abejas artificial

5.5.1. Introducción

El algoritmo del enjambre de abejas artificial es un algoritmo estocástico evolutivo, al igual que los ya mencionados anteriormente, que se basa en el comportamiento de los enjambres de abejas para la optimización de problemas. El modelo usado en este algoritmo depende de tres factores, las fuentes de alimento, las abejas buscadoras empleadas y las abejas buscadoras libres (hay dos tipos) [45].

Para la versión binaria se utilizan funciones de transferencia.

5.5.2. Funcionamiento y Operadores

Los operadores del algoritmo son los siguientes:

1. **Abejas Empleadas (Employed Bees):** Estas abejas están asociadas con fuentes de alimento específicas y son responsables de explotar estas fuentes. Llevan información sobre la fuente de alimento, incluyendo la distancia, la dirección y la rentabilidad, y comparten esta información dentro de la colmena, influyendo en las decisiones de otras abejas.
2. **Abejas Observadoras (Outlooker Bees):** Estas abejas esperan dentro de la colmena y toman decisiones sobre a qué fuentes de alimento dirigirse basándose en la información proporcionada por las abejas empleadas. Seleccionan las fuentes de alimento según la probabilidad que está directamente relacionada con la rentabilidad de las fuentes.
3. **Abejas Exploradoras (Scout Bees):** Son abejas que buscan nuevas fuentes de alimento sin información previa. Su rol es crucial para la exploración y el descubrimiento de nuevas fuentes de alimento. Cuando una fuente se agota o no mejora después de cierto número de intentos (determinado por un parámetro de control llamado "límite"), una abeja empleada puede convertirse en exploradora.

Estos operadores trabajan en un ciclo repetitivo donde las abejas empleadas primero van a las fuentes de alimento y actualizan la información sobre su rentabilidad. Luego, las abejas observadoras seleccionan fuentes de alimento basadas en esta información actualizada y las abejas exploradoras buscan nuevas fuentes [45]. Este proceso se repite hasta cumplir con los criterios de terminación del algoritmo.

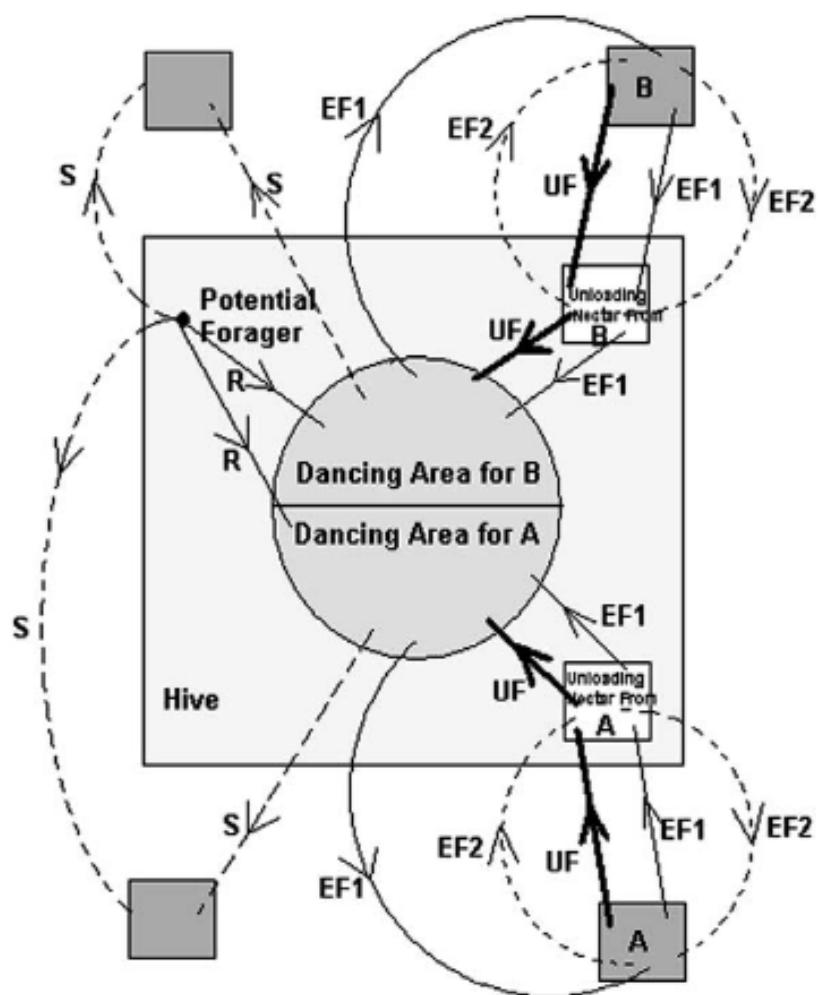


Figura 5.6: Figura obtenida de [94].

5.6. Algoritmo de optimización del saltamontes

5.6.1. Introducción

Los **grasshoppers** (saltamontes o langosta cuando están en enjambre) son insectos cuyo ciclo de vida consiste en tres etapas: *huevo, ninfa, adulto*. Un saltamontes puede ser encontrado en el enjambre en su estado ninfa y adulto. La diferencia entre ninfa y adulto es la velocidad de movimiento con pasos pequeños en la ninfa y lo contrario en su vida adulta (Las soluciones adultas se encargan de la exploración y las ninfas de la explotación) [36].

Para la versión binaria se utilizan funciones de transferencia.

5.6.2. Funcionamiento y Operadores

Operador de posición (primera propuesta)

La posición de un individuo se define en la siguiente ecuación:

$$x_i = r_1 s_i + r_2 g_i + r_3 a_i \quad (5.9)$$

Donde x_i es la posición del agente número i , s la interacción social, g la fuerza gravitacional, a la advección del viento y r_1, r_2, r_3 son números aleatorios entre $[0, 1]$. Esta función de actualización de posición no puede ser usada para la optimización debido a que se alcanza la zona de comfort de manera prematura y no se converge en un punto específico [36].

Posición (propuesta final)

$$X_i^d = c_1 \left(\sum_{j=1, j \neq i}^N c_2 \frac{ub_d - lb_d}{2} s(|x_j^d - x_i^d|) \frac{x_j - x_i}{d_{ij}} \right) + \hat{T}_d \quad (5.10)$$

Donde ud_b es la cota superior en la dimensión D , lb_d es la cota inferior en la dimensión D , \hat{T}_d es el valor de la dimensión D en el objetivo (la mejor solución encontrada hasta la fecha) y c es el coeficiente decreciente para hacer cada vez más pequeña la zona de comfort, zona de atracción y zona de repulsión.

No se considera la gravedad (G) y la dirección del viento siempre es a favor del objetivo \hat{T}_d .

Una diferencia con otros algoritmos de enjambres como PSO es que **GAO** actualiza la posición de cada saltamontes a partir de la posición actual, el mejor punto encontrado hasta el momento y la posición de todos los otros

agentes, mientras que **PSO** lo hace a partir de la posición actual del agente y la mejor posición encontrada [36].

Esto significa que PSO no tienen en cuenta al resto de agentes, mientras que GAO sí.

Los parámetros c_1, c_2 tienen dos propósitos:

- $c_1 \rightarrow$ Reduce el movimiento del enjambre entero alrededor del objetivo, es decir, maneja la relación entre exploración y explotación para el conjunto global de soluciones.
- $c_2 \rightarrow$ Reduce la zona de comfort y las regiones de repulsión y atracción. s es la función que decide si un agente debe sentirse atraído o repelido al objetivo, mientras que $\frac{ub_d - lb_d}{2}$ decrece de manera lineal el espacio de búsqueda a medida que c_2 decrece.

Ha de tenerse en cuenta también que en el proceso de búsqueda, la exploración debe ir antes que la explotación. En los saltamontes sin embargo, la fase explotativa que es la **ninfa** es la que se da primero. Por ello el parámetro c debe decrecer a medida que las iteraciones sobre el algoritmo van pasando. Este mecanismo hace que se promueva la explotación a medida que las iteraciones se incrementan, ya que se va reduciendo cada vez más el movimiento general del enjambre y se reduce la zona de comfort. c se actualiza por tanto así:

$$c = c_{\max} - l \cdot \frac{c_{\max} - c_{\min}}{L} \quad (5.11)$$

Donde l es la iteración actual y L el número máximo de iteraciones.

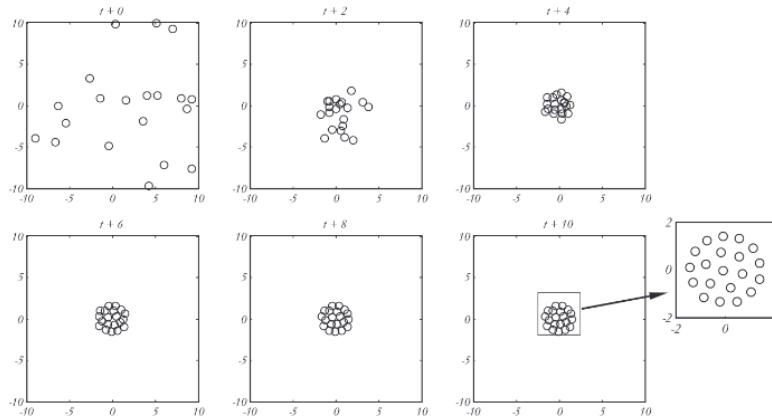


Figura 5.7: Esta figura ha sido seleccionada de [36]. Muestra la convergencia de los “saltamontes” en tan solo unas pocas iteraciones usando la fórmula 5.10.

5.7. Algoritmo de la libélula

5.7.1. Introducción

Los **dragonflies** (libélulas en español) son pequeños insectos que se pueden ver en enjambre solo cuando cazan o migran, dando lugar a dos tipos de enjambre:

- Las diferencias dadas entre ambos enjambres son notorias. En caza, se organizan en enjambres pequeños que vuelan en muchas direcciones en busca de comida. Este tipo de enjambre es llamado **estático**. Cuando se organizan en enjambres para migrar, estos son numerosos y vuelan en una sola dirección. Estos enjambres en contraposición a los anteriores son llamados **dinámicos** [38].
- Es obvio el paralelismo que se da entre las principales fases de la búsqueda optimizatoria (**exploración y explotación**). La fase dinámica es un paralelismo con la exploración, ya que se buscan pequeños objetivos en grupos por el espacio. La fase estática (explotación) se da cuando todos los agentes empiezan a migrar a una dirección concreta.

Para la versión binaria se utilizan funciones de transferencia.

5.7.2. Funcionamiento y Operadores

Separación

$$S_i = - \sum_{j=1}^N X - X_j \quad (5.12)$$

Se refiere a la elusión estática de colisión entre individuos con su vecindario. Donde X es la posición del agente actual y X_j la posición de su vecino número j . Mientras que N es el número total de agentes.

Alineamiento

$$A_i = \frac{\sum_{j=1}^N V_i}{N} \quad (5.13)$$

Se refiere a la coordinación de velocidad entre individuos de un vecindario. Donde V_j es la velocidad del individuo número j en el vecindario.

Cohesión

$$C_i = \frac{\sum_{j=1}^N X_j}{N} - X \quad (5.14)$$

Se refiere a la fuerza de atracción de los individuos hacia el centro de masa del vecindario. Se da por hecho un radio alrededor de cada libélula agrupando su vecindario, pues este es muy importante para el comportamiento de esta. En un enjambre dinámico, las libélulas tienden a alinear su vuelo al tiempo que mantienen una separación y cohesión adecuadas.

En un enjambre estático, sin embargo, las alineaciones son muy bajas mientras que la cohesión es alta para atacar a las presas. Por lo tanto, se asigna a las libélulas pesos de alta alineación y baja cohesión cuando exploran el espacio de búsqueda y de baja alineación y alta cohesión cuando explotan el espacio de búsqueda. Para la transición entre exploración y explotación, los radios de vecindad se incrementan proporcionalmente al número de iteraciones [38].

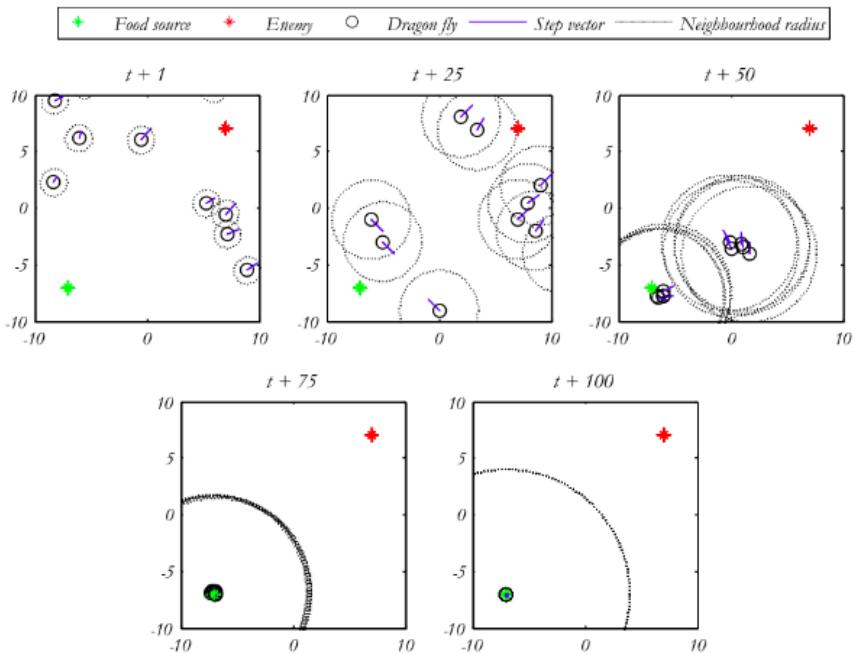


Figura 5.8: Vecindario de libélulas [38].

Atracción comida

$$F_i = X^+ - X \quad (5.15)$$

La fuerza de atracción hacia la comida (la mejor posición encontrada hasta el momento). Donde X^+ es la mejor posición encontrada hasta el momento.

Repulsión depredador

$$E_i = X^- + X \quad (5.16)$$

La fuerza de repulsión hacia un depredador (la peor posición encontrada hasta el momento). Donde X^- es la peor posición encontrada hasta el momento.

Dirección

$$\Delta X_{t+1} = (sS_i + aA_i + cC_i + fF_i + eE_i) + w\Delta X_t \quad (5.17)$$

El delta de X indica el vector dirección del movimiento de las libélulas. Cada elemento en minúscula es el factor del operador al que multiplica, de forma que es posible acentuar el efecto de cada uno de estos con múltiples combinaciones.

Posición

$$X_{t+1} = X_t + \Delta X_{t+1} \quad (5.18)$$

El operador de actualización de posición.

5.8. Algoritmo de optimización de ballenas

5.8.1. Introducción

Simula el comportamiento de las ballenas jorobadas en la caza, utilizando tanto el azar como el mejor agente de búsqueda para perseguir a la presa. Además, emplea una espiral para simular el mecanismo de ataque de la red de burbujas de las ballenas jorobadas [37].

Para la versión binaria se utilizan funciones de transferencia.

5.8.2. Funcionamiento y Operadores

Rodeado de presa

$$\vec{D} = |\vec{C} \cdot \vec{X}^*(t) - \vec{X}(t)| \quad (5.19)$$

$$\vec{X}(t+1) = \vec{X}^*(t) - \vec{A} \cdot \vec{D} \quad (5.20)$$

Donde t indica la iteración actual, \vec{A} y \vec{C} son vectores de coeficientes, X^* es el vector de posición de la mejor solución encontrada hasta el momento y \vec{X} es el vector posición.

Los vectores \vec{A} y \vec{C} se calculan como sigue:

$$\vec{A} = 2\vec{a} \cdot \vec{r} - \vec{a} \quad (5.21)$$

$$\vec{C} = 2 \cdot \vec{r} \quad (5.22)$$

Donde \vec{a} es decrementado linealmente desde 2 hasta 0 y \vec{r} es un vector aleatorio con valores en $[0, 1]$.

Ataque de red de burbujas (explotación)

- **Mecanismo de encogimiento del cerco** Este mecanismo es conseguido mediante el decremento del valor de \vec{a} . El vector \vec{A} es un vector de valores aleatorios en el intervalo $[-a, a]$ donde a es decrementado desde 2 hasta 0.

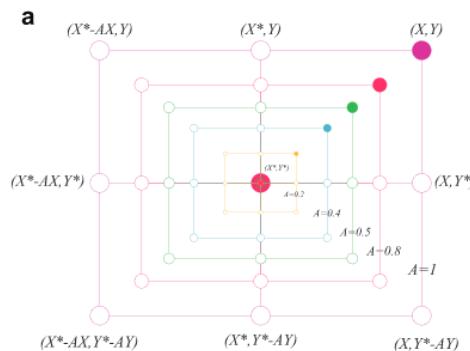


Figura 5.9: Ataque de red de burbujas usando el mecanismo de encogimiento del cerco [37].

- **Mecanismo de actualización de posición en espiral** Este mecanismo calcula la distancia entre la ballena (x, y) y la presa (x^*, y^*) . Una ecuación con forma de espiral es creada entonces entre la ballena y la presa para simular el comportamiento de estas al cazar.

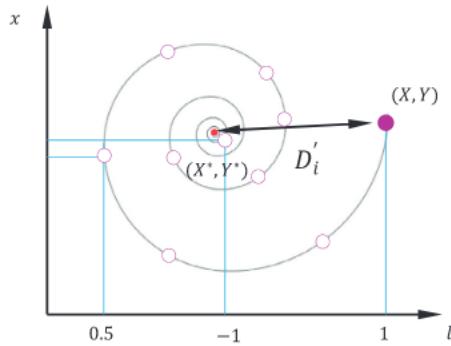


Figura 5.10: Ataque de red de burbujas usando el mecanismo de actualización de posición en espiral [37].

$$\vec{X}(t+1) = \vec{D}' \cdot e^{bl} \cdot \cos(2\pi l) + \vec{X}^*(t) \quad (5.23)$$

Donde $\vec{D}' = |\vec{X}^*(t) - \vec{X}(t)|$ e indica la distancia entre presa y ballena. b es una constante para definir la forma de la espiral, l es un número aleatorio entre $[-1, 1]$.

Se utilizan ambos mecanismos, dependiendo de si cierto número p aleatorio es mayor o menor a 0.5.

Búsqueda de la presa (exploración)

El mismo método utilizando el vector \vec{A} puede ser utilizado para la exploración. Se puede usar \vec{A} con valores más grandes a 1 o menores a -1 para forzar al agente a moverse lejos de otro agente.

$$\vec{D} = |\vec{C} \cdot \vec{X}_{rand} - \vec{X}| \quad (5.24)$$

$$\vec{X}(t+1) = \vec{X}_{rand} - \vec{A} \cdot \vec{D} \quad (5.25)$$

Cuando $|A| \geq 1 \rightarrow$ Exploración, cuando $|A| < 1 \rightarrow$ Explotación.

5.9. Algoritmo del murciélagos

5.9.1. Introducción

El algoritmo *metaheurístico* de los murciélagos se basa en la capacidad de eco-localización de estos para detectar a su presa y cazarla finalmente. La eco-localización de los murciélagos no solo les permite detectar a su presa, sino discriminar entre distintos tipos de insectos. Más concretamente, este

algoritmo se basa en los micro murciélagos, pues estos usan más frecuentemente la eco-localización en comparación a otras especies de murciélagos. Estos murciélagos emiten un pulso de sonido, el cual, rebota en los objetos y entidades cercanas, permitiendo al murciélagos componer una “visión” general de sus alrededores. La frecuencia del pulso emitido podría estar relacionada con distintas estrategias de caza [40].

Acústica de la eco-localización

Aunque cada pulso dura solo unos pocos milisegundos (entre 8 y 10 ms), mantiene una frecuencia constante, generalmente entre 25 kHz y 150 kHz. La longitud de onda (λ) de estos impulsos ultrasónicos, con una frecuencia (f) y velocidad del sonido (v) en el aire a 340 m/s, está dada por

$$\lambda = \frac{v}{f} \quad (5.26)$$

con longitudes entre 2 mm y 14 mm para frecuencias de 25 kHz a 150 kHz, respectivamente. Esta longitud de onda coincide con el tamaño de las presas. Los murciélagos pueden detectar obstáculos tan pequeños como cabellos humanos, utilizando la eco-localización para construir un modelo 3D del entorno, y discriminando distancias, orientaciones y tipos de presas. Aunque los murciélagos tienen buena vista y olfato, utilizan principalmente la eco-localización para la detección de presas y la navegación eficiente [40].

Para la versión binaria se utilizan funciones de transferencia.

5.9.2. Funcionamiento y Operadores

Para simplificar:

- Los murciélagos usan la ecolocación para percibir la distancia y distinguir alimento de barreras.
- Vuelan aleatoriamente con velocidad v_i y frecuencia fija f_{min} , ajustando longitud de onda λ y sonido A_0 .
- Se asume que la intensidad del sonido varía desde A_0 hasta un mínimo A_{min} .

Movimiento

Las soluciones (posiciones) y las velocidades de los murciélagos en un instante i son:

$$\begin{aligned}x_i^t &= x_i^{t-1} + v_i^t \\v_i^t &= v_i^{t-1} + (x_i^t - x_*) f_i \\f_i &= f_{\min} + (f_{\max} - f_{\min}) \beta\end{aligned}\quad (5.27)$$

Donde β es un vector aleatorio escogido de una distribución uniforme, x_* es la mejor posición global que es seleccionada tras comprobar los n murciélagos. Como el productor de $\lambda_i f_i$ es el incremento de la velocidad, se puede usar tanto uno como otro para ajustar la velocidad mientras el otro es fijado, todo depende del problema de interés [40]. En el documento original, se utiliza un rango para f de $[0, 100]$. A cada murciélago se le asigna una frecuencia uniformemente de $[f_{\min}, f_{\max}]$. Una vez se ha seleccionado una mejor solución, se actualizan los vectores posición de cada murciélago:

$$x_{\text{new}} = x_{\text{old}} + \epsilon A^t \quad (5.28)$$

Donde $\epsilon \in [-1, 1]$ es un número aleatorio y A^t es la intensidad sonora media de los murciélagos.

Intensidad sonora y Emisión de pulso

La intensidad sonora y el ratio de emisión de pulsos deben actualizarse según avance el número de iteraciones. A_i debe decrementarse a la vez que el ratio r_i se incrementa según el murciélago se acerca a su presa [40]. Se actualizan del siguiente modo:

$$A_i^{t+1} = \alpha A_i^t, \quad r_i^{t+1} = r_i^0 [1 - \exp(-\gamma t)] \quad (5.29)$$

Donde γ, α son constantes. Estos valores solo se actualizan si las nuevas soluciones son mejores, de manera que se guíe hacia un óptimo.

5.10. Optimización de los lobos grises

5.10.1. Introducción

- El algoritmo se inspira en el comportamiento de la manada de los lobos grises y sus escala jerárquica. Los lobos grises son super-depredadores, es decir, están en la cima de la cadena alimentaria [34]. Su escala jerárquica es la siguiente:

1. $\alpha \rightarrow$ Los lobos alfa son aquellos que lideran la manada, no por su fuerza, si no por su capacidad organizativa dentro de la manada. Estos solo puede procrear dentro de la manada y son los encargados de tomar decisiones importantes sobre todo el conjunto.
2. $\beta \rightarrow$ Los lobos beta son subordinados directos de los alfa. Son la opción más directa cuando se trata de sustituir a un alfa. Deben mostrar respeto a los alfa y comandan al resto de la manada según dicte el alfa.
3. $\omega \rightarrow$ El nivel más bajo de la cadena. Son los últimos en comer, pero no por ello carecen de importancia, pues la manada puede llegar a enfrentarse a graves problemas cuando este nivel de lobos falta. Son totalmente necesarios para mantener la estructura.
4. $\delta \rightarrow$ Lobos que someten a los omegas, pero que deben someterse a los alfas y omegas. A este grupo pertenecen los cazadores, ancianos, centinelas y cuidadores.

■ Modelado de jerarquía:

- Se considera la solución más buena como la solución α , siendo las siguientes en orden de mejor a peor **fitness** la β , la δ y por último la ω .
- La caza (optimización) es guiada por las soluciones α , β y δ mientras que las soluciones ω siguen a estos tres lobos.

Para la versión binaria se utilizan funciones de transferencia.

5.10.2. Funcionamiento y Operadores

Rodeo de la presa

$$\vec{D} = |\vec{C} \cdot \vec{X}_p(t) - \vec{X}(t)| \quad (5.30)$$

$$\vec{X}(t+1) = \vec{X}_p(t) - \vec{A} \cdot \vec{D} \quad (5.31)$$

Donde t indica la iteración actual, \vec{A} y \vec{C} son vectores de coeficientes, \vec{X}_p es el vector posición de la presa y \vec{X} el vector posición del lobo gris. Los vectores \vec{A} y \vec{C} se calculan:

$$\vec{A} = 2\vec{a} \cdot \vec{r}_1 - \vec{a} \quad (5.32)$$

$$\vec{C} = 2 \cdot \vec{r}_2 \quad (5.33)$$

Donde los componentes del vector \vec{a} son decrementados de 2 a 0 linealmente según avanzan las iteraciones. Las variables r_1 y r_2 son vectores aleatorios entre $[0, 1]$.

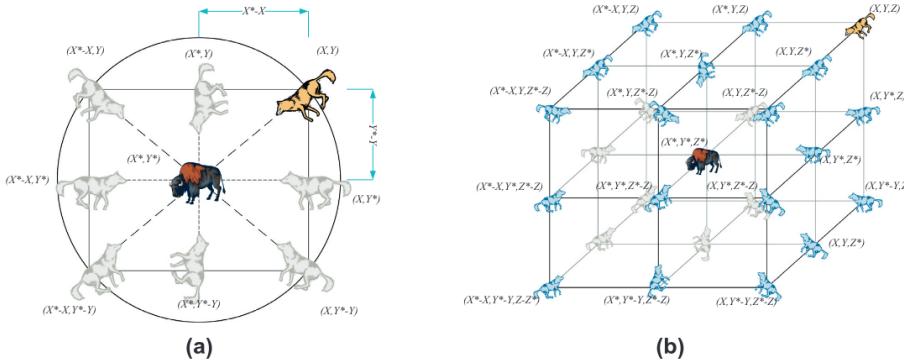


Figura 5.11: Esta figura ha sido seleccionada de [34]. Muestra en espacios 2D (a) y 3D (b) diferentes posiciones que un lobo puede tomar alrededor de la presa dada la ecuación 5.31. Define una vecindad alrededor de la solución en forma de hiper-esfera.

Caza

Se supone que el alfa, beta y delta tienen mayor conocimiento de donde está la presa (óptimo) y por tanto la posición del resto de lobos debe actualizarse siguiendo a estos tres primeros [34].

$$\vec{D}_\alpha = |\vec{C}_1 \cdot \vec{X}_\alpha - \vec{X}|, \quad \vec{D}_\beta = |\vec{C}_2 \cdot \vec{X}_\beta - \vec{X}|, \quad \vec{D}_\delta = |\vec{C}_3 \cdot \vec{X}_\delta - \vec{X}| \quad (5.34)$$

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1 \cdot \vec{D}_\alpha, \quad \vec{X}_2 = \vec{X}_\beta - \vec{A}_2 \cdot \vec{D}_\beta, \quad \vec{X}_3 = \vec{X}_\delta - \vec{A}_3 \cdot \vec{D}_\delta \quad (5.35)$$

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \quad (5.36)$$

Ataque a la presa (explotación)

Cuando los lobos están muy cerca de la presa (solución óptima), el valor de A disminuye, llevando a los lobos a converger hacia la presa. Esto se refleja en una reducción en la amplitud de los saltos en el espacio de búsqueda, facilitando una búsqueda local intensiva cerca de la mejor solución encontrada [34].

Búsqueda de la presa (exploración)

Para fomentar la exploración y evitar los óptimos locales, los vectores A pueden tomar valores que obliguen a los lobos a alejarse de la presa, explorando nuevas áreas del espacio de búsqueda. Este comportamiento se activa cuando los elementos de A son mayores que 1 o menores que -1, lo que amplía la

capacidad del algoritmo para explorar el espacio de búsqueda de manera más amplia [34].

5.11. Algoritmo de la luciérnaga

5.11.1. Introducción

Las luciérnagas son insectos capaces de producir un brillo único, muy visible y característico sobre todo de noche. El brillo que estas producen puede llegar a tener significados que hoy día siguen debatiéndose en la comunidad científica. Pese a ello, se conoce que este brillo tiene dos funciones fundamentales y bien definidas:

1. **Comunicación:** Sirve como instrumento para la comunicación entre dos luciérnagas para el apareamiento.
2. **Atracción:** Se usa para atraer a potenciales presas de la luciérnaga.

Por simplicidad del algoritmo, se asumen tres reglas sobre el comportamiento y naturaleza de las luciérnagas [35]:

1. Las luciérnagas solo tienen un sexo, por lo que todas se ven atraídas por todas y no hay discriminación más allá de que una luciérnaga sea más atractiva que otra.
2. La luminosidad es proporcional a lo atractiva que es una luciérnaga para el resto.
3. La luminosidad de una luciérnaga es determinada por los límites de la función objetivo.

Para la versión binaria se utilizan funciones de transferencia.

5.11.2. Funcionamiento y Operadores

Intensidad de la luz

$$\beta = \beta_0 e^{-\gamma r^2} \quad (5.37)$$

Donde β_0 es el atractivo en $r = 0$, es decir, en distancia igual a 0 y γ es un coeficiente fijo de absorción de luz [35]. La distancia entre dos luciérnagas es la distancia cartesiana:

$$r_{ij} = \|x_i - x_j\|_2 = \sqrt{\sum_{k=1}^d (x_{i,k} - x_{j,k})^2} \quad (5.38)$$

Movimiento

El movimiento de un agente (luciérnaga) es descrito por la siguiente ecuación, que según aumente el brillo y con ello el atractivo, hace que el vector posición se dirija hacia la luciérnaga atractiva:

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{ij}^2} (x_j^t - x_i^t) + \alpha \epsilon_i^t \quad (5.39)$$

Donde α es un parámetro de aleatoriedad y ϵ un vector de números alatorios sacados de una distribución Gaussiana o uniforme.

El parámetro α debe ser decrementado gradualmente de forma que las luciérnagas pasen de moverse tan aleatoriamente en un principio a converger en un punto óptimo al final [35]:

$$\alpha = \alpha_\infty + (\alpha_0 - \alpha_\infty) e^{-t} \quad (5.40)$$

Donde $t \in (0, t_{max}]$, siendo t_{max} el número máximo de generaciones, α_0 el parámetro inicial y α_∞ el valor final. De esta forma la exploración da paso a la explotación a medida que el número de iteraciones avanza.

5.12. Búsqueda Cuckoo

5.12.1. Introducción

El algoritmo de búsqueda de los cucos se basa en el tipo de ave de cuyo nombre se inspira el algoritmo. Se basa en el tipo de crianza de estos pájaros y su interesante comportamiento parasitario, así como en los vuelos Lévy [95].

El parasitismo de cría que adoptan estos pájaros implica poner huevos en nidos de otras aves. Los tipos incluyen parasitismo intra-específico, cría cooperativa y toma de nidos. Las aves anfitrionas pueden rechazar los huevos ajenos, lo que las lleva a abandonar el nido o construir uno nuevo. Algunos cucos parasitarios imitan los huevos de las aves anfitrionas para evitar el rechazo y aumentar el éxito reproductivo [39].

Para la versión binaria se utilizan funciones de transferencia.

5.12.2. Funcionamiento y Operadores

En el algoritmo de búsqueda cuco se implementa un tipo de vuelo Lévy, debido a que se ha observado que el vuelo de ciertos insectos y aves sigue este tipo de movimiento [39].

El algoritmo de búsqueda cuco (Cuckoo) se basa en tres reglas ideales simplificadas:

1. **Puesta de huevos:** Cada cuco pone un huevo a la vez y lo deposita en un nido elegido al azar. Nuevo huevo en nido $x_i^{t+1} = x_i^t + \alpha \bigoplus Levy(\lambda)$, siendo α el tamaño de paso y siendo el vuelo Lévy:

$$Levy \sim u = t^{-\lambda} \quad (5.41)$$

Siendo t la iteración actual y λ un parámetro de estabilidad. El símbolo \bigoplus se refiere a una operación de multiplicación entrada por entrada, lo que significa que cada entrada de dos matrices (o vectores) se multiplica una con la otra, resultando en una nueva matriz (o vector) con las mismas dimensiones [39].

2. **Selección de nidos:** Los nidos con huevos de alta calidad se pasan a las generaciones siguientes. Si $f(x_j) > f(x_i)$, entonces $x_i = x_j$.
3. **Descubrimiento de huevos:** Los huevos son descubiertos por aves anfitrionas con una probabilidad p_a . Probabilidad de descubrimiento: $p_a \in [0, 1]$

Cuando un huevo es descubierto, el ave anfitriona puede desecharlo o abandonar el nido y construir uno nuevo $\rightarrow rand() < p_a$. El objetivo es usar las nuevas soluciones (huevos de cuco) para reemplazar soluciones no tan buenas en los nidos [39].

Capítulo 6

Diseño experimental

En este capítulo se describirá el proceso de experimentación llevado a cabo, los conjuntos de datos usados para los distintos experimentos y los parámetros usados en cada uno de los algoritmos.

Los conjuntos de datos usados son lo siguientes: Se han seleccionado los

Dataset	Instancias	Características	Clases	Área
sonar	207	60	2	Biología
spambase-460	459	54	2	Informática
spectf-heart	348	44	2	Medicina
waveform5000	5000	40	3	Física
ionosphere	350	34	2	Meteorología
dermatology	366	34	6	Medicina
zoo	101	18	7	Biología
parkinsons	200	22	2	Medicina
wdbc	568	29	2	Medicina
wine	182	13	3	Alimentación
breast-cancer	286	9	2	Medicina
diabetes	768	8	2	Medicina
ecoli	336	7	8	Biología
yeast	1483	8	10	Biología
iris	149	4	3	Biología

Tabla 6.1: Información de los conjuntos de datos ordenada por número de características

conjuntos de datos descritos en la tabla 6.1 por los siguientes motivos:

- **Variedad de áreas:** Se ha buscado representar una amplia gama de áreas, como Medicina, Biología, Alimentación, Meteorología, Física e Informática. Esto asegura que los problemas abordados sean variados

y abarquen diferentes dominios de aplicación.

- **Diversidad de problemas:** Los conjuntos de datos elegidos cubren una variedad de problemas, desde diagnósticos médicos y clasificación de especies hasta predicciones en meteorología y detección de spam. Esta diversidad permite evaluar el rendimiento de los algoritmos en múltiples contextos.
- **Número de características:** Algunos conjuntos de datos, como *spectf-heart* y *spambase-460*, tienen un gran número de características, lo que es particularmente útil para evaluar la eficacia de las técnicas de selección de características. Esta variedad en el número de características permite probar la robustez de los algoritmos en situaciones de alta dimensionalidad. La inclusión de otros como *iris* con tan pocas características hacen de conjunto de datos de referencia, todos los algoritmos deberían optimizar este *dataset*.
- **Popularidad y uso en investigaciones:** Los conjuntos de datos seleccionados son populares y ampliamente utilizados en estudios y artículos de referencia [53, 57, 62, 96]. Esto facilita la comparación de resultados con otros trabajos y asegura que los datos son reconocidos y aceptados en la comunidad científica.
- **Variedad en el número de instancias y clases:** La selección incluye conjuntos de datos con diferentes números de instancias y clases. Esta variedad es crucial para probar la escalabilidad y versatilidad de los algoritmos.
- **Relevancia práctica:** Muchos de estos conjuntos de datos, como *diabetes* y *breast-cancer*, tienen una alta relevancia práctica en sus respectivas áreas, lo que subraya la importancia de encontrar soluciones eficientes y precisas para problemas del mundo real.

Los datos han sido normalizados entre 0 y 1 siguiendo la función de normalización *MinMaxScaler* de *scikit-learn* [97]. Los conjuntos de datos se han dividido en 20 % test, 20 % validación y 60 % entrenamiento. Se han llevado a cabo 10 ejecuciones sobre cada algoritmo en cada conjunto de datos, tanto en su versión original como en su versión binaria. Estas ejecuciones se han procesado de forma paralela en el servidor de *Hércules* [8].

Se han recabado datos de todas las ejecuciones para poder comparar los resultados. En forma de gráficos se ha representado la curva de convergencia de cada algoritmo además de una curva de decrecimiento de las características de cada conjunto de datos. Ambos gráficos son muy interesantes a la hora de comparar algoritmos para un mismo conjunto de datos.

Además se han recabado los siguientes datos en ficheros de texto para su posterior análisis:

- **classifier**: Clasificador usado (kNN o SVC).
- **dataset**: El conjunto de datos utilizado para el proceso de selección de características.
- **optimizer**: Algoritmo metaheurístico usado como optimizador.
- **all_fitness**: Lista que contiene los valores *fitness* de todas las soluciones generadas durante la optimización.
- **best**: Mejor valor encontrado durante la optimización.
- **avg**: Promedio de los valores de *fitness* de todas las soluciones generadas durante la optimización.
- **std_dev**: Desviación estándar de los valores de *fitness* de todas las soluciones generadas durante la optimización. Indica qué tan dispersos están los valores alrededor del promedio.
- **acc**: Precisión (accuracy) del modelo de clasificación, es decir, la proporción de instancias correctamente clasificadas sobre el total de instancias.
- **n_features**: Número de características seleccionadas.
- **selected_rate**: Tasa de selección de características, que representa la proporción de características seleccionadas respecto al total de características disponibles.
- **execution_time**: Tiempo de ejecución del algoritmo optimizatorio.

Para la función de *fitness* se ha usado:

$$\text{fitness} = \text{acc} \cdot \alpha + \text{red} \cdot (1 - \alpha) \quad (6.1)$$

Donde *acc* quiere decir precisión y *red* el porcentaje de características reducidas. La variable α actúa como medio de ponderación y ha sido fijado en $\alpha = 0.9$, dando un 90 % de prioridad a la precisión. Se procede a mostrar los parámetros fijados para cada algoritmo (como referencia se han seguido los parámetros de los artículos originales y binarios ya citados para cada algoritmo).

Los parámetros fijados en cada uno de los algoritmos de optimización metaheurísticos han sido elegidos por las recomendaciones dadas en los artículos de referencia de los que se sacan, más concretamente los artículos que presentan la versión binaria. Estos son citados en la descripción de cada tabla.

Algoritmo	Parámetros
GOA [62]	$c_{min}: 0.00001$ $c_{max}: 1$ $F: 0.5$ $L: 1.5$
WOA [98]	Parámetro espiral: 1
ABCO [45]	Abeja empleada: 3 Abeja vigilante: 3 Límite: 3
BA [65]	$\alpha: 0.9$ $\gamma: 0.9$ $f_{min}: 0$ $f_{max}: 2$
PSO [73]	$w: 0.9$ $c_1: 2$ $c_2: 2$
FA [69]	$\alpha_0: 0.5$ $\beta_0: 0.2$ $\gamma_0: 1$
GA[86]	Ratio de cruce: 1 Ratio de mutación: 0.05 Elite: 2 $\eta: 1$ $\alpha: \sqrt{0.3}$
ACO [92]	$\alpha: 1$ Q: 1 Feromona inicial: 0.1 Ratio de evaporación: 0.049
CS [68]	Ratio de descubrimiento: 0.25 $\alpha: 1$ $\lambda: 1.5$
DE [44]	$F: 0.5$ $Cr: 0.1$

Tabla 6.2: Parámetros de diferentes algoritmos de optimización

Dada la cantidad de variantes y cantidad de valores de parámetros propuestos en **GA**, estos han sido escogidos de manera empírica y basándose en recomendaciones del tutor del proyecto. En **WOA**, en el paper original [37] y en el de la versión binaria [57], no se da un valor. El valor por defecto ha sido sacado de la implementación de [98]. Los algoritmos de **DA** y **GWO** no tienen parámetros configurables.

Cada destacar que el algoritmo **ACO** no se ha añadido al grupo de experimentación de los algoritmos continuos, debido a la naturaleza de su diseño, pues se considera que las versiones existentes en codificación real perturban demasiado el diseño de este, además de que por su propia inspiración no tendría sentido su modificación.

Por último, cabe mencionar que se ha introducido un límite de corte en la selección de características. De este método se benefician sobre todo los algoritmos continuos, debido a que estos ajustan los pesos en el rango de valores de $[0, 1]$, pero no reducen completamente como los binarios. De esta forma, toda característica que tenga un peso asociado menor a 0.05 se considera suficientemente irrelevante como para descartarla.

Para evaluar el rendimiento de los algoritmos, se plantean las siguientes hipótesis:

Hipótesis nula (H_0):

$$H_0 : \text{Rendimiento}_{Alg_1} \leq \text{Rendimiento}_{Alg_2} \quad (6.2)$$

Hipótesis alternativa (H_1):

$$H_1 : \text{Rendimiento}_{Alg_1} > \text{Rendimiento}_{Alg_2} \quad (6.3)$$

Se va a usar el test de *Wilcoxon* [99] junto a un *post hoc* de *Holms* [100]. El test de *Wilcoxon* es una prueba no paramétrica que se utiliza para comparar dos conjuntos de datos emparejados. Este test es útil cuando se quiere determinar si existe una diferencia significativa entre dos condiciones (por ejemplo, el rendimiento de dos algoritmos) sin asumir que los datos siguen una distribución normal. El procedimiento *post hoc* de *Holm* es una técnica utilizada para controlar la tasa de error tipo en múltiples comparaciones. En estudios donde se realizan múltiples pruebas estadísticas (como es el caso), la probabilidad de obtener resultados falsos positivos aumenta y por ello se necesite de este corrector a la hora de realizar afirmaciones sobre los tests.

Capítulo 7

Análisis y resultados

En este capítulo se van a discutir los resultados obtenidos tras realizar los experimentos descritos en el capítulo anterior. Se mostrarán gráficas que ilustren la curva de convergencia de los algoritmos en algunos conjunto de datos, para cada clasificador (kNN y SVC) y en cada versión del algoritmo, es decir, original y versión binaria. Junto a eso también se han elaborado una gráficas de “cajas y bigotes” o *boxplots* que ilustran no solo los mejores resultados, sino su robustez y estabilidad tras varias iteraciones.

Además se mostrarán tablas y gráficas de los ranking obtenidos en versión binaria y en versión continua para cada clasificador tras hacer la media de los resultados en todos los conjuntos de datos.

Por último, los resultados vendrán expuestos en forma de tabla junto a las métricas correspondientes, de forma que se facilite el análisis de los algoritmos, de las versiones reales y continuas y de los clasificadores usados.

También se realizarán análisis estadísticos de los resultados obtenidos utilizando como herramienta la página web **TACOlab** [101], facilitada por el tutor del proyecto, de forma que se puedan hacer los mejores y más robustos tests de forma automática y sencilla.

Como detalle a tener en cuenta, se ha añadido en las comparaciones un optimizador al que se le ha llamado “Dummy”, debido a su naturaleza totalmente aleatorio y sin criterio alguno. Se añade como algoritmo de referencia, ninguna otra metaheurística debería ser peor que esta en promedio. En caso contrario podría decirse que el algoritmo en cuestión está mal diseñado.

Pese a que se dividirán los análisis entre binario y continuo, las versiones discretas se escribirán con la letra *b* delante. Por ejemplo, el algoritmo **FA** en binario sería **bFA**.

7.1. Binario

En esta sección se discutirán y analizarán los resultados para las metaheurísticas de codificación binaria o discreta. Estas versiones tienen un enfoque del problema distinto al de las reales, pues codifican cada característica con un 0 o un 1, o lo que es lo mismo, descartan características. Dada su naturaleza, cabría pensar que son los algoritmos más específicos que podrían usarse para el problema se selección de características.

7.1.1. General

En esta sub-sección se presentarán los resultados generales obtenidos con las metaheurísticas binarias. Esto permitirá una visión más amplia de su desempeño y proporcionará intuiciones iniciales sobre la efectividad de cada algoritmo en distintos contextos.

Como ya se aclaró en capítulos anteriores, para obtener estos resultados se han ejecutado 10 veces cada algoritmo en cada conjunto de datos. Se muestran los resultados generales de rankings en para **SVC** y **kNN** en 7.1a y 7.1b, respectivamente.

Como puede apreciarse en los rankings mencionados, los algoritmos con mejor rendimiento son **bGWO**, **bPSO**, **bCS** y **GA**. En concreto, **bGWO** obtiene una diferencia muy notable con respecto al resto de los algoritmos.

Los peores son **bABCO**, **bGOA** y **bDA**. Además lo son con una puntuación que los diferencia mucho del los algoritmos con un rendimiento medio, sobre todo **bABCO**.

También es apreciable que los mejores algoritmos producen mejores resultados de media en **SVC** que en **kNN**, aunque los resultados de todo el conjunto de algoritmos parecen más estables en **kNN**.

7.1.2. Variabilidad

Se muestran algunos de los resultados obtenidos con los algoritmos binarios en 7.2a, 7.2b, 7.2c y 7.2d con el clasificador **kNN**. Obviamente, no es posible incluir gráficas para todas las combinaciones posibles sin saturar el capítulo; para ello, se remite al lector al capítulo de apéndices.

Se puede observar que la mayoría de los algoritmos obtienen resultados bastante buenos en los distintos conjuntos de datos, considerándose los valores de fitness suficientemente buenos para considerarlos soluciones de calidad, obviamente considerando cada problema por separado, con las características de cada uno. Es importante destacar que no todos los conjuntos de datos presentan la misma dificultad, ya que distintos tipos de problemas presentan

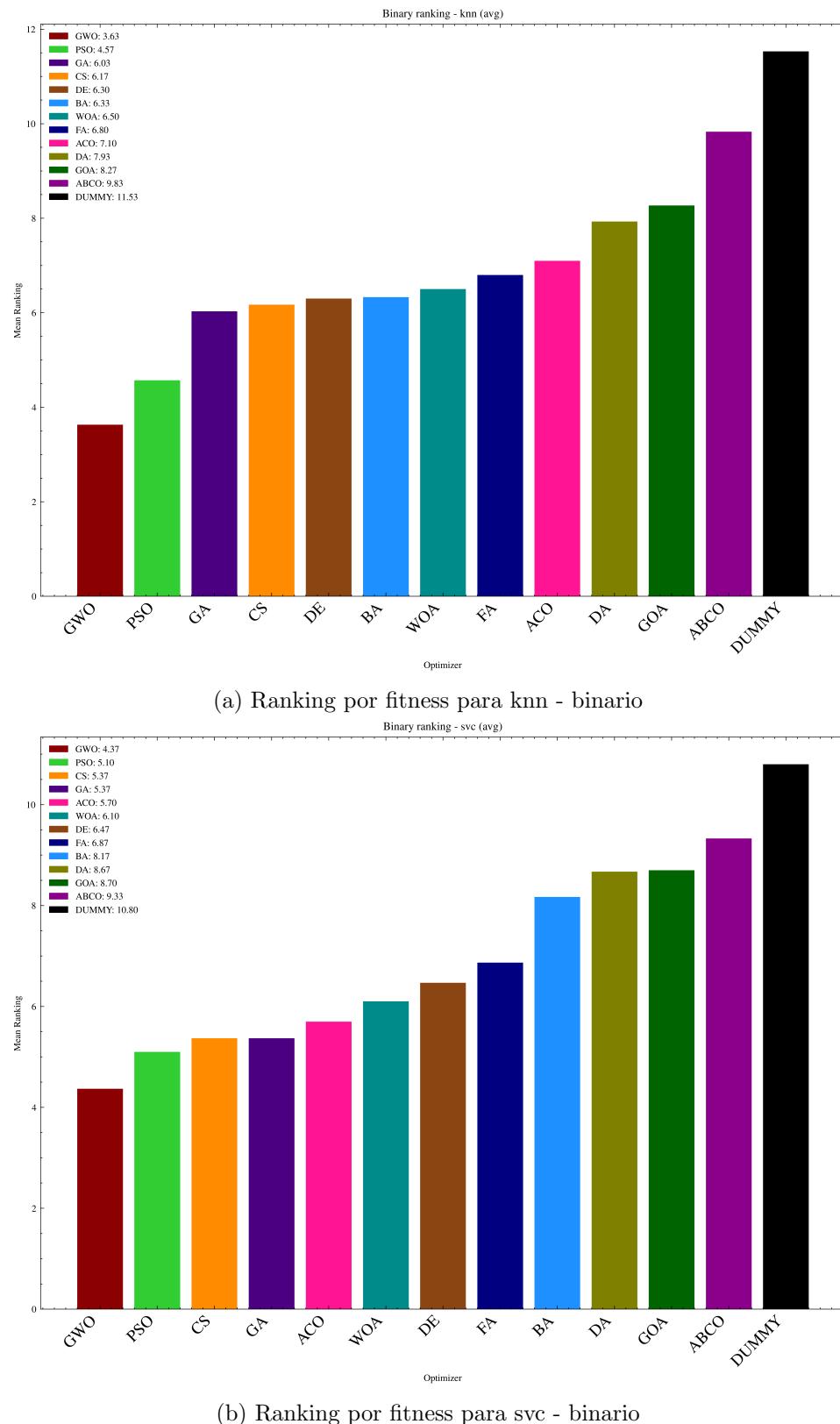
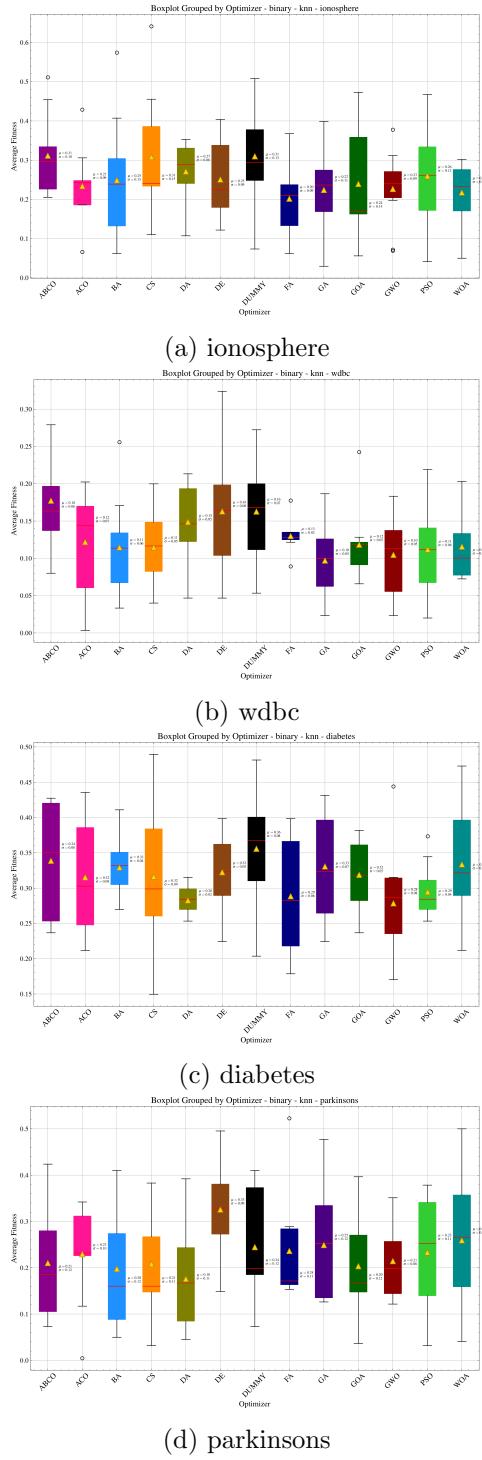


Figura 7.1: Rankings para fitness - binario

Figura 7.2: *Boxplots* para kNN - binario

diferentes desafíos. En particular, se puede notar que los algoritmos aplicados al conjunto de datos *ionosphere* (figura 7.2a) muestran menos variabilidad en sus valores de fitness, reflejándose en una desviación estándar menor en comparación con los resultados de *diabetes* (figura 7.2c) o *parkinsons* (figura 7.2d). Con ello, queda reflejado como a los algoritmos les cuesta menos optimizar ciertos problemas y suelen ser más estables en estos. Esto puede deberse a muchos factores, por ejemplo, mayor cantidad de mínimos locales en algunos problemas en comparación a otros.

También se pueden identificar ciertos algoritmos que parecen tener un mejor desempeño en general. Por ejemplo, algoritmos como **bGWO** y **bPSO** tienden a obtener resultados muy buenos y poco variantes, mientras que otros como **ACO** y **bABCO** parecen ser menos efectivos en los conjuntos de datos analizados.

Pese a ello, se percibe una variabilidad de los resultados en *fitness* bastante similar en todos los algoritmos usados. Donde un algoritmo es robusto en cuanto a variabilidad, en otro no lo es. Esto se observa en todos los problemas.

7.1.3. Fitness

Se procede a comparar y analizar los distintos algoritmos basándonos en la métrica de *fitness*. Como se explica en la ecuación 6.1, este valor está compuesto por otras dos métricas a tener en cuenta, la precisión o *accuracy* y el ratio de reducción de características. La métrica de *fitness* es el marcador más importante para comprobar la calidad de un algoritmo. Pese a ello, más tarde se analizarán ambas métricas que lo componen por separado, pues es interesante obtener una visión más precisa de los resultados.

Debe tenerse en cuenta que, al estar compuesto el *fitness* en un 90% por el *accuracy*, ambas métricas estarán altamente relacionadas entre sí en la gran mayoría de los casos.

Como puede observarse en la tabla ??, los algoritmos que consiguen mejores resultados son **ACO**, **bGWO**, **bPSO**, **bFA**, **bDE**, **bGA**, **bCS** y **bWOA** con el clasificador **SVC**. Los más destacables en cuanto al número de veces que estos se repiten por ser los mejores por *dataset* son **bCS**, **ACO**, **bDE** y **bGWO** 7.1. Puede verse que en **SVC** los resultados son más discriminativos y se reparten menos los mejores resultados entre el conjunto completo de algoritmos.

Mejor vs Peor

Como se puede observar en 7.1a y en 7.1b, el algoritmo con mejor puntuación es **bGWO** y el peor **bABCO** (se puede ver también su valor en los rankings

Algoritmo	N veces mejor (knn)	N veces mejor (svc)
bCS	2	4
ACO	1	3
bDE	2	2
bGWO	3	2

Tabla 7.1: Algoritmos con más veces mejor resultados - knn y svc - binario

de las tablas en el apéndice: A.4, A.3).

En las figuras de A.58 y A.57 se aprecia como **bABCO** muestra rendimientos muy buenos en seg\xfbin que conjunto de datos. Por ejemplo, en *yeast* o en *sonar*, el algoritmo obtiene resultados muy decentes y competitivos siendo estos dos conjuntos de datos de los m\xfas complicados de la selecci\xf3n, por convergencia y n\xfamero de caracter\xedsticas. Pese a ello, en conjunto, no consigue superar al resto.

En cambio **bGWO** parece un adaptarse bien a todos los problemas seleccionados sin excepci\xf3n. Es capaz de obtener el m\xf3nimo, en el valor *fitness* de evaluaci\xf3n durante el entrenamiento, en casi todos los *datasets*.

	Original	Holm	Hommel	Hochberg
dummy	6.104E-05	0.001	0.001	0.001
abco	0.003	0.029	0.029	0.029
da	0.005	0.054	0.051	0.054
goa	0.010	0.092	0.082	0.092
fa	0.022	0.172	0.106	0.172
de	0.026	0.179	0.106	0.179
ga	0.038	0.230	0.127	0.191
ba	0.055	0.277	0.166	0.191
aco	0.055	0.277	0.166	0.191
woa	0.064	0.277	0.191	0.191
cs	0.135	0.277	0.271	0.271
pso	0.346	0.346	0.346	0.346

Tabla 7.2: P-valores del bGWO vs el resto - knn - binario

Se procede a hacer una comparativa estad\xedstica mediante la prueba de *Wilcoxon* y el test post hoc de *Holm* para determinar las diferencias significativas entre el mejor algoritmo (**bGWO**) y el resto. Los resultados, presentados en las tablas 7.2 y 7.3, muestran los p-valores originales y ajustados, lo que nos permite identificar cu\xe1les de las diferencias observadas son estad\xedsticamente significativas con un error corregido. Este an\xe1lisis es crucial para validar

	Original	Holm	Hommel	Hochberg
dummy	3.052E-04	0.004	0.004	0.004
abco	0.002	0.024	0.022	0.024
ba	0.004	0.043	0.037	0.043
da	0.005	0.048	0.043	0.048
goa	0.008	0.066	0.066	0.066
woa	0.073	0.514	0.367	0.514
de	0.109	0.656	0.547	0.599
fa	0.121	0.656	0.599	0.599
ga	0.252	1.000	0.599	0.599
cs	0.364	1.000	0.599	0.599
aco	0.490	1.000	0.599	0.599
pso	0.599	1.000	0.599	0.599

Tabla 7.3: P-valores del bGWO vs el resto - svc - binario

los resultados y asegurar que las conclusiones obtenidas no sean producto del azar, proporcionando una base sólida para afirmar la superioridad de **bGWO** frente a los demás algoritmos en los conjuntos de datos evaluados, en frente al peor de ellos.

Si bien no se observan diferencias significativas sobre la mayoría de algoritmos, sí sobre el peor (**bABCO**) usando **kNN**. Tanto en la versión que utiliza el clasificador **SVC** como en **kNN** se obtienen resultados suficientemente significativos como para poder considerar que los resultados de **bGWO** frente a los de **bABCO** no son producto del azar. Por tanto, utilizando un valor crítico de $\alpha = 0.05$ se puede rechazar la hipótesis nula. Esto no solo ocurre con **bABCO** en los resultados de **SVC**, sino que también con **bBA** y **bDA**. Por ello se puede afirmar que **bGWO** en este último clasificador es mejor, con un respaldo estadístico, frente a esos algoritmos también.

Clásicos vs Modernos

Los algoritmos clásicos de optimización como el **bPSO**, **bGA** o **bDE** han demostrado ser extremadamente efectivos en una amplia gama de problemas. Su capacidad para explorar y explotar el espacio de búsqueda ha llevado a soluciones de alta calidad en diversas aplicaciones, desde la ingeniería hasta la economía. Estos métodos son conocidos por su robustez, simplicidad y eficacia, lo que los convierte en herramientas valiosas.

Sin embargo, en los últimos años, han surgido nuevos algoritmos de optimización que prometen mejorar aún más estos resultados. Algoritmos como el

bWOA, **bGWO** y **bCS**, aunque este último es de hecho bastante antiguo, han sido diseñados para abordar algunas de las limitaciones de los métodos clásicos, como la convergencia prematura y la exploración insuficiente del espacio de búsqueda. Estos algoritmos modernos incorporan nuevas estrategias y mecanismos de búsqueda inspirados en la naturaleza, que buscan mejorar la eficiencia y la precisión de las soluciones obtenidas.

Los rankings han sido creados basándose en los valores *fitness* de los algoritmos para todos los conjuntos de datos. De forma visual es capaz de verse de un vistazo aquellos más versátiles, los que mejor se han comportado en distintos escenarios. En los rankings que se pueden observar en las gráficas 7.1a para **kNN** y en 7.1b para **SVC** puede verse como el mejor algoritmo es el **bGWO**, el algoritmo de los lobos grises, considerado como moderno. Le siguen algoritmos como **bPSO**, **bCS** y **bGA**.

El **bPSO**, a pesar de ser uno de los algoritmos más antiguos en la optimización metaheurística, sigue manteniendo una posición destacada en los rankings. Estos hechos junto a su simplicidad y efectividad en la convergencia hacia soluciones óptimas, hacen ver que **bPSO** sigue siendo una opción muy interesante, incluso en problemas de tipo binario como la selección de características, escenario para los que **bPSO** no fue diseñado. Esto hace de esta metaheurística una opción que parece sugerir mucha robustez.

El **bCS**, basado en el comportamiento de anidación de ciertos pájaros, también se muestra como una opción fuerte, compitiendo por el podio con **bGA**.

Finalmente, el **bGA**, uno de los algoritmos más estudiados y aplicados en el ámbito de la optimización, sigue siendo relevante. Su capacidad para encontrar soluciones óptimas a través de mecanismos inspirados en la evolución natural, como la selección, el cruce y la mutación, le permite abordar una amplia gama de problemas con éxito. Además, este algoritmo al ser inicialmente pensado para aplicarse en dominios de codificación discreta, hace de esta metaheurística una opción muy interesante y robusta en el problema de selección de características, además de llevar años siendo aplicado con éxito en otros dominios.

Es visible que algunas de estas nuevas propuestas son muy potentes, llegando a mejorar en la propuesta binaria en comparación a algoritmos clásicos que llevan tiempo en la escena como las opciones *de facto*. Pese a ello, el

grueso de los algoritmos comparados son opciones modernas, y muchos de estos algoritmos clásicos obtienen mejores resultados que la mayoría de este reciente grupo. De los cinco mejores, tres de ellos son clásicos, lo cual es más impresionante si se considera que hay un desbalance en número de *clásicos vs modernos*. **bDE** es otro algoritmo asentado, que es capaz de clasificar en posiciones medias de *fitness*.

Por tanto, los clásicos siguen siendo una opción muy potente e interesante. Pese a desarrollarse nuevos algoritmos, estos siguen obteniendo soluciones de alta calidad.

7.1.4. Clasificación y reducción

Como se ha mencionado al principio de la anterior sección y como se explica en la fórmula 6.1, el *fitness* está compuesto por las métricas de *accuracy* en la clasificación y por el ratio de reducción de las características. Con ello se intenta reducir tantas características como sea posible, manteniendo un valor lo más alto posible de precisión.

Con estos aspectos aclarados, en esta sección se analizarán los algoritmos desde la perspectiva individual de cada una de las métricas que componen la función *fitness*. De esta manera, se proveerá información de cuál o cuáles algoritmos son los mejores para cada una de las métricas.

Esta visión es interesante ya que, los pesos fijados por el autor para cada métrica son los estándar en la mayoría de artículos, pero es posible que en otros problemas sea necesario ajustarlos a las necesidades del mismo. Por tanto si se conocen las características de los diferentes algoritmos para cada métrica por separado, se obtendrá un conocimiento que proporcionará un uso de los mismos más flexible.

Mejor vs Resto

La metaheurística **ACO** es sin lugar a dudas la mejor en reducción de características (A.13, A.14) y a su vez, la peor en precisión según los rankings (A.9, A.10). Un ejemplo de esto es en el conjunto de datos de *ionosphere*. Se puede observar como el segundo algoritmo que más reduce, que es **bGWO**, obtiene una reducción media del 74 %, mientras que el **ACO** obtiene una reducción media del 91.5 %, con tan solo una diferencia de 4.6 % en precisión a favor de **bGWO** A.15.

Si bien es cierto que su precisión no es de las mejores, podría considerarse aceptable en muchos casos. Teniendo ese aspecto en cuenta, añadido a que es capaz de reducir características a un nivel muy significativo con respecto

al resto de algoritmos, **ACO** se presenta como un algoritmo muy interesante para aquellos problemas en los que se pueda permitir cierto nivel de error en la precisión, pero en cambio, sea crucial una compresión del tamaño del modelo o un tiempo den entrenamiento corto.

Los mejores algoritmos en *accuracy* son **bCS** y **bGWO** como puede observarse en las tablas A.9 y A.10, mientras que el peor, como se ha comentado anteriormente, es el **ACO**. Si bien es cierto que **bGWO** empata en **kNN** con **bBA**, este no se ha considerado como mejor por obtener un resultado peor en **SVC** mientras que **GWO** es bueno en ambos. Pese a ello, no hay grandes diferencias en los algoritmos del top.

Si comparamos los p-valores 7.4, 7.5, tras aplicar post hoc para mayor seguridad, de **ACO** con respecto al resto de algoritmos, se ve que claramente la mejora en reducción no se debe al azar de manera muy probable en todos los casos comparativos.

	Original	Holm	Hommel	Hochberg
dummy	6.104E-05	0.001	0.001	0.001
fa	6.104E-05	0.001	0.001	0.001
goa	6.104E-05	0.001	0.001	0.001
abco	0.001	0.010	0.002	0.002
ba	0.001	0.010	0.002	0.002
cs	0.001	0.010	0.002	0.002
da	0.001	0.010	0.002	0.002
de	0.001	0.010	0.002	0.002
ga	0.001	0.010	0.002	0.002
pso	0.001	0.010	0.002	0.002
woa	0.001	0.010	0.002	0.002
gwo	0.002	0.010	0.002	0.002

Tabla 7.4: P-valores de ACO vs el resto - knn - binario

En cuanto a la clasificación, no se encuentran diferencias significativas entre los algoritmos, a excepción del **bCS** contra el **ACO**, el cual parece ser significativamente peor.

Clásicos vs Modernos

Se procede a comparar los algoritmos clásicos frente a los algoritmos modernos en cuanto a su capacidad clasificatoria, es decir, el valor de cada uno de

	Original	Holm	Hommel	Hochberg
dummy	6.104E-05	0.001	0.001	0.001
fa	6.104E-05	0.001	0.001	0.001
goa	6.104E-05	0.001	0.001	0.001
abco	0.001	0.010	0.001	0.001
ba	0.001	0.010	0.001	0.001
cs	0.001	0.010	0.001	0.001
da	0.001	0.010	0.001	0.001
de	0.001	0.010	0.001	0.001
ga	0.001	0.010	0.001	0.001
pso	0.001	0.010	0.001	0.001
woa	0.001	0.010	0.001	0.001
gwo	0.001	0.010	0.001	0.001

Tabla 7.5: P-valores de ACO vs el resto - svc - binario

ellos obtenido en *accuracy*.

En los rankings definidos en la tabla A.9 y en A.10, el top de los mejores algoritmos está compuesto por los algoritmos **CS**, **FA**, **GWO**, **DE**, y **PSO** en **SVC**, y los algoritmos **WOA**, **BA**, **GWO**, **PSO**, y **FA** en **kNN**. El top de los peores algoritmos está compuesto, en cambio, por **GOA**, **ACO**, **DA**, y **GA** en **SVC**, y los algoritmos **ACO**, **DA**, **GOA**, y **GA** en **kNN**.

Si bien es cierto que la mayoría en el top de mejores algoritmos en *accuracy* son algoritmos modernos, los clásicos no se quedan atrás. En las comparaciones obtienen muy buenos resultados, no siendo los mejores y muy lejos de ser los peores. En los algoritmos modernos hay mayor diversidad en este punto. Hay unos pocos que destacan, pero el resto parecen verse superados en muchas ocasiones por los algoritmos clásicos.

En cuanto a la reducción de características, se observa en los rankings 7.3 como los mejores son algoritmos como **ACO**, **bGWO**, **GA** y **bPSO**. Incluso **bDE** es capaz de superar a varios modernos. El **bABCO**, sin embargo, parece obtener resultados pésimos, quedando el último en los rankings.

Si se observan los p-valores obtenidos en las tablas A.11 y A.12, y se agrupan los modernos y los clásicos, se obtienen los siguientes resultados:

- **bPSO**: A excepción del **ACO** (el mejor), **bGWO** (segundo mejor) y **bGA**, es superior a todo el resto de algoritmos con una diferencia

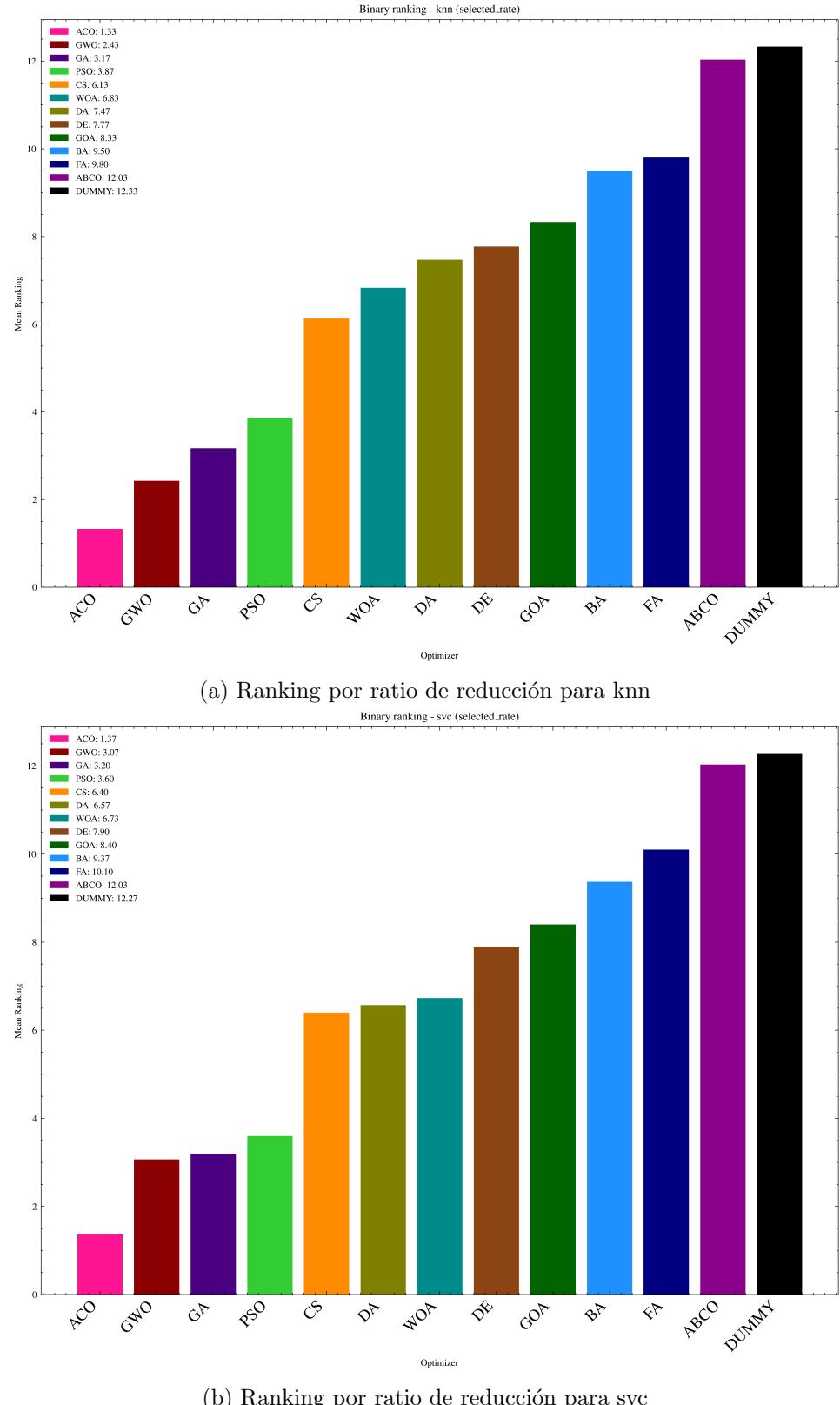


Figura 7.3: Rankings para reducción de características

significativa más que suficiente tras corrección post hoc con el método **Holms**.

- **ACO:** Es el mejor con diferencia significativa más que suficiente como para poder afirmar que este algoritmo es muy destacable en la reducción de características.
- **bGA:** Ocurre lo mismo que con **bPSO**. Pese a excepciones de algoritmos por encima suyo en el ranking, **bGA** es muy superior al resto de algoritmos, con diferencias muy notables y de nuevo significativas.
- **bDE:** Obtiene resultados muy buenos, pero en la media en comparación con todos.
- **bABCO:** El peor parado del grupo de los algoritmos clásicos. No parece ser una buena propuesta a la hora de reducir.

Dados estos resultados, los algoritmos clásicos salen muy bien parados en la reducción de características frente a los modernos. Si bien es cierto que hay dos algoritmos que destacan sobre el resto de los modernos (**bCS** y **bGWO**) en la reducción de características, los clásicos en conjunto, con excepción de **bABCO**, parecen una opción robusta y que, en combinación a sus resultados de precisión (*accuracy*), son muy precisos a la hora de seleccionar características importantes.

Por lo explicado, combinando los resultados analizados en *accuracy* y en selección de características, se puede intuir que los algoritmos clásicos son muy eficientes en la reducción de características, superando a varios algoritmos modernos y además con una calidad en términos de *accuracy* prácticamente idéntica.

bWOA vs bGWO

Dentro del conjunto de algoritmos analizados, hay ciertos pares que resultan interesantes por la semejanza algorítmica e inspirativa entre ambos. En esta sección, se compararán los algoritmos **bWOA** y **bGWO**. Ambos pertenecen a la categoría de algoritmos de optimización inspirados en la naturaleza y ambos se inspiran fuertemente en el comportamiento social y estrategias de caza de ciertos animales.

Dentro del contexto de la caza, tanto **WOA** como **GWO** utilizan ecuaciones que simulan el rodeo de la presa, aunque con diferencias en la implementación y el comportamiento específico. A continuación se detallan los comportamientos de rodeo de la presa en la caza:

- **bWOA:** La ballena ajusta su posición no solo moviéndose hacia la presa sino también rodeándola en un patrón circular, lo cual le permite cubrir más área y ajustar su aproximación con mayor precisión. La espiral seguida en la fórmula depende del parámetro de control D .
- **bGWO:** Los lobos rodean a la presa en grupos liderados por alfa, beta y delta. Al reducir la distancia a estas posiciones líderes, los lobos de menor rango simulan el cerco y convergen hacia la presa de manera cooperativa. El acercamiento utiliza un vector de coeficientes A que decrece linealmente.

Ambos algoritmos por tanto, y pese a sus diferencias, comparten una naturaleza y comportamiento parecido a la hora buscar soluciones. Al comparar ambos resultados en *accuracy*, se aprecia que estos no difieren de manera suficiente en su rendimiento como para poder rechazar la hipótesis nula. Véase en las tablas A.7, A.8.

Es en cambio en la comparación en la métrica de reducción de características se aprecia que **bGWO** presenta una diferencia notable con respecto a **bWOA**. Puede observarse gráficamente en las figuras 7.4 como **bWOA** reduce considerablemente menos que **bGWO**. Comprobarse también que las diferencias entre ambos son estadísticamente significativas 7.6, 7.7

	gwo	woa
gwo	-	- (0.001)
woa	+ (0.001)	-

Tabla 7.6: P-valores tras corrección entre bGWO y bWOA - knn

	gwo	woa
gwo	-	- (0.002)
woa	+ (0.002)	-

Tabla 7.7: P-valores tras corrección entre bGWO y bWOA - svc

Por tanto, pese a no haber una diferencia estadística suficientemente fuerte entre los resultados de precisión de ambos algoritmos, si la hay en la reducción de características. Por ello se puede afirmar que **bGWO** es más interesante que **bWOA**.

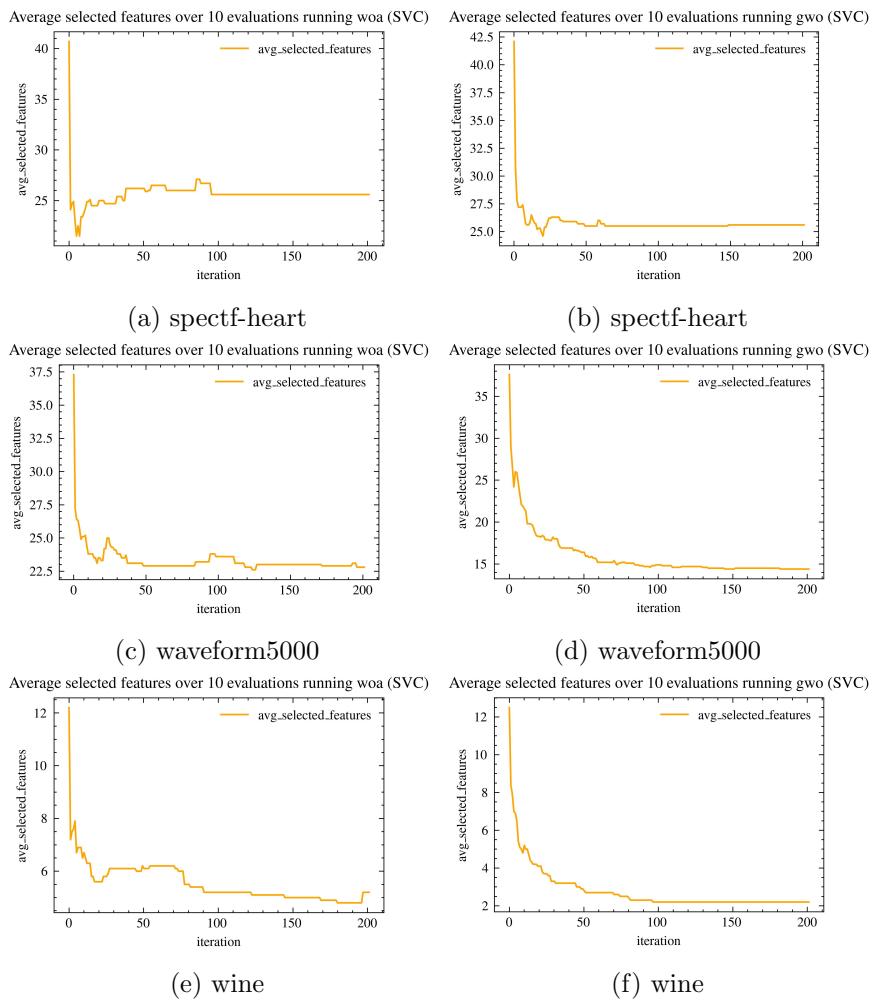


Figura 7.4: Reducción de características en bWOA y bGWO

7.1.5. Tiempo de ejecución

Una última métrica muy importante a analizar es el tiempo de ejecución. Por definición, la mayoría de metaheurísticas deberían ser rápidas, al menos en comparación a algoritmos exactos, pues es en sustitución de estos donde brillan realmente. En el caso concreto del problema de selección de características es algo a tener muy en cuenta, pues unos de los cuellos de botella principales en el entrenamiento de grandes modelos es el tiempo de ejecución. Por tanto, en esta sección se procederá a analizar varios algoritmos y sus diferencias en cuanto a tiempos.

Los resultados arrojan dos algoritmos con diferencias relevantes, **bFA** con una diferencia positiva y **bABCO** con una diferencia negativa. Véase en la table A.16. El resto de algoritmo obtienen tiempos muy similares en todos los conjuntos de datos.

7.1.6. Convergencia

En esta última sub-sección se analizará la convergencia de los diferentes algoritmos. Inicialmente, si se observan las gráficas de convergencia de cada *dataset* A.57, A.58, A.59, A.60 y se analizan las curvas, se puede decir que absolutamente todos los algoritmos convergen, ya sea a una solución mejor o peor, pero todos lo hacen (incluso **Dummy**). No hay ninguno en ningún problema que retroceda en cuanto a su valor *fitness* y todos poseen una curva descendiente bastante suave. Con el tiempo, esta curva tiende a alcanzar un límite inferior, como si de una función logarítmica se tratase. Esto indica, que el algoritmo está mejorando cada vez menos porque ha encontrado un óptimo local.

Claramente, el peor algoritmo y al que más le cuesta converger es **ACO**. Pese a ello, **ACO** parece variar mucho en su proceso optimizatorio entre distintos conjuntos de datos.

Otro algoritmo que converge más lentamente que el resto es el **bDA** y el **bGOA**. Por ejemplo, en el problema de *diabetes*, representado en la figura A.57c, se puede apreciar que el **bDA** es mejor que **bGOA** y que ambos son los más lentos a excepción del **bDummy**.

Pese a las dificultades de algunos algoritmos, todos convergen correctamente en todos los conjuntos de datos, lo cual es esperable.

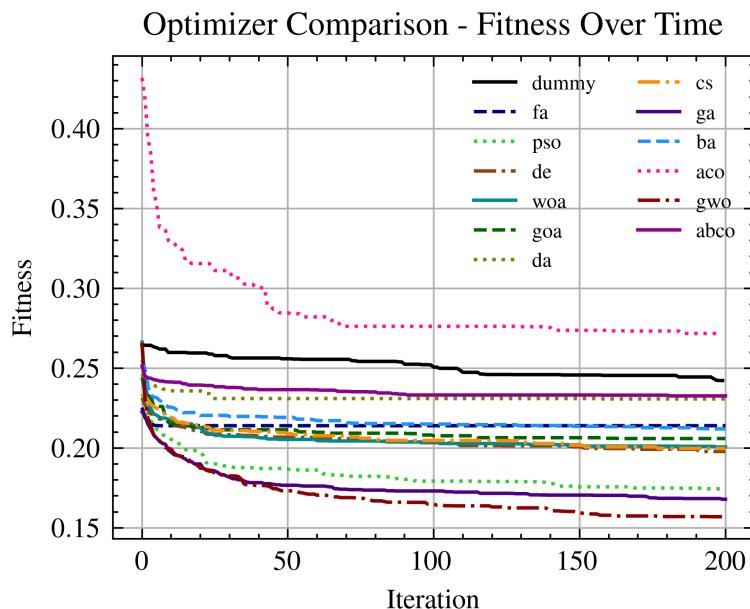


Figura 7.5: Convergencia de todas las metaheurísticas en waveform5000 - knn - binario

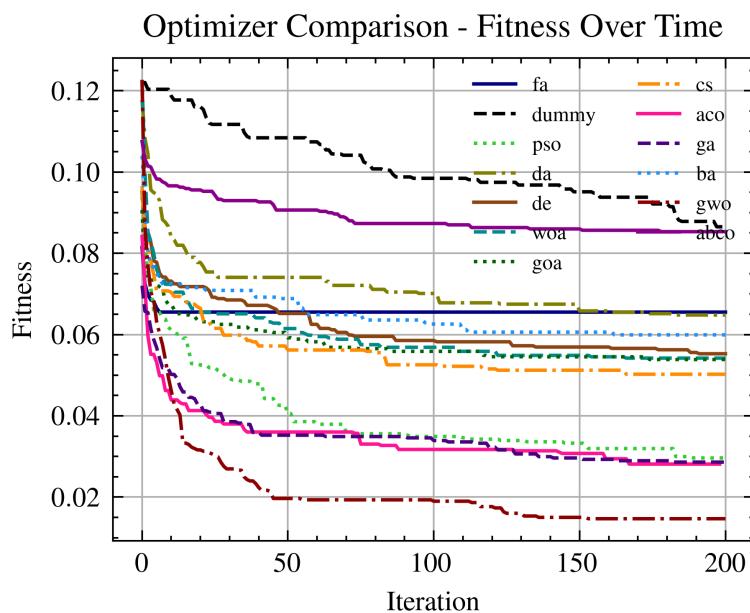


Figura 7.6: Convergencia de todas las metaheurísticas en wdbc - knn - binario

7.2. Continuo

En esta sección se analizarán las metaheurísticas en su versión real o continua. En este apartado se ha optado por no usar en las comparativas el algoritmo **ACO** en su versión continua. Dado que **ACO** fue concebido como un algoritmo binario y que sus implementaciones, pese a haberlas, se alejan bastante de la idea original y distorsionan el algoritmo, se ha tomado la decisión de no incluirlo. Por tanto, se compararán las versiones continuas, que son las originales de cada algoritmo, entre sí. En el caso de **GA** se hace uso de operadores especiales, según se explicó en anteriores capítulos, para adaptarlo a un contexto de codificación real, siendo el único que requería de una modificación de los operadores.

La estrategia seguida en los algoritmos continuos es distinta a los binarios. En los binarios, como se ha mencionado, se eligen características, mientras que otras son rechazadas. Por otro lado, en los continuos no se llegan a descartar características, sino que se ponderan por importancia. Desde este punto de vista, se podría pensar que los continuos tienen mayor maniobrabilidad en el entrenamiento, pues se encargan de ajustar pesos en vez de eliminarlos, por tanto trabajan con más información y más rango de acción. A priori, mayor cantidad de información es mejor. El análisis realizado en esta sección tratará de comprobar como se comportan los continuos para el problema de selección de características, partiendo del hecho de que ninguno de ellos ha sido diseñado específicamente para ello, como sí es el caso de los binarios.

7.2.1. General

De igual forma que en el apartado anterior sobre los binarios, se procede a presentar una serie de resultados iniciales en forma de rankings, los cuales permitirán una visión general del comportamiento.

Según se observa en las figuras de 7.7a y 7.7b, lo que más llama la atención es que hay algoritmos clasificados peor incluso que el **Dummy**, el cual es totalmente aleatorio. Estos son **DA** en **SVC** y **GOA** en ambos clasificadores. Parecen obtener resultados pésimos, pese a ser sus versiones originales.

En cambio, algoritmos que ya habían destacado en binario, lo vuelven a hacer en continuo, además con una puntuación muy buena con respecto al resto de metaheurísticas. Estos son **CS** y **GWO**. En el top de mejores optimizadores entran además los algoritmos de **BA** y **WOA**, los cuales si bien no eran malos en binario, no eran los mejores, más bien promedio. Se mantiene el algoritmo genético dentro del top de los cinco mejores. Los algoritmos clásicos ocupan el grueso del medio en los rankings.

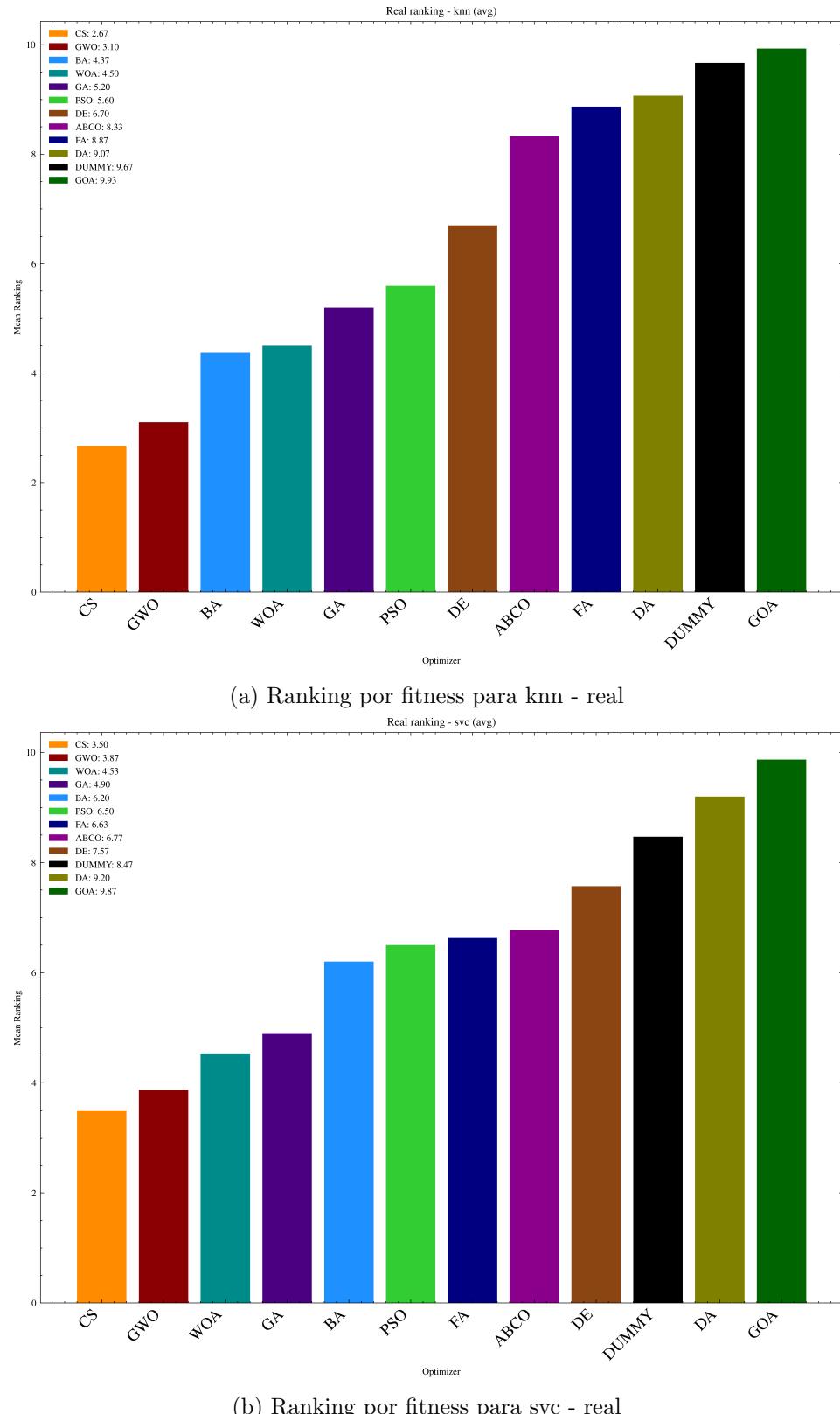


Figura 7.7: Rankings para fitness - real

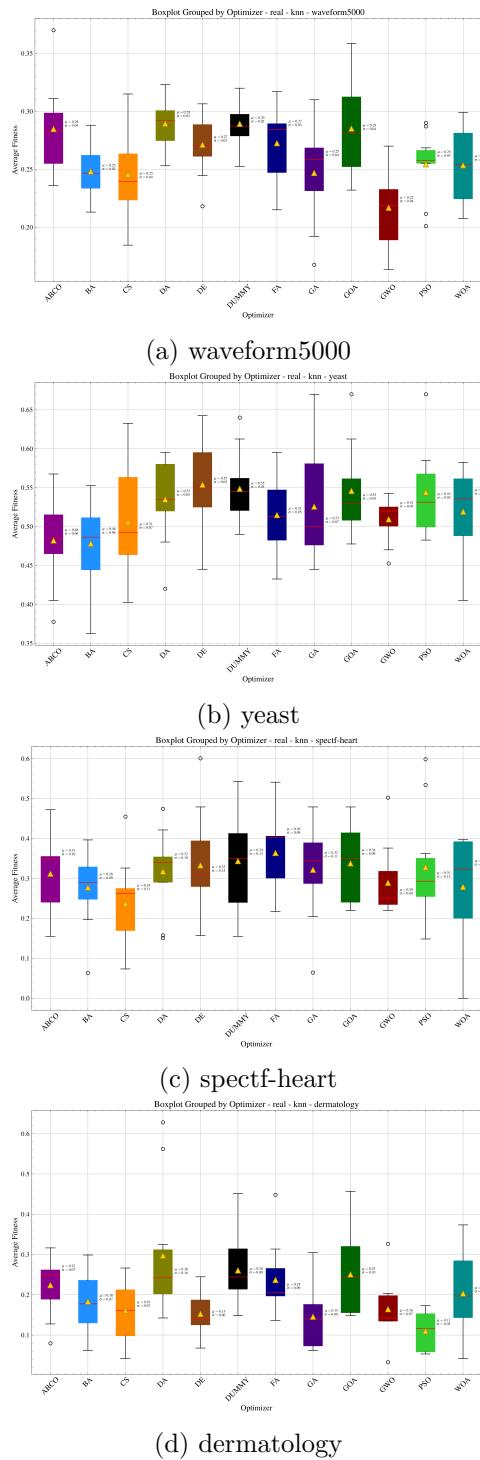
7.2.2. Variabilidad

En las figuras presentadas en 7.8a, 7.8b, 7.8c, 7.8d pueden observarse los resultados obtenidos para los conjuntos de datos **waveform5000**, **yeast**, **spectf-heart** y **dermatology** usando el clasificador **kNN**.

A primera vista, destaca ver la *desviación típica* de los resultados de la mayoría de propuestas, pues es bastante pequeña en general. Esto da a entender que los resultados obtenidos en las 10 pruebas difieren poco entre sí y por tanto estos son bastante robustos.

Destacar como algoritmos muy notorios el **CS**, **GWO**, **BA** y **PSO**, que parecen obtener muy buenos resultados y ser bastante compactos en cuanto a su desviación. Por el contrario, algoritmos como **GOA** parecen obtener un rendimiento más bajo y ser muy variables en comparación al resto. Si se observan las colas de **GOA** en las figuras, esta es más larga por arriba, hacia valores muy altos de *fitness* (se está minimizando, por tanto no es bueno). No representa el grueso de los resultados, pues es la cola, que representa un 25 % de los datos, pero sigue siendo un valor más que importante.

Fitness

Figura 7.8: *Boxplots* para knn - real

Capítulo 8

Bibliografía

- [1] J. E. Hopcroft y J. D. Ullman, *Introduction to Automata Theory, Languages and Computation* (Addison-Wesley Series in Computer Science), 1st. Addison-Wesley Publishing Company, 1979.
- [2] J. v. Leeuwen, *Algorithms and complexity* (Handbook of theoretical computer science / ed. by Jan. van Leeuwen), eng, 1. MIT Press paperback ed., 2. printing. Amsterdam: Elsevier, 1998, OCLC: 247934368, ISBN: 978-0-262-22038-5. URL: <http://www.gbv.de/dms/bowker/toc/9780444880710.pdf> (visitado 20-04-2024).
- [3] Y. Abu-Mostafa, M. Magdon-Ismail y H. Lin, *Learning From Data*. AMLBook, 2012.
- [4] Udacity, *Curse of Dimensionality - Georgia Tech - Machine Learning*, [Online]. Available: Retrieved 2022-06-29, feb. de 2015.
- [5] D. Molina, J. Poyatos, J. Ser, S. García, A. Hussain y F. Herrera, «Comprehensive Taxonomies of Nature- and Bio-inspired Optimization: Inspiration Versus Algorithmic Behavior, Critical Analysis Recommendations,» English, *Cognitive Computation*, vol. 12, n.º 5, págs. 897-939, 2020, ISSN: 1866-9956. DOI: 10.1007/s12559-020-09730-8.
- [6] W. Clark, W. Polakov y F. Trabold, *The Gantt Chart, a Working Tool of Management*, English. New York: The Ronald Press Company, 1922.
- [7] J. Woidasky y E. Cetinkaya, «Use Pattern Relevance for Laptop Repair and Product Lifetime,» *Journal of Cleaner Production*, vol. 288, pág. 125 425, mar. de 2021, ISSN: 0959-6526. DOI: 10.1016/j.jclepro.2020.125425. URL: <https://www.sciencedirect.com/science/article/pii/S0959652620354718> (visitado 16-04-2024).

- [8] CITIC-UGR Sala de Servidores y Clústeres de Computadores, http://citic.ugr.es/pages/informacion_general/equipamiento/cluster, [Online; accessed 12-May-2024].
- [9] A. A. Kulkarni, G. G. Krishnasamy y A. A. Abraham, «Introduction to Optimization,» en *Optimization*. sep. de 2017, vol. 114, págs. 1-7, ISBN: 978-3-319-44253-2. DOI: 10.1007/978-3-319-44254-9_1.
- [10] A. Eiben y J. Smith, *Introduction to Evolutionary Computing* (Natural Computing Series), 2nd. Berlin, Heidelberg: Springer, 2015, pág. 30, S2CID 20912932, ISBN: 978-3-662-44873-1. DOI: 10.1007/978-3-662-44874-8.
- [11] S. Shalev-Shwartz y S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. CUP, 2014, ISBN: 978-3-319-44253-2.
- [12] J. Miao y L. Niu, «A Survey on Feature Selection,» *Procedia Computer Science*, Promoting Business Analytics and Quantitative Management of Technology: 4th International Conference on Information Technology and Quantitative Management (ITQM 2016), vol. 91, págs. 919-926, ene. de 2016, ISSN: 1877-0509. DOI: 10.1016/j.procs.2016.07.111. URL: <https://www.sciencedirect.com/science/article/pii/S1877050916313047> (visitado 02-04-2024).
- [13] N. Venkat, *The Curse of Dimensionality: Inside Out*, sep. de 2018. DOI: 10.13140/RG.2.2.29631.36006.
- [14] R. Bellman, *Dynamic Programming*. Princeton Univ Pr, 1957, ISBN: 978-0691079516.
- [15] R. E. Bellman, *Adaptive Control Processes*. Princeton University Press, 1961.
- [16] D. Peng, Z. Gui y H. Wu, *Interpreting the Curse of Dimensionality from Distance Concentration and Manifold Effect*, en-US, arXiv:2401.00422 [cs], ene. de 2024. DOI: 10.48550/arXiv.2401.00422. URL: <http://arxiv.org/abs/2401.00422> (visitado 20-04-2024).
- [17] K. Beyer, J. Goldstein, R. Ramakrishnan y U. Shaft, «When Is “Nearest Neighbor” Meaningful?» Inglés, en *Proceedings of the 7th International Conference on Database Theory*, ép. Lecture Notes in Computer Science, vol. 1540, Springer, 1999, págs. 217-235. URL: <http://www.springerlink.com/link.asp?id=04p94cqnbge862kh>.
- [18] L. Bianchi, M. Dorigo, L. M. Gambardella y W. J. Gutjahr, «A survey on metaheuristics for stochastic combinatorial optimization,» *Natural Computing*, vol. 8, n.º 2, págs. 239-287, 2009. DOI: 10.1007/s11047-008-9098-4.
- [19] J. Xu y J. Zhang, «Exploration-exploitation tradeoffs in metaheuristics: Survey and analysis,» en *Proceedings of the 33rd Chinese control conference*, IEEE, 2014, págs. 8633-8638.

- [20] J. Xu y J. Zhang, «Exploration-exploitation tradeoffs in metaheuristics: Survey and analysis,» en *Proceedings of the 33rd Chinese Control Conference*, 2014, pág. 8633-8638. DOI: 10.1109/ChiCC.2014.6896450.
- [21] D. H. Wolpert y W. G. Macready, «No free lunch theorems for optimization,» *IEEE Transactions on Evolutionary Computation*, vol. 1, n.º 1, págs. 67-82, 1997. DOI: 10.1109/4235.585893.
- [22] M. Goldblum, M. Finzi, K. Rowan y A. G. Wilson, *The No Free Lunch Theorem, Kolmogorov Complexity, and the Role of Inductive Biases in Machine Learning*, 2023. arXiv: 2304.05366 [cs.LG].
- [23] C. García-Martínez, F. J. Rodríguez y M. Lozano, «Arbitrary function optimisation with metaheuristics,» en, *Soft Computing*, vol. 16, n.º 12, págs. 2115-2133, dic. de 2012, Therefore, there is a need for studies that provide a more rigorous and comprehensive comparative evaluation of the different proposals in this field, proposing a dual study that uses both binary and real adaptations with modern algorithms., ISSN: 1433-7479. DOI: 10.1007/s00500-012-0881-x. URL: <https://doi.org/10.1007/s00500-012-0881-x> (visitado 04-06-2024).
- [24] K. P. Murphy, *Machine Learning: A Probabilistic Perspective* (Adaptive Computation and Machine Learning), 1.^a ed. The MIT Press, 2012.
- [25] S. Sah, *Machine Learning: A Review of Learning Types*, jul. de 2020. DOI: 10.20944/preprints202007.0230.v1.
- [26] C. Cortes y V. Vapnik, «Support-Vector Networks,» en, *Machine Learning*, vol. 20, n.º 3, págs. 273-297, sep. de 1995. DOI: 10.1007/BF00994018. URL: <http://link.springer.com/10.1007/BF00994018> (visitado 02-04-2024).
- [27] scikit-learn Developers, *Support Vector Machines — scikit-learn documentation*, <https://scikit-learn.org/stable/modules/svm.html>, Archived from the original on 2017-11-08, Retrieved 2024-04-27, 2024.
- [28] T. Hastie, R. Tibshirani y J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd. New York: Springer, 2009.
- [29] E. Fix y J. L. Hodges, «Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties,» *International Statistical Review / Revue Internationale de Statistique*, vol. 57, n.º 3, págs. 238-247, 1989, Publisher: [Wiley, International Statistical Institute (ISI)], ISSN: 0306-7734. DOI: 10.2307/1403797. URL: <https://www.jstor.org/stable/1403797> (visitado 02-04-2024).

- [30] T. Cover y P. Hart, «Nearest Neighbor Pattern Classification,» en, *IEEE Transactions on Information Theory*, vol. 13, n.^o 1, págs. 21-27, ene. de 1967. DOI: 10.1109/TIT.1967.1053964. URL: <http://ieeexplore.ieee.org/document/1053964/> (visitado 02-04-2024).
- [31] G. Guo, H. Wang, D. Bell, Y. Bi y K. Greer, «KNN Model-Based Approach in Classification,» en *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, R. Meersman, Z. Tari y D. C. Schmidt, eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, págs. 986-996, ISBN: 978-3-540-39964-3.
- [32] K. Kira y L. A. Rendell, «A Practical Approach to Feature Selection,» English, en *Proc. 9th International Workshop on Machine Learning (ICML 1992)*, 1992, págs. 249-256, ISBN: 978-1-55860-247-2. DOI: 10.1016/B978-1-55860-247-2.50037-1.
- [33] C. Ding y H. Peng, «Minimum Redundancy Feature Selection from Microarray Gene Expression Data,» English, *Journal of Bioinformatics and Computational Biology*, vol. 3, n.^o 2, págs. 185-205, 2005, ISSN: 0219-7200. DOI: 10.1142/S0219720005001004.
- [34] S. Mirjalili, S. M. Mirjalili y A. Lewis, «Grey Wolf Optimizer,» en-US, *Advances in Engineering Software*, vol. 69, págs. 46-61, mar. de 2014, ISSN: 0965-9978. DOI: 10.1016/j.advengsoft.2013.12.007. URL: <https://www.sciencedirect.com/science/article/pii/S0965997813001853> (visitado 18-11-2023).
- [35] X.-S. Yang, «Chapter 8 - Firefly Algorithms,» en-US, en *Nature-Inspired Optimization Algorithms*, X.-S. Yang, ed., Oxford: Elsevier, ene. de 2014, págs. 111-127, ISBN: 978-0-12-416743-8. DOI: 10.1016/B978-0-12-416743-8.00008-7. URL: <https://www.sciencedirect.com/science/article/pii/B9780124167438000087> (visitado 23-01-2024).
- [36] S. Saremi, S. Mirjalili y A. Lewis, «Grasshopper Optimisation Algorithm: Theory and application,» en-US, *Advances in Engineering Software*, vol. 105, págs. 30-47, mar. de 2017, ISSN: 0965-9978. DOI: 10.1016/j.advengsoft.2017.01.004. URL: <https://www.sciencedirect.com/science/article/pii/S0965997816305646> (visitado 04-11-2023).
- [37] S. Mirjalili y A. Lewis, «The Whale Optimization Algorithm,» en-US, *Advances in Engineering Software*, vol. 95, págs. 51-67, mayo de 2016, ISSN: 0965-9978. DOI: 10.1016/j.advengsoft.2016.01.008. URL: <https://www.sciencedirect.com/science/article/pii/S0965997816300163> (visitado 14-10-2023).

- [38] S. Mirjalili, «Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems,» en, *Neural Computing and Applications*, vol. 27, n.º 4, págs. 1053-1073, mayo de 2016, ISSN: 1433-3058. DOI: 10.1007/s00521-015-1920-1. URL: <https://doi.org/10.1007/s00521-015-1920-1> (visitado 06-11-2023).
- [39] X.-S. Yang y S. Deb, *Cuckoo Search via Levy Flights*, en-US, arXiv:1003.1594 [math] version: 1, mar. de 2010. DOI: 10.48550/arXiv.1003.1594. URL: <http://arxiv.org/abs/1003.1594> (visitado 13-03-2024).
- [40] X.-S. Yang, *A New Metaheuristic Bat-Inspired Algorithm*, en-US, Issue: arXiv:1004.4170 arXiv:1004.4170 [physics], abr. de 2010. DOI: 10.48550/arXiv.1004.4170. URL: <http://arxiv.org/abs/1004.4170> (visitado 22-01-2024).
- [41] J. Kennedy y R. Eberhart, «Particle swarm optimization,» en-US, en *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, nov. de 1995, 1942-1948 vol.4. DOI: 10.1109/ICNN.1995.488968. URL: <https://ieeexplore.ieee.org/document/488968>.
- [42] J. H. Holland, *Adaptation in Natural and Artificial Systems*, 2nd. Ann Arbor, MI: University of Michigan Press, 1975, second edition, 1992.
- [43] M. Dorigo y G. D. Caro, «Ant colony optimization: a new meta-heuristic,» en *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, vol. 2, jul. de 1999, 1470-1477 Vol. 2. DOI: 10.1109/CEC.1999.782657. URL: <https://ieeexplore.ieee.org/document/782657>.
- [44] R. Storn y K. Price, «Differential Evolution - A Simple and Efficient Heuristic for global Optimization over Continuous Spaces,» *Journal of Global Optimization*, vol. 11, n.º 4, págs. 341-359, dic. de 1997, ISSN: 1573-2916. DOI: 10.1023/A:1008202821328. URL: <https://doi.org/10.1023/A:1008202821328>.
- [45] D. Karaboga, «AN IDEA BASED ON HONEY BEE SWARM FOR NUMERICAL OPTIMIZATION,» en,
- [46] D. Simon, *Evolutionary Optimization Algorithms*. Wiley, 2013, ISBN: 9781118659502. URL: <https://books.google.es/books?id=gwUwIEPqk30C>.
- [47] A. Colomi, M. Dorigo y V. Maniezzo, «Distributed Optimization by Ant Colonies,» en *Proceedings of the First European Conference on Artificial Life*, ene. de 1991.
- [48] M. Jaderberg et al., *Population Based Training of Neural Networks*, 2017. arXiv: 1711.09846 [cs.LG].

- [49] T. Dokeroglu, A. Deniz y H. E. Kiziloz, «A comprehensive survey on recent metaheuristics for feature selection,» en-US, *Neurocomputing*, vol. 494, págs. 269-296, jul. de 2022, ISSN: 0925-2312. DOI: 10.1016/j.neucom.2022.04.083. URL: <https://www.sciencedirect.com/science/article/pii/S092523122200474X> (visitado 22-09-2023).
- [50] I. Boussaïd, J. Lepagnot y P. Siarry, «A survey on optimization metaheuristics,» *Information Sciences, Prediction, Control and Diagnosis using Advanced Neural Computations*, vol. 237, págs. 82-117, jul. de 2013, ISSN: 0020-0255. DOI: 10.1016/j.ins.2013.02.041. URL: <https://www.sciencedirect.com/science/article/pii/S0020025513001588> (visitado 20-04-2024).
- [51] P. Agrawal, H. Abutarboush, T. Ganesh y A. Mohamed, «Metaheuristic algorithms on feature selection: A survey of one decade of research (2009-2019),» en-US, *IEEE Access*, vol. 9, págs. 26 766-26 791, 2021, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3056407.
- [52] A. G. Guy, D. A. Bohan, S. J. Powers y A. M. Reynolds, «Avoidance of conspecific odour by carabid beetles: a mechanism for the emergence of scale-free searching patterns,» *Animal Behaviour*, vol. 76, n.º 3, págs. 585-591, 2008, ISSN: 0003-3472. DOI: 10.1016/j.anbehav.2008.04.004. URL: <https://www.sciencedirect.com/science/article/pii/S0003347208001668>.
- [53] E. Emary, H. M. Zawbaa y A. E. Hassanien, «Binary grey wolf optimization approaches for feature selection,» en-US, *Neurocomputing*, vol. 172, págs. 371-381, ene. de 2016, ISSN: 0925-2312. DOI: 10.1016/j.neucom.2015.06.083. URL: <https://www.sciencedirect.com/science/article/pii/S0925231215010504> (visitado 18-11-2023).
- [54] Q. Al-Tashi, S. J. A. Kadir, H. M. Rais, S. Mirjalili y H. Alhussein, «Binary Optimization Using Hybrid Grey Wolf Optimization for Feature Selection,» *IEEE Access*, vol. 7, págs. 39 496-39 508, 2019, Cited by: 354; All Open Access, Gold Open Access, Green Open Access. DOI: 10.1109/ACCESS.2019.2906757. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85065103601&doi=10.1109%2fACCESS.2019.2906757&partnerID=40&md5=ec600f296b2a0e1986aecc74e526443c>.
- [55] P. Hu, J.-S. Pan y S.-C. Chu, «Improved Binary Grey Wolf Optimizer and Its application for feature selection,» *Knowledge-Based Systems*, vol. 195, 2020, Cited by: 282. DOI: 10.1016/j.knosys.2020.105746. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85081912362&doi=10.1016%2fj.knosys.2020.105746&partnerID=40&md5=e4bc8c5352463511026244d5841acd92>.

- [56] A. G. Hussien, A. E. Hassanien, E. H. Houssein, S. Bhattacharyya y M. Amin, «S-shaped Binary Whale Optimization Algorithm for Feature Selection,» en, en *Recent Trends in Signal and Image Processing*, S. Bhattacharyya, A. Mukherjee, H. Bhaumik, S. Das y K. Yoshida, eds., ép. Advances in Intelligent Systems and Computing, Singapore: Springer, 2019, págs. 79-87, ISBN: 978-981-10-8863-6. DOI: 10.1007/978-981-10-8863-6_9.
- [57] M. Mafarja y S. Mirjalili, «Whale optimization approaches for wrapper feature selection,» *Applied Soft Computing*, vol. 62, págs. 441-453, ene. de 2018, ISSN: 1568-4946. DOI: 10.1016/j.asoc.2017.11.006. URL: <https://www.sciencedirect.com/science/article/pii/S1568494617306695> (visitado 20-11-2023).
- [58] M. Mafarja et al., «Binary dragonfly optimization for feature selection using time-varying transfer functions,» English, *Knowledge-Based Systems*, vol. 161, págs. 185-204, 2018, ISSN: 0950-7051. DOI: 10.1016/j.knosys.2018.08.003.
- [59] A. I. Hammouri, M. Mafarja, M. A. Al-Betar, M. A. Awadallah e I. Abu-Doush, «An improved Dragonfly Algorithm for feature selection,» *Knowledge-Based Systems*, vol. 203, 2020, Cited by: 139. DOI: 10.1016/j.knosys.2020.106131. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85086804574&doi=10.1016%2fj.knosys.2020.106131&partnerID=40&md5=b7fb8f78062741d4eb957090cadbdae7>.
- [60] G. I. Sayed, A. Tharwat y A. E. Hassanien, «Chaotic dragonfly algorithm: an improved metaheuristic algorithm for feature selection,» *Applied Intelligence*, vol. 49, n.º 1, págs. 188-205, 2019, Cited by: 201; All Open Access, Bronze Open Access. DOI: 10.1007/s10489-018-1261-8. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85052571042&doi=10.1007%2fs10489-018-1261-8&partnerID=40&md5=eabb8615b280a1c2e012b4786cf398b2>.
- [61] M. M. Mafarja, D. Eleyan, I. Jaber, A. Hammouri y S. Mirjalili, «Binary Dragonfly Algorithm for Feature Selection,» Cited by: 184; All Open Access, Green Open Access, vol. 2018-January, 2017, págs. 12-17. DOI: 10.1109/ICTCS.2017.43. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85050194810&doi=10.1109%2fICTCS.2017.43&partnerID=40&md5=a1c7a06b5517e09598543c9ea3aec738>.
- [62] M. Mafarja, I. Aljarah, H. Faris, A. I. Hammouri, A. M. Al-Zoubi y S. Mirjalili, «Binary grasshopper optimisation algorithm approaches for feature selection problems,» *Expert Systems with Applications*, vol. 117, págs. 267-286, mar. de 2019, ISSN: 0957-4174. DOI: 10.1016/j.eswa.2018.09.015. URL: <https://www.sciencedirect.com/science/article/pii/S0957417418305864> (visitado 23-10-2023).

- [63] I. Aljarah, A. M. Al-Zoubi, H. Faris, M. A. Hassonah, S. Mirjalili y H. Saadeh, «Simultaneous Feature Selection and Support Vector Machine Optimization Using the Grasshopper Optimization Algorithm,» *Cognitive Computation*, vol. 10, n.º 3, págs. 478-495, 2018, Cited by: 198; All Open Access, Green Open Access. DOI: 10.1007/s12559-017-9542-9. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85040796296&doi=10.1007%2fs12559-017-9542-9&partnerID=40&md5=2ab6babc5f6bf1d512e4fce656ac88e9>.
- [64] M. Mafarja et al., «Evolutionary Population Dynamics and Grasshopper Optimization approaches for feature selection problems,» *Knowledge-Based Systems*, vol. 145, págs. 25-45, 2018, Cited by: 351; All Open Access, Green Open Access. DOI: 10.1016/j.knosys.2017.12.037. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85040020546&doi=10.1016%2fj.knosys.2017.12.037&partnerID=40&md5=48c70ba8b45fb49d6218a39cfdc5c11d>.
- [65] S. Mirjalili, S. Mirjalili y X.-S. Yang, «Binary bat algorithm,» en-US, *Neural Computing and Applications*, vol. 25, n.º 3-4, págs. 663-681, 2014, Number: 3-4, ISSN: 0941-0643. DOI: 10.1007/s00521-013-1525-5.
- [66] R. Y. M. Nakamura, L. A. M. Pereira, K. A. Costa, D. Rodrigues, J. P. Papa y X.-S. Yang, «BBA: A binary bat algorithm for feature selection,» Cited by: 340, 2012, págs. 291-297. DOI: 10.1109/SIBGRAPI.2012.47. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84872367831&doi=10.1109%2fSIBGRAPI.2012.47&partnerID=40&md5=ff823b28bacfe7b29f95d173351af4bf>.
- [67] D. Rodrigues et al., «A wrapper approach for feature selection based on Bat Algorithm and Optimum-Path Forest,» *Expert Systems with Applications*, vol. 41, n.º 5, págs. 2250-2258, 2014, Cited by: 223. DOI: 10.1016/j.eswa.2013.09.023. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84890121325&doi=10.1016%2fj.eswa.2013.09.023&partnerID=40&md5=a8b03951959d33ab4549ab19d93b1131>.
- [68] D. Rodrigues et al., «BCS: A Binary Cuckoo Search algorithm for feature selection,» en *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, ISSN: 2158-1525, mayo de 2013, págs. 465-468. DOI: 10.1109/ISCAS.2013.6571881. URL: <https://ieeexplore.ieee.org/document/6571881> (visitado 13-03-2024).
- [69] L. Zhang, L. Shan y J. Wang, «Optimal feature selection using distance-based discrete firefly algorithm with mutual information criterion,» *Neural Computing and Applications*, 2016. DOI: 10.1007/s00521-016-2204-0.

- [70] Y. Zhang, X. Song y D. Gong, «A return-cost-based binary firefly algorithm for feature selection,» *Information Sciences*, vol. 418-419, págs. 561-574, 2017, Cited by: 198. DOI: 10.1016/j.ins.2017.08.047. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85027418461&doi=10.1016%2fj.ins.2017.08.047&partnerID=40&md5=d58e60ad246a9e78eeeed90fee3db508>.
- [71] B. Selvakumar y K. Muneeswaran, «Firefly algorithm based feature selection for network intrusion detection,» *Computers and Security*, vol. 81, págs. 148-155, 2019, Cited by: 225. DOI: 10.1016/j.cose.2018.11.005. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85058021385&doi=10.1016%2fj.cose.2018.11.005&partnerID=40&md5=7b5ec941535b4fdbba6e9fa389d0559>.
- [72] Z. Hu, Y. Bao, T. Xiong y R. Chiong, «Hybrid filter-wrapper feature selection for short-term load forecasting,» *Engineering Applications of Artificial Intelligence*, vol. 40, págs. 17-27, 2015, Cited by: 169. DOI: 10.1016/j.engappai.2014.12.014. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84923374513&doi=10.1016%2fj.engappai.2014.12.014&partnerID=40&md5=0da35f2297a63bf29bb4533935cf5e9c>.
- [73] S. Mirjalili y A. Lewis, «S-shaped versus V-shaped transfer functions for binary Particle Swarm Optimization,» *Swarm and Evolutionary Computation*, vol. 9, págs. 1-14, abr. de 2013, ISSN: 2210-6502. DOI: 10.1016/j.swevo.2012.09.002. URL: <https://www.sciencedirect.com/science/article/pii/S2210650212000648> (visitado 29-10-2023).
- [74] S. W. Lin, K. C. Ying, S. C. Chen y Z. J. Lee, «Particle swarm optimization for parameter determination and feature selection of support vector machines,» *Expert Systems with Applications*, vol. 35, n.º 4, págs. 1817-1824, 2008, Cited by: 801. DOI: 10.1016/j.eswa.2007.08.088. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-48749109333&doi=10.1016%2fj.eswa.2007.08.088&partnerID=40&md5=797a9356d2b699d1b67214c0dfc71c05>.
- [75] B. Xue, M. Zhang y W. N. Browne, «Particle swarm optimization for feature selection in classification: A multi-objective approach,» *IEEE Transactions on Cybernetics*, vol. 43, n.º 6, págs. 1656-1671, 2013, Cited by: 965. DOI: 10.1109/TSMCB.2012.2227469. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84887799080&doi=10.1109%2fTSMCB.2012.2227469&partnerID=40&md5=77e7aaaf2d80ac386fa7702f464f25d81>.
- [76] X. Wang, J. Yang, X. Teng, W. Xia y R. Jensen, «Feature selection based on rough sets and particle swarm optimization,» *Pattern Recognition Letters*, vol. 28, n.º 4, págs. 459-471, 2007, Cited by: 745; All Open

- Access, Green Open Access. DOI: 10.1016/j.patrec.2006.09.003. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-33845523839&doi=10.1016%2fj.patrec.2006.09.003&partnerID=40&md5=7d2f5d2292c94da6d266d62a5af22217>.
- [77] M. S. Kiran, «A binary artificial bee colony algorithm and its performance assessment,» en, *Expert Systems with Applications*, vol. 175, pág. 114817, ago. de 2021, ISSN: 09574174. DOI: 10.1016/j.eswa.2021.114817. URL: <https://linkinghub.elsevier.com/retrieve/pii/S095741742100258X> (visitado 28-11-2023).
- [78] H. Rao et al., «Feature selection based on artificial bee colony and gradient boosting decision tree,» *Applied Soft Computing Journal*, vol. 74, págs. 634-642, 2019, Cited by: 424. DOI: 10.1016/j.asoc.2018.10.036. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85056633453&doi=10.1016%2fj.asoc.2018.10.036&partnerID=40&md5=ea596d5758105a3777fa9232e27645f8>.
- [79] E. Zorarpaci y S. A. Özal, «A hybrid approach of differential evolution and artificial bee colony for feature selection,» *Expert Systems with Applications*, vol. 62, págs. 91-103, 2016, Cited by: 303. DOI: 10.1016/j.eswa.2016.06.004. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84975037777&doi=10.1016%2fj.eswa.2016.06.004&partnerID=40&md5=9e481521fd271007fadbac43e199aa6f>.
- [80] E. Hancer, B. Xue, M. Zhang, D. Karaboga y B. Akay, «Pareto front feature selection based on artificial bee colony optimization,» *Information Sciences*, vol. 422, págs. 462-479, 2018, Cited by: 254; All Open Access, Green Open Access. DOI: 10.1016/j.ins.2017.09.028. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85029537929&doi=10.1016%2fj.ins.2017.09.028&partnerID=40&md5=43f4305ef38c9560eaa0436f6d936ee1>.
- [81] G. Pampara, A. Engelbrecht y N. Franken, «Binary Differential Evolution,» en *2006 IEEE International Conference on Evolutionary Computation*, ISSN: 1941-0026, jul. de 2006, págs. 1873-1879. DOI: 10.1109/CEC.2006.1688535. URL: <https://ieeexplore.ieee.org/document/1688535> (visitado 25-03-2024).
- [82] Y. Zhang, D. Gong, X. Gao, T. Tian y X. Sun, «Binary differential evolution with self-learning for multi-objective feature selection,» *Information Sciences*, vol. 507, págs. 67-85, 2020, Cited by: 314. DOI: 10.1016/j.ins.2019.08.040. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85070734759&doi=10.1016%2fj.ins.2019.08.040&partnerID=40&md5=84fb3bbba58178da727bd9143d385c70>.

- [83] E. Hancer, B. Xue y M. Zhang, «Differential evolution for filter feature selection based on information theory and feature ranking,» *Knowledge-Based Systems*, vol. 140, págs. 103-119, 2018, Cited by: 295; All Open Access, Green Open Access. DOI: 10.1016/j.knosys.2017.10.028. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85034999468&doi=10.1016%2fj.knosys.2017.10.028&partnerID=40&md5=234726be47f971c3d7b5785b3919b973>.
- [84] K. Socha, «ACO for Continuous and Mixed-Variable Optimization,» en, en *Ant Colony Optimization and Swarm Intelligence*, M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada y T. Stützle, eds., ép. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2004, págs. 25-36, ISBN: 978-3-540-28646-2. DOI: 10.1007/978-3-540-28646-2_3.
- [85] Y. He, F. Zhang, S. Mirjalili y T. Zhang, «Novel binary differential evolution algorithm based on Taper-shaped transfer functions for binary optimization problems,» *Swarm and Evolutionary Computation*, vol. 69, pág. 101 022, mar. de 2022, ISSN: 2210-6502. DOI: 10.1016/j.swevo.2021.101022. URL: <https://www.sciencedirect.com/science/article/pii/S221065022100184X> (visitado 27-03-2024).
- [86] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998, ISBN: 0262631857.
- [87] T. V. Mathew, «Genetic algorithm,» *Report submitted at IIT Bombay*, vol. 53, 2012.
- [88] S. Mirjalili y S. Mirjalili, «Genetic algorithm,» *Evolutionary algorithms and neural networks: Theory and applications*, págs. 43-55, 2019.
- [89] B. L. Miller, «Genetic Algorithms, Tournament Selection, and the Effects of Noise,» en,
- [90] Z. C. Dagdia y M. Mirchev, «Chapter 15 - When Evolutionary Computing Meets Astro- and Geoinformatics,» en *Knowledge Discovery in Big Data from Astronomy and Earth Observation*, P. Škoda y F. Adam, eds., Elsevier, 2020, págs. 283-306, ISBN: 978-0-12-819154-5. DOI: 10.1016/B978-0-12-819154-5.00026-6. URL: <https://www.sciencedirect.com/science/article/pii/B9780128191545000266>.
- [91] Purdue University College of Engineering, *Lecture 4: Real-Coded Genetic Algorithms*, Lecture notes, Accessed on April 27, 2024. URL: <https://engineering.purdue.edu/~sudhoff/ee630/Lecture04.pdf>.

- [92] S. Kashef y H. Nezamabadi-pour, «An advanced ACO algorithm for feature subset selection,» *Neurocomputing*, vol. 147, págs. 271-279, ene. de 2015, ISSN: 0925-2312. DOI: 10.1016/j.neucom.2014.06.067. URL: <https://www.sciencedirect.com/science/article/pii/S0925231214008601>.
- [93] A. P. Engelbrecht, *Computational Intelligence: An Introduction*, 2nd. Wiley Publishing, 2007, ISBN: 0470035617.
- [94] D. Karaboga y B. Akay, «A comparative study of Artificial Bee Colony algorithm,» *Applied Mathematics and Computation*, vol. 214, n.º 1, págs. 108-132, 2009, Cited by: 2940. DOI: 10.1016/j.amc.2009.03.090. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-67349273050&doi=10.1016%2fj.amc.2009.03.090&partnerID=40&md5=505464030a4a96a1998b20803cd113ce>.
- [95] *Lévy Flight - an overview — ScienceDirect Topics*. URL: <https://www.sciencedirect.com/topics/physics-and-astronomy/levy-flight> (visitado 06-05-2024).
- [96] Y. Zhang, X.-f. Song y D.-w. Gong, «A return-cost-based binary firefly algorithm for feature selection,» en, *Information Sciences*, vol. 418-419, págs. 561-574, dic. de 2017, ISSN: 00200255. DOI: 10.1016/j.ins.2017.08.047. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0020025516314098> (visitado 21-03-2024).
- [97] *MinMaxScaler*, Scikit-learn Documentation, URL: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html>.
- [98] Y. Li, T. Han, H. Zhao y H. Gao, «An Adaptive Whale Optimization Algorithm Using Gaussian Distribution Strategies and Its Application in Heterogeneous UCAVs Task Allocation,» *IEEE Access*, vol. 7, págs. 110138-110158, 2019. URL: <https://api.semanticscholar.org/CorpusID:201621393>.
- [99] D. Rey y M. Neuh, «Wilcoxon-Signed-Rank Test,» en *International Encyclopedia of Statistical Science*, M. Lovric, ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, págs. 1658-1659, ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_616. URL: https://doi.org/10.1007/978-3-642-04898-2%5C_616.
- [100] S. Holm, «A simple sequentially rejective multiple test procedure,» *Scandinavian Journal of Statistics*, vol. 6, n.º 2, págs. 65-70, 1979.
- [101] D. Molina, *TACO Website*, <https://tacolab.org/>, Accessed: 2024-06-01, 2024.

Apéndice A

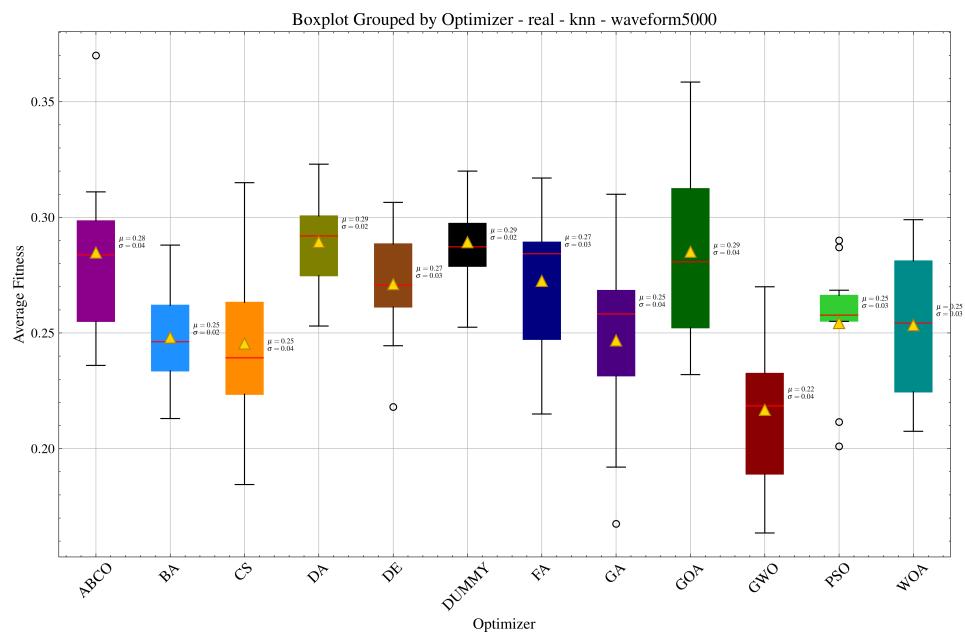
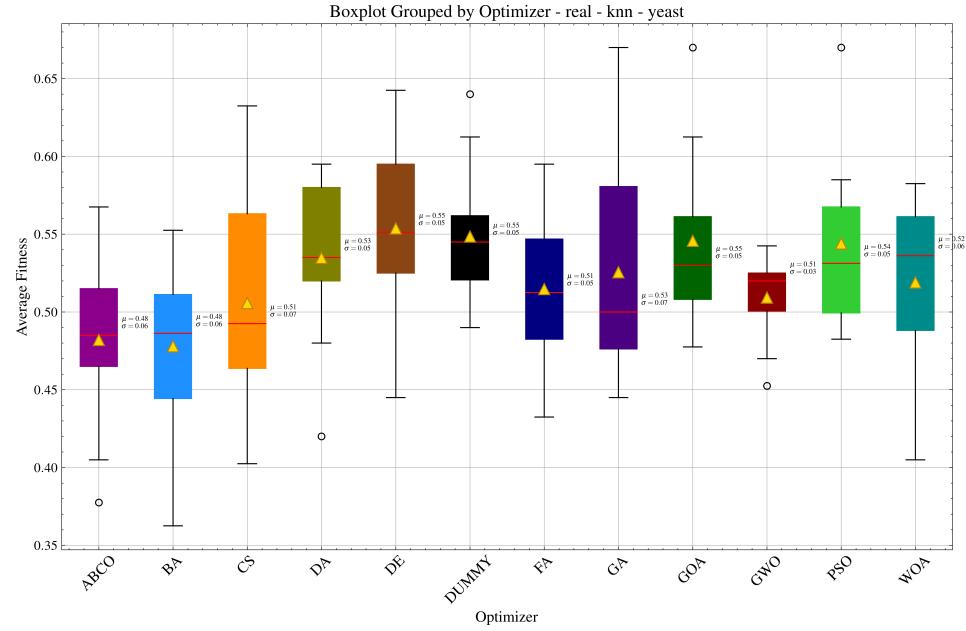
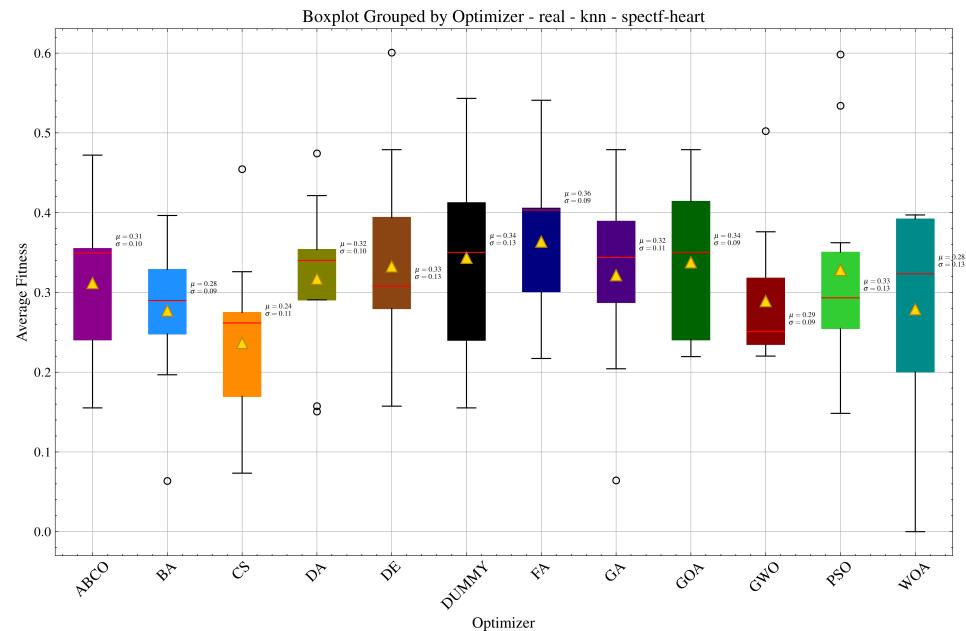


Figura A.1: *Boxplot* waveform5000 - knn - real

Figura A.2: *Boxplot yeast - knn - real*Figura A.3: *Boxplot spectf-heart - knn - real*

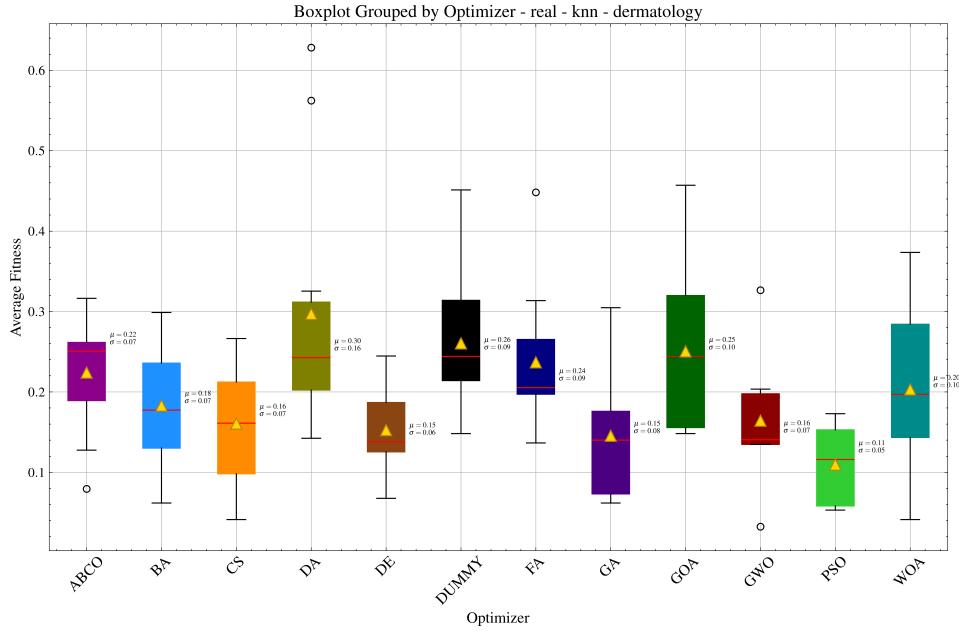


Figura A.4: Boxplot dermatology - knn - real

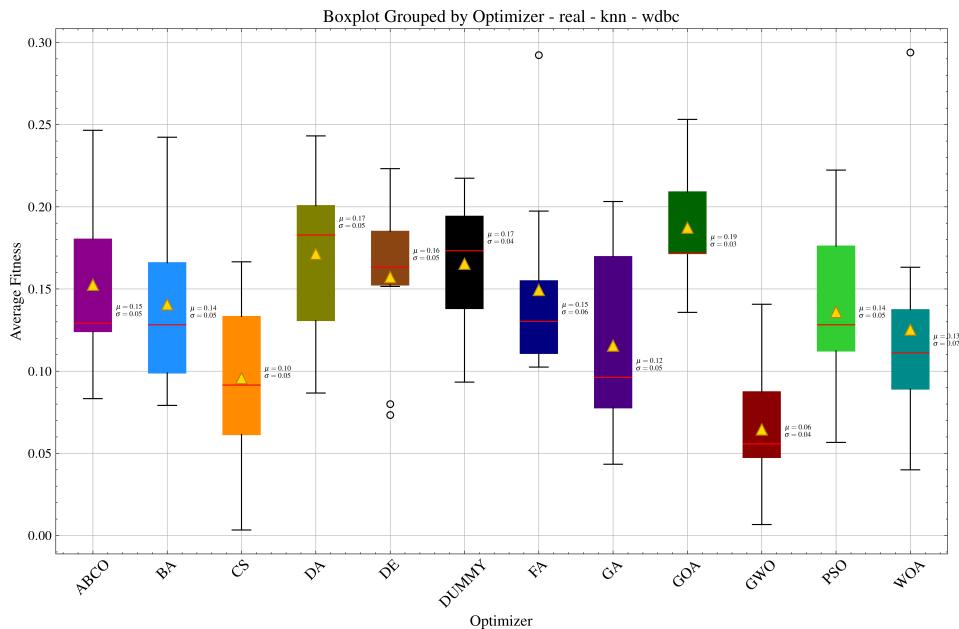


Figura A.5: Boxplot wdbc - knn - real

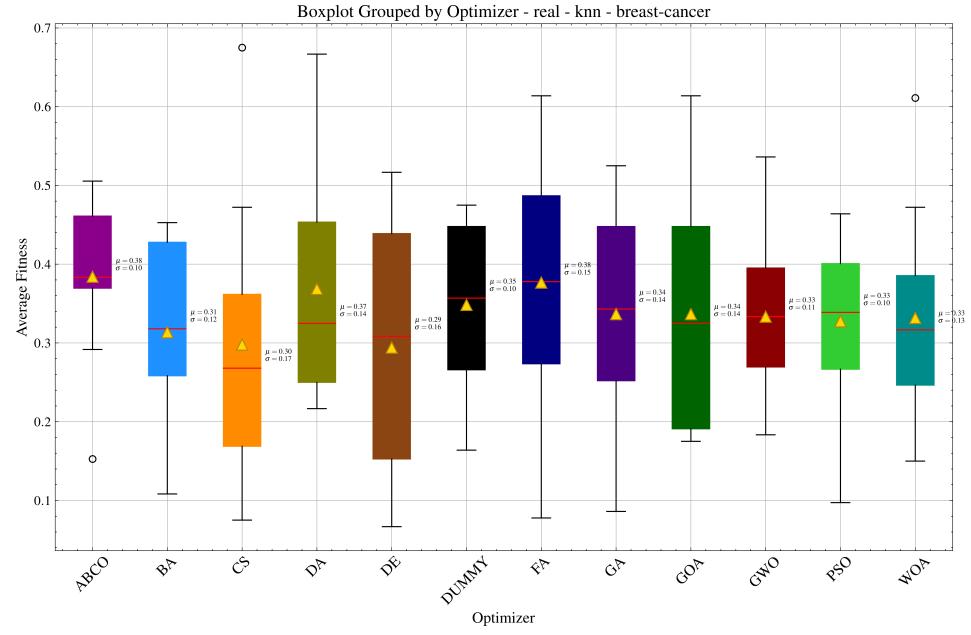


Figura A.6: Boxplot breast-cancer - knn - real

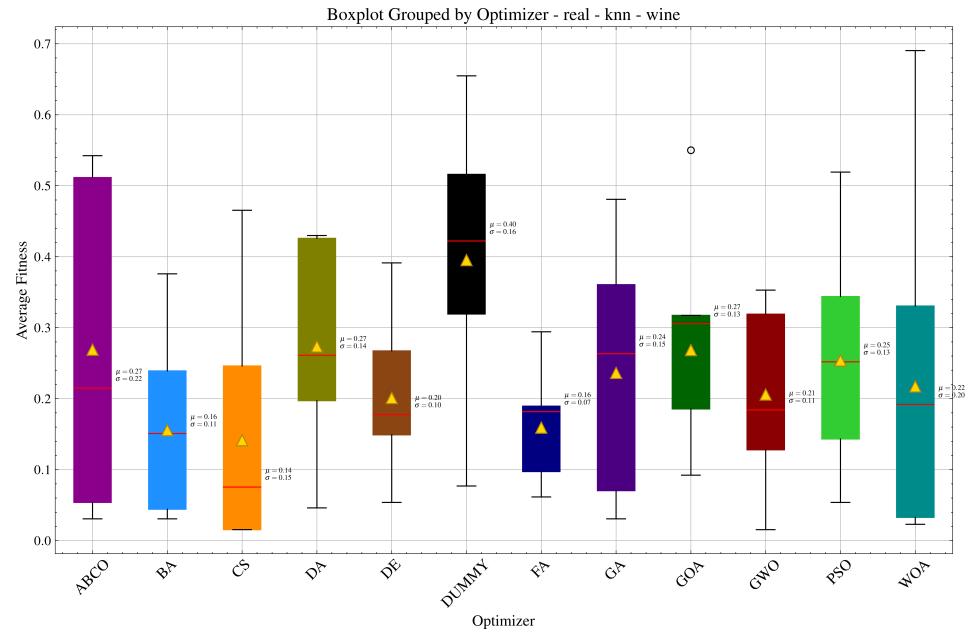


Figura A.7: Boxplot wine - knn - real

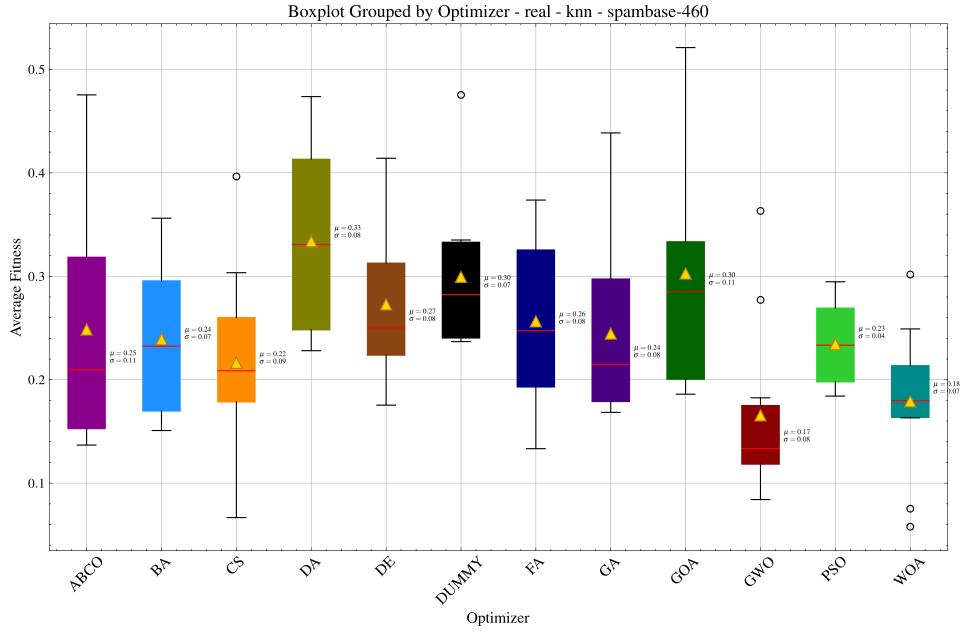


Figura A.8: *Boxplot spambase-460 - knn - real*

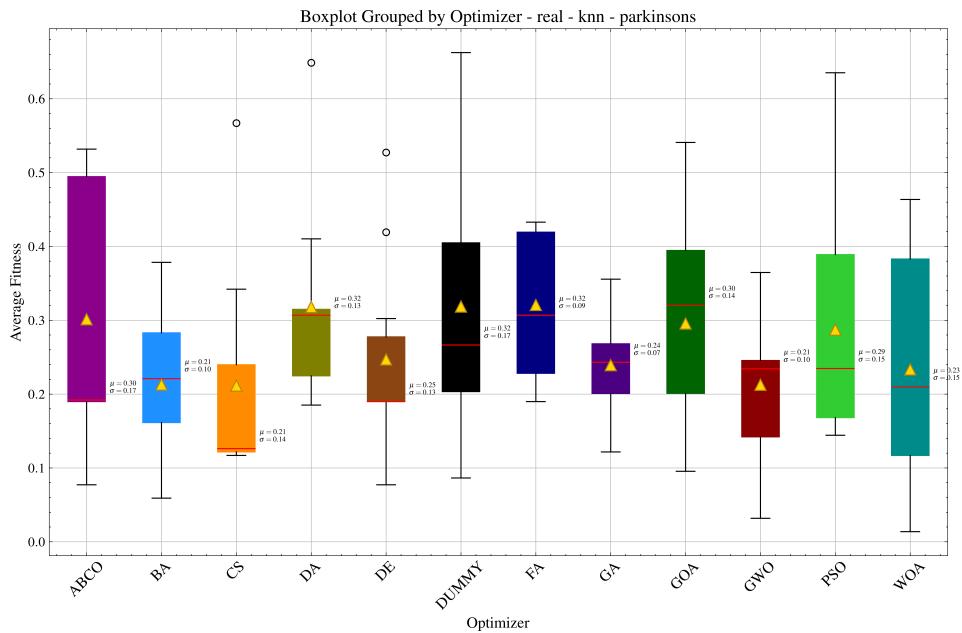


Figura A.9: *Boxplot parkinsons - knn - real*

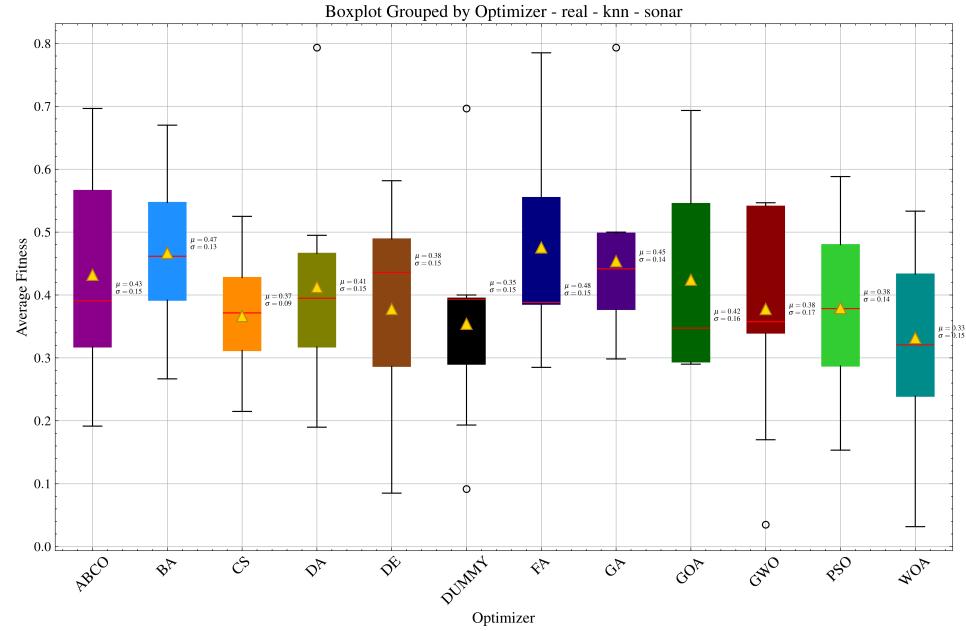


Figura A.10: Boxplot sonar - knn - real

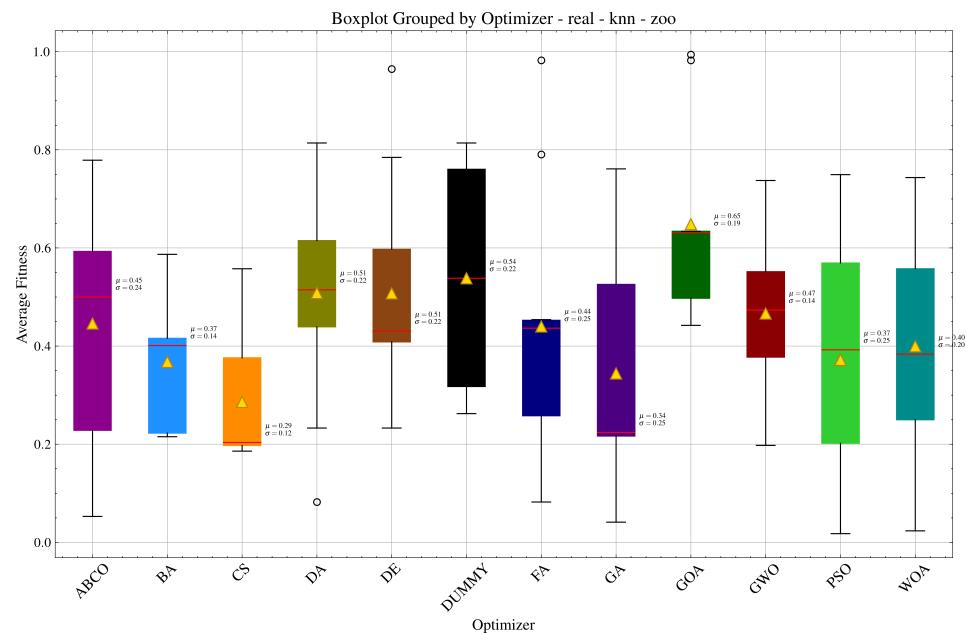


Figura A.11: Boxplot zoo - knn - real

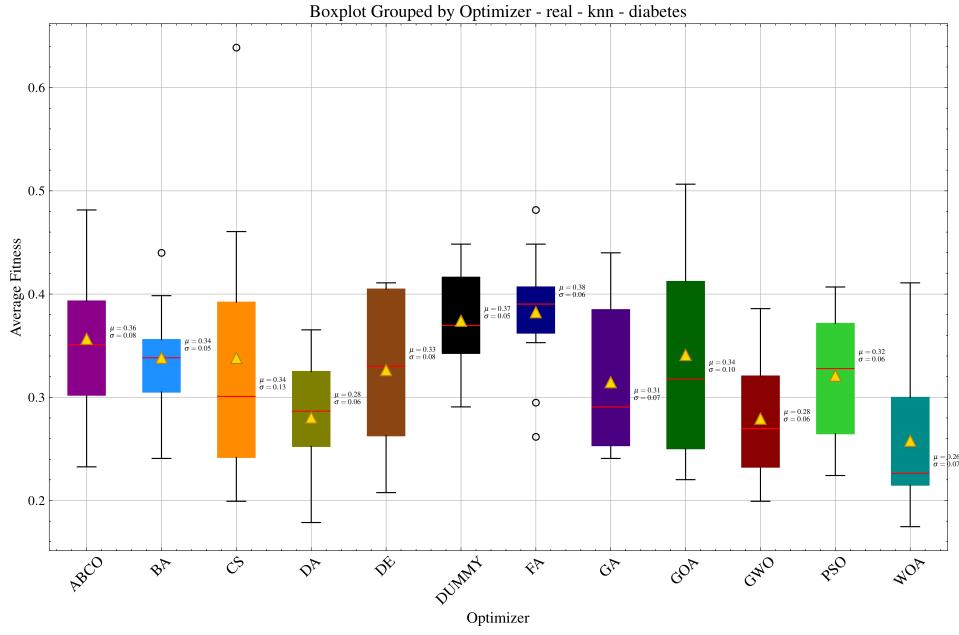


Figura A.12: *Boxplot diabetes - knn - real*

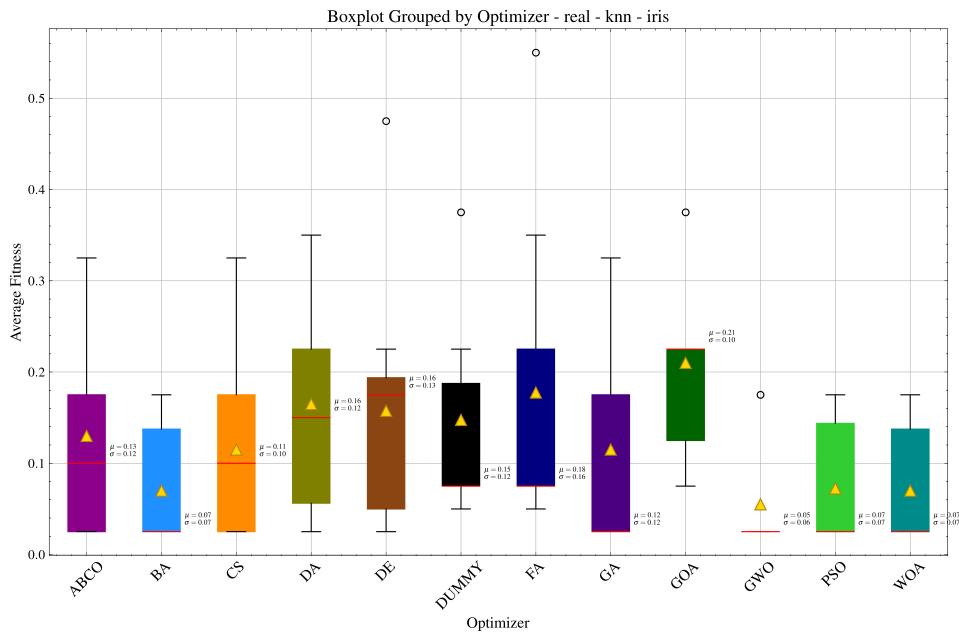


Figura A.13: *Boxplot iris - knn - real*

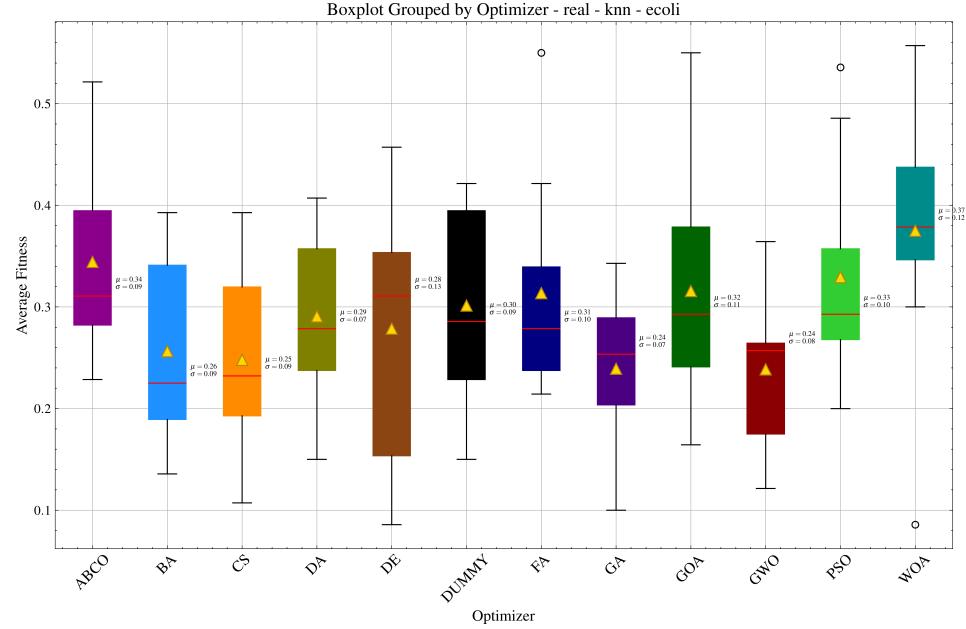


Figura A.14: Boxplot ecoli - knn - real

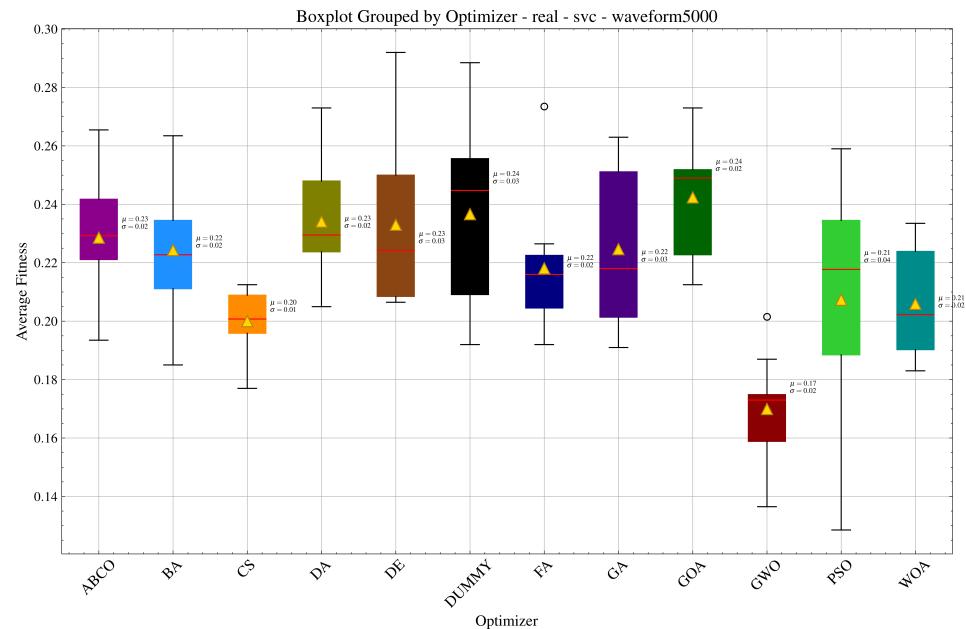


Figura A.15: Boxplot waveform5000 - svc - real

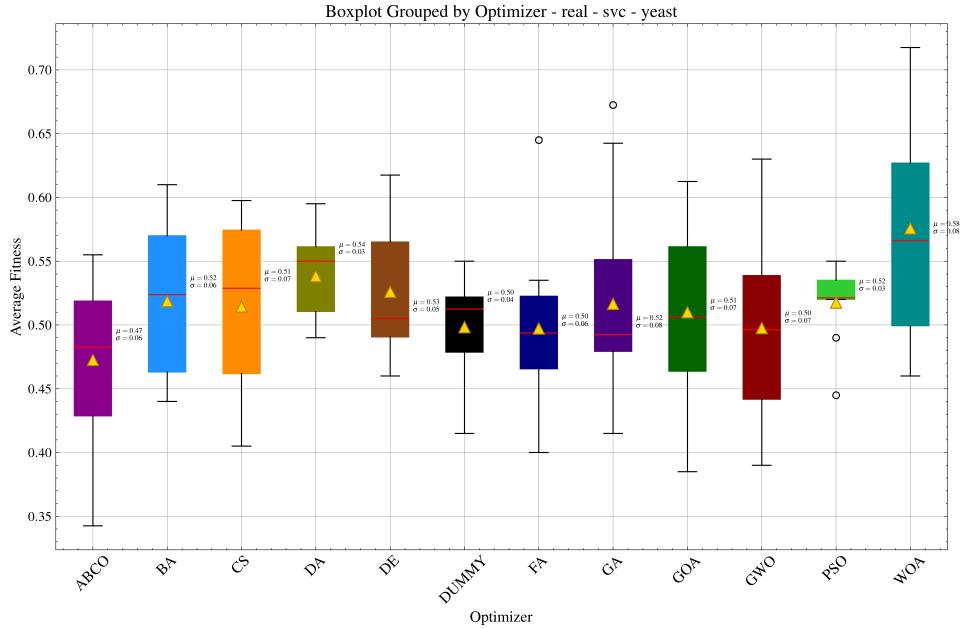


Figura A.16: Boxplot yeast - svc - real

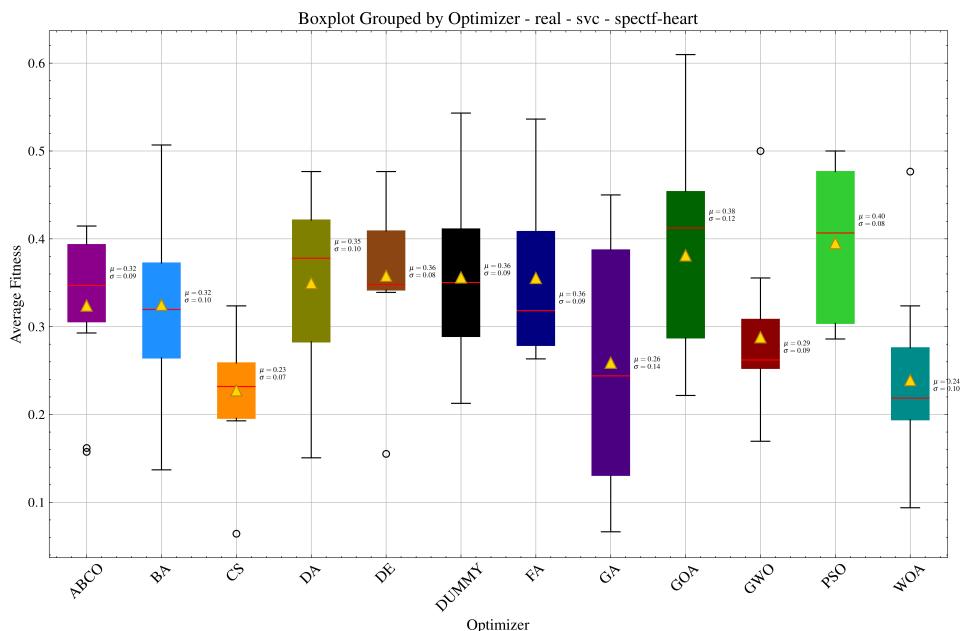


Figura A.17: Boxplot spectf-heart - svc - real

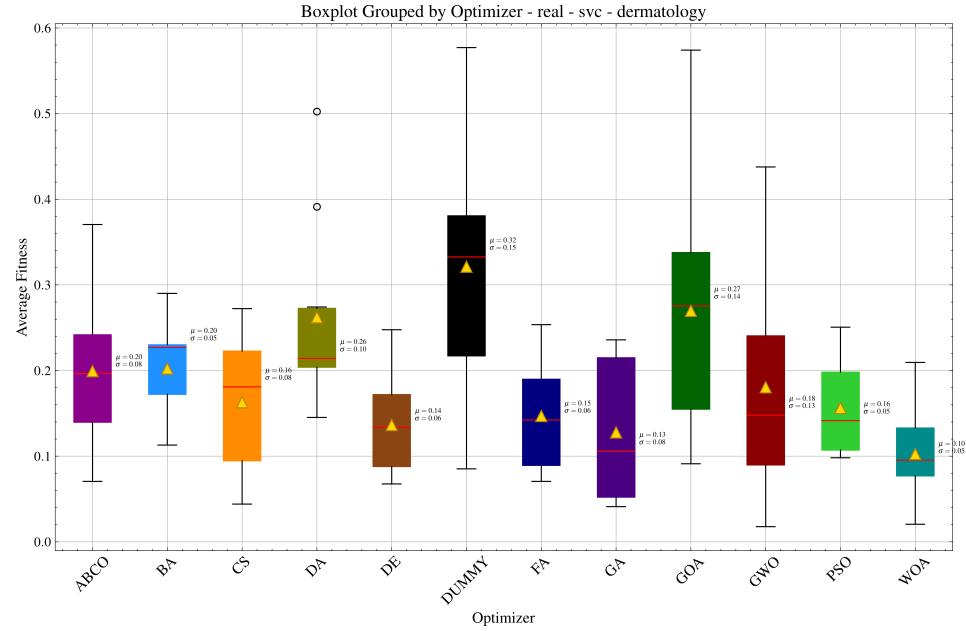


Figura A.18: Boxplot dermatology - svc - real

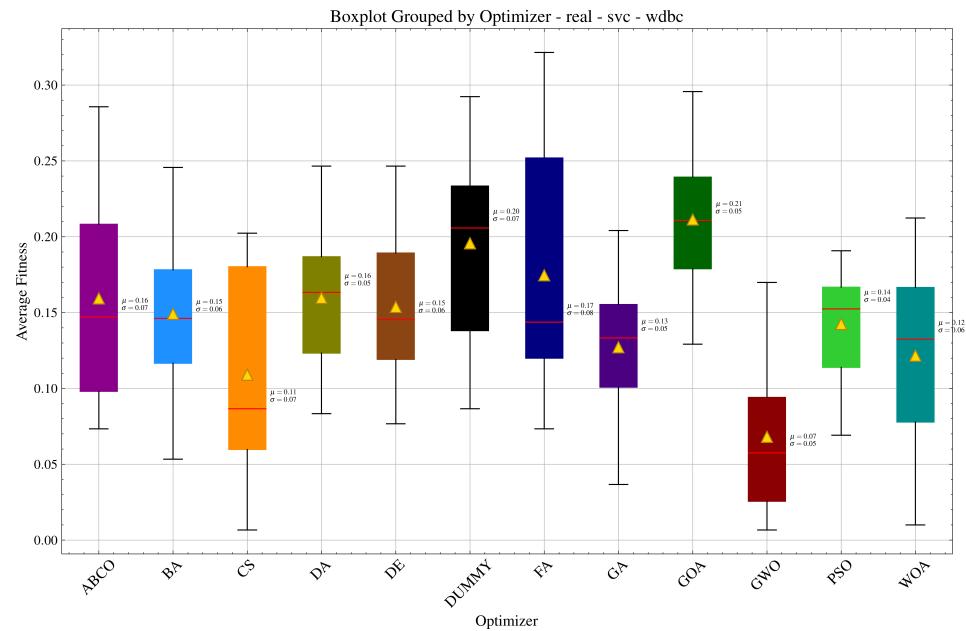
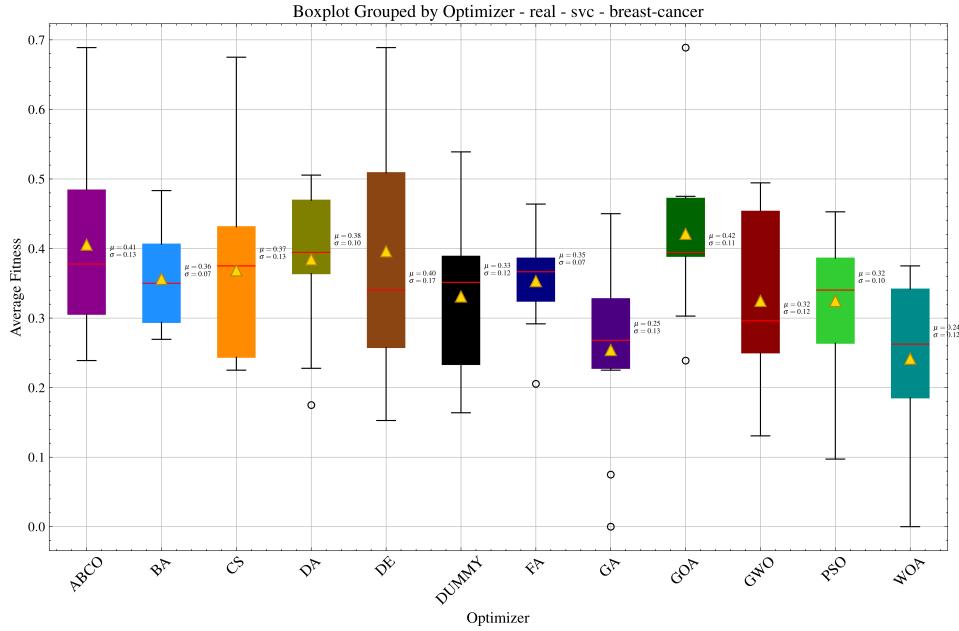
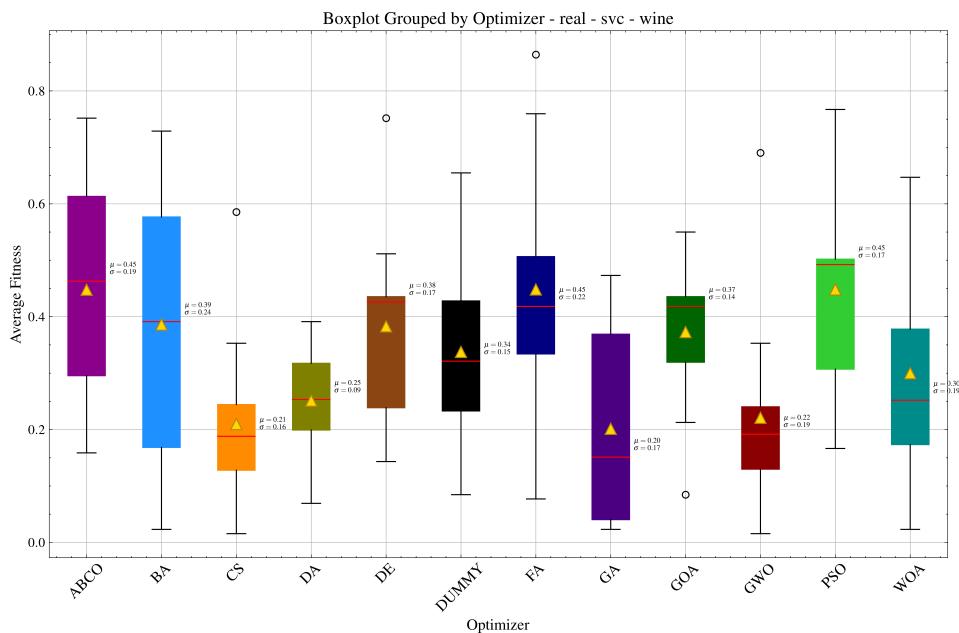


Figura A.19: Boxplot wdbc - svc - real

Figura A.20: *Boxplot* breast-cancer - svc - realFigura A.21: *Boxplot* wine - svc - real

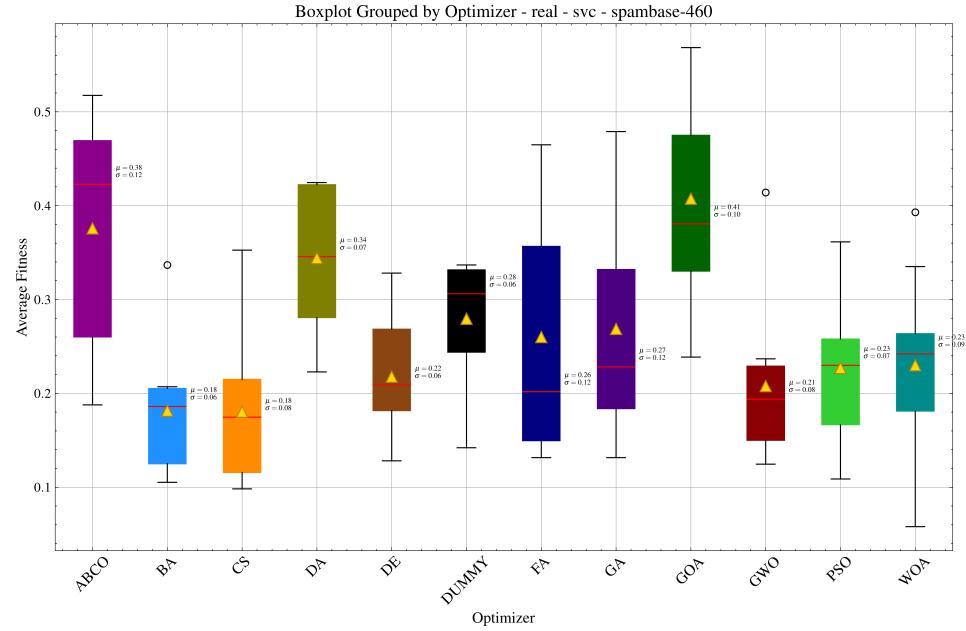


Figura A.22: Boxplot spambase-460 - svc - real

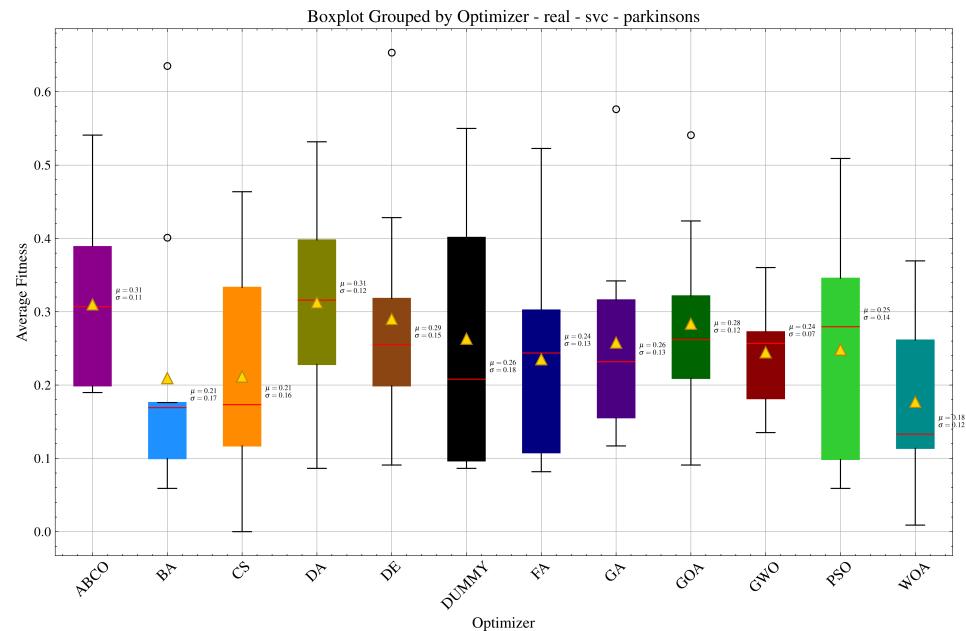


Figura A.23: Boxplot parkinsons - svc - real

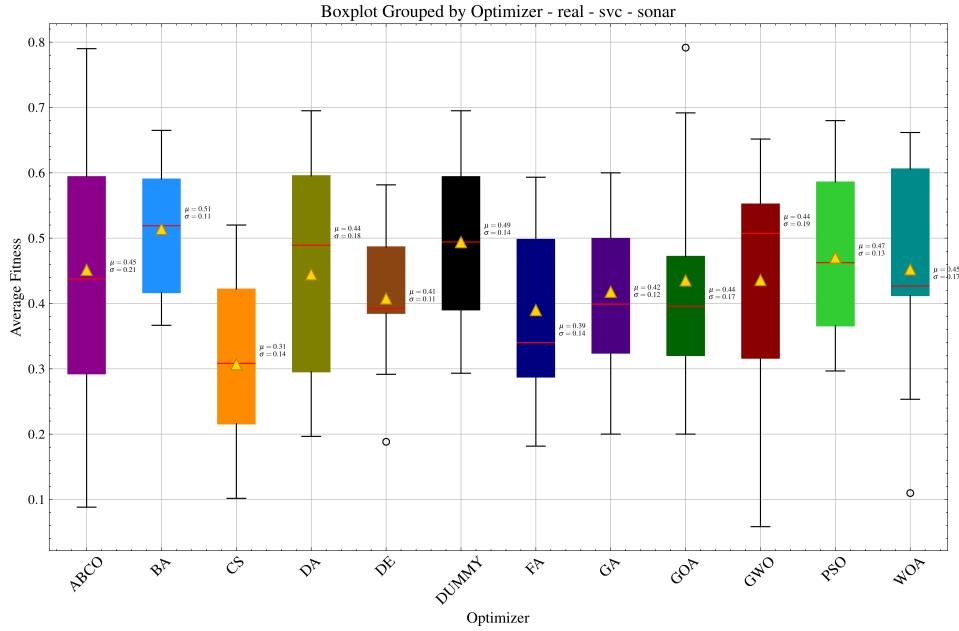


Figura A.24: Boxplot sonar - svc - real

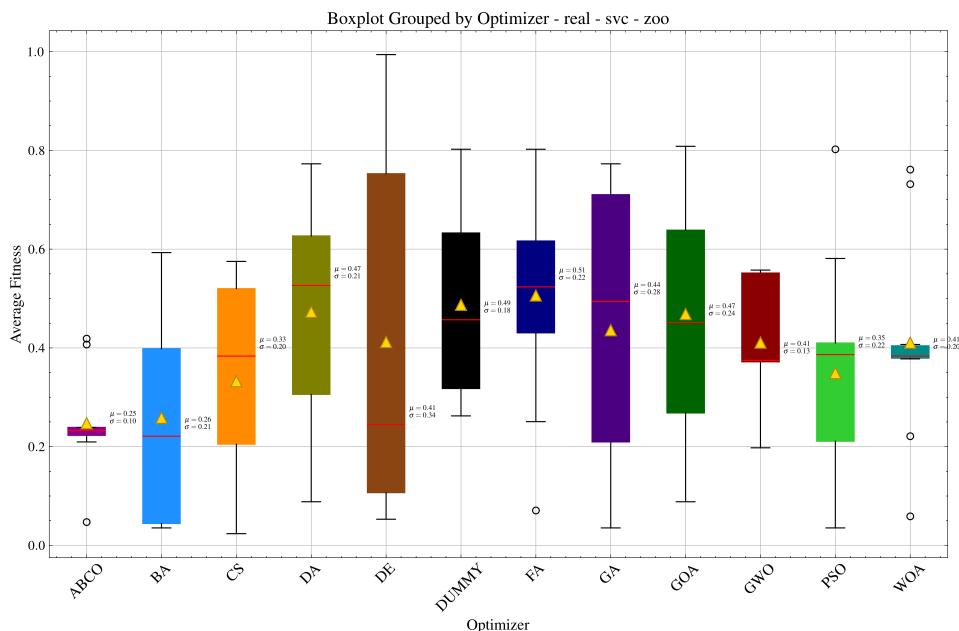


Figura A.25: Boxplot zoo - svc - real

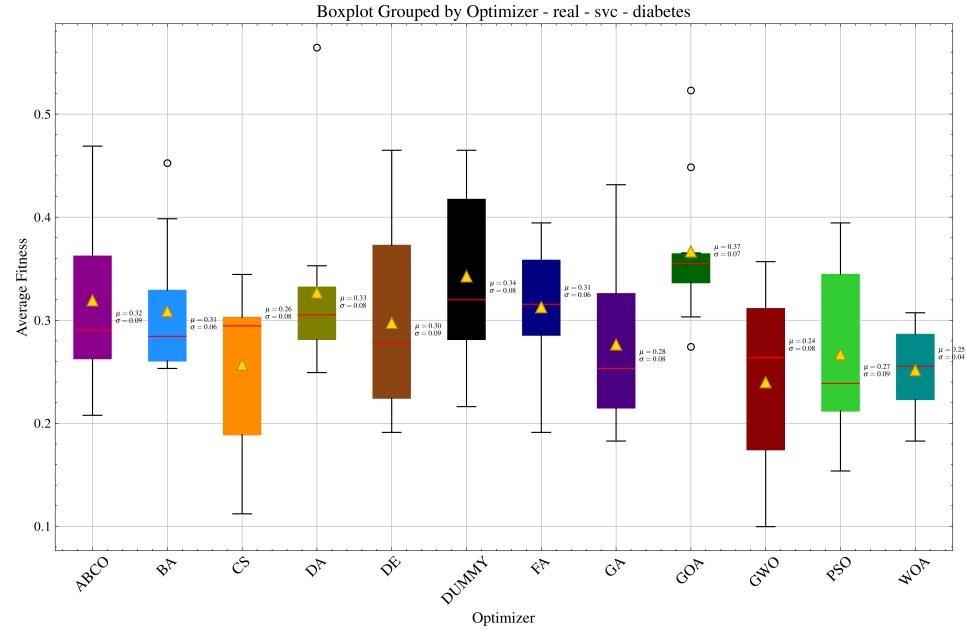


Figura A.26: Boxplot diabetes - svc - real

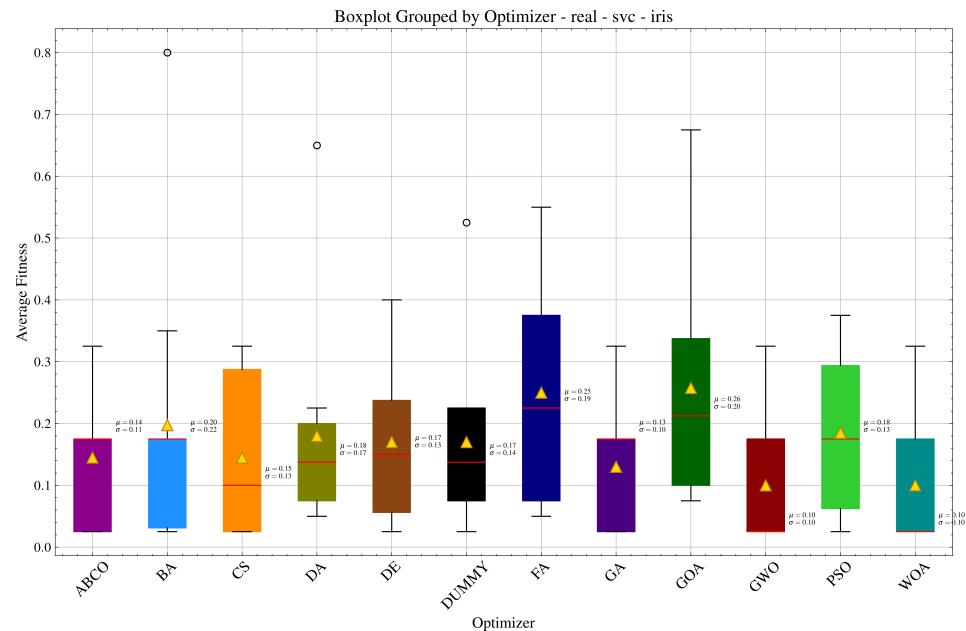
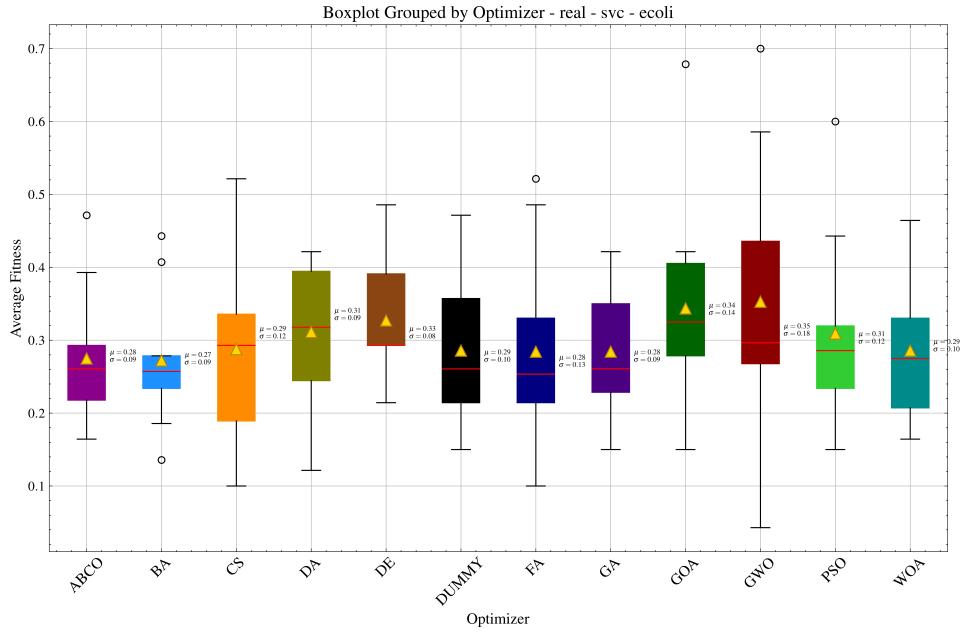
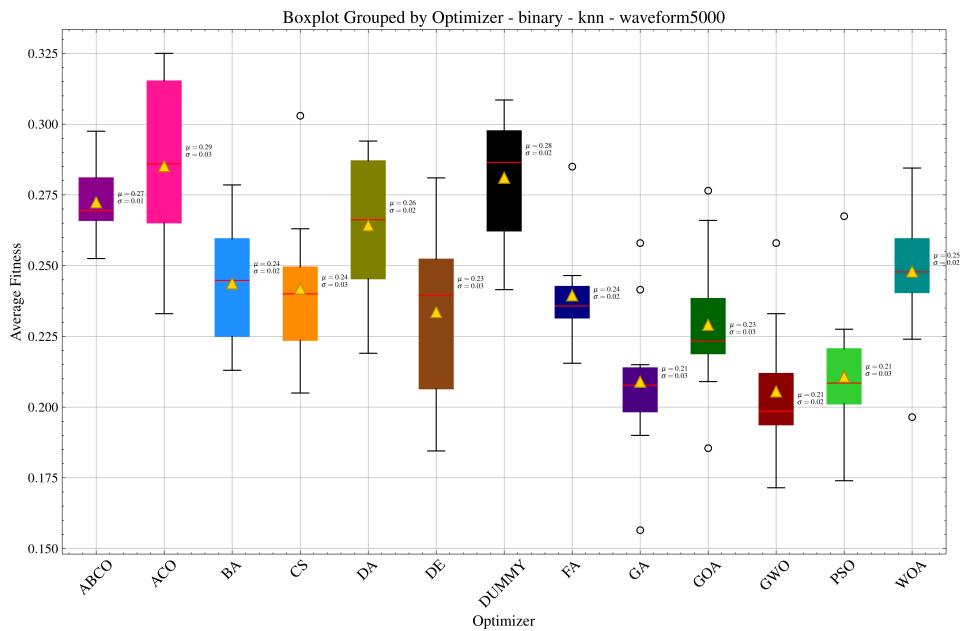
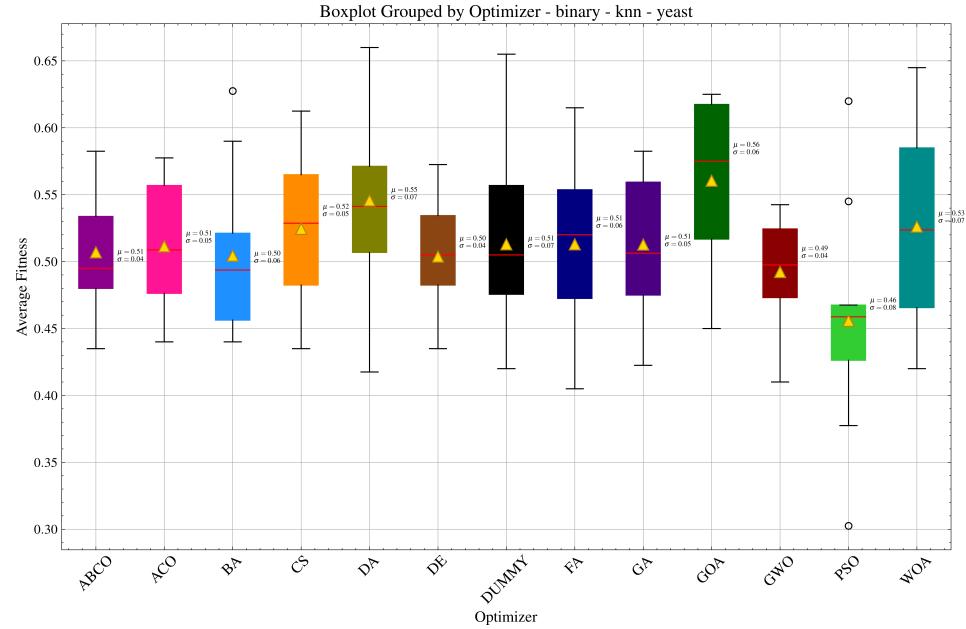
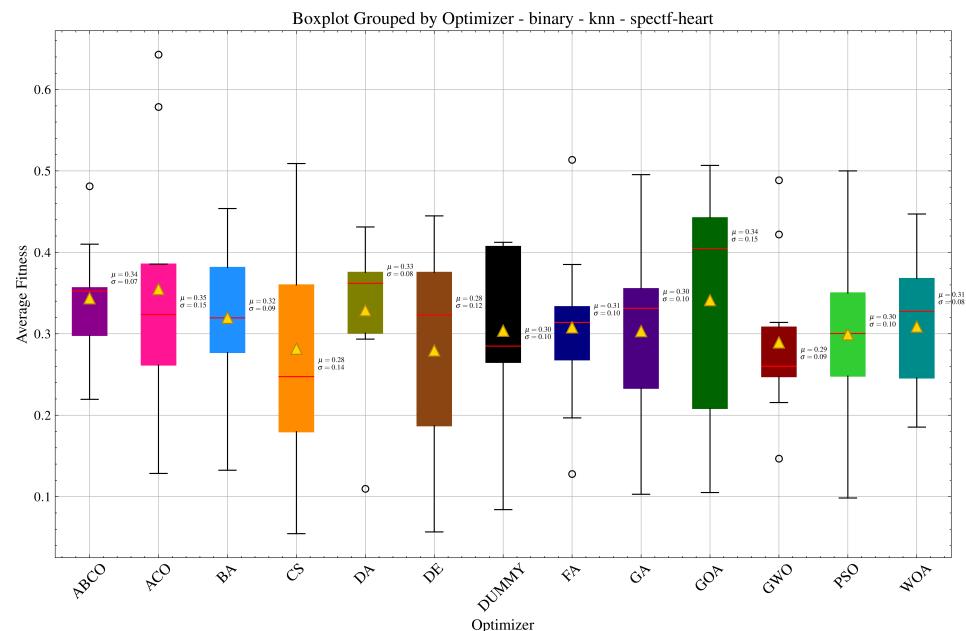


Figura A.27: Boxplot iris - svc - real

Figura A.28: *Boxplot ecoli - svc - real*Figura A.29: *Boxplot waveform5000 - knn - binary*

Figura A.30: *Boxplot* yeast - knn - binaryFigura A.31: *Boxplot* spectf-heart - knn - binary

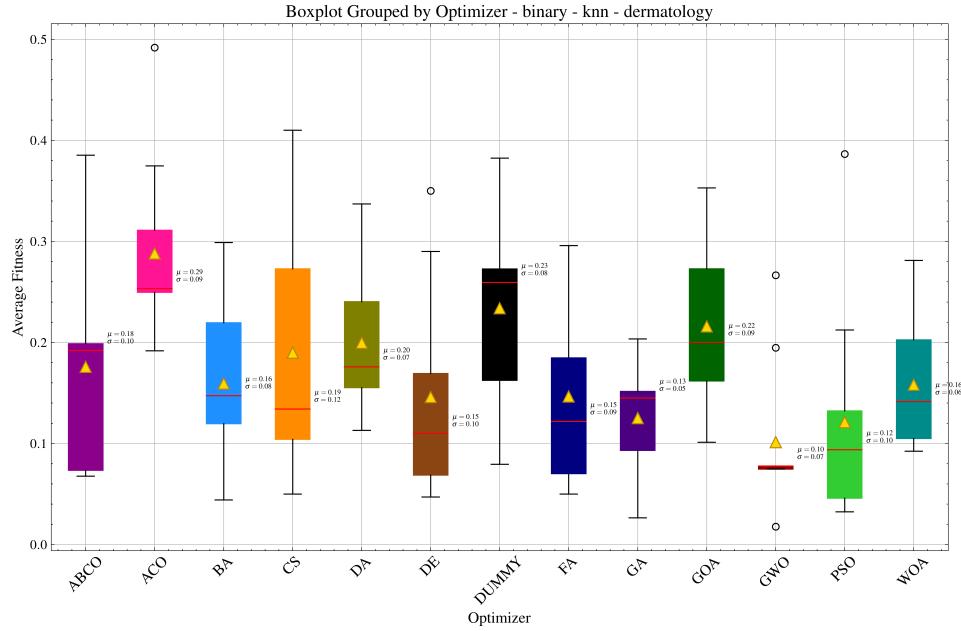


Figura A.32: *Boxplot* dermatology - knn - binary

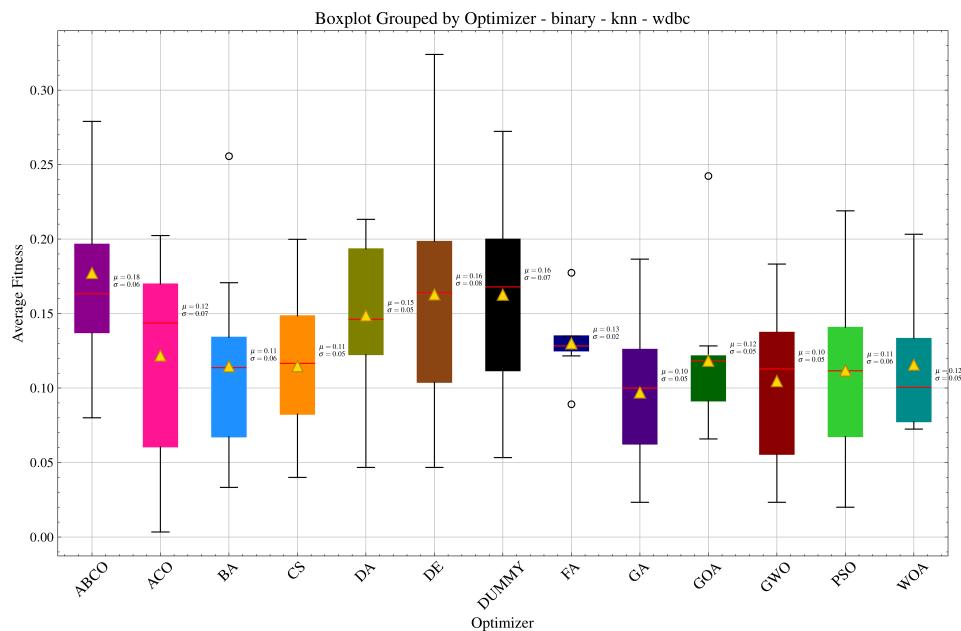


Figura A.33: *Boxplot* wdbc - knn - binary

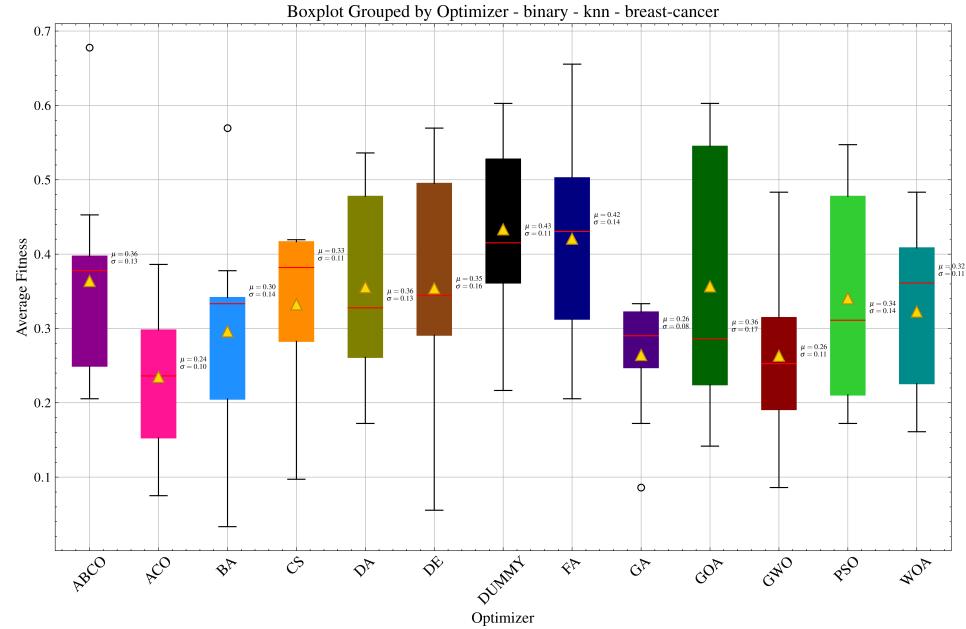


Figura A.34: Boxplot breast-cancer - knn - binary

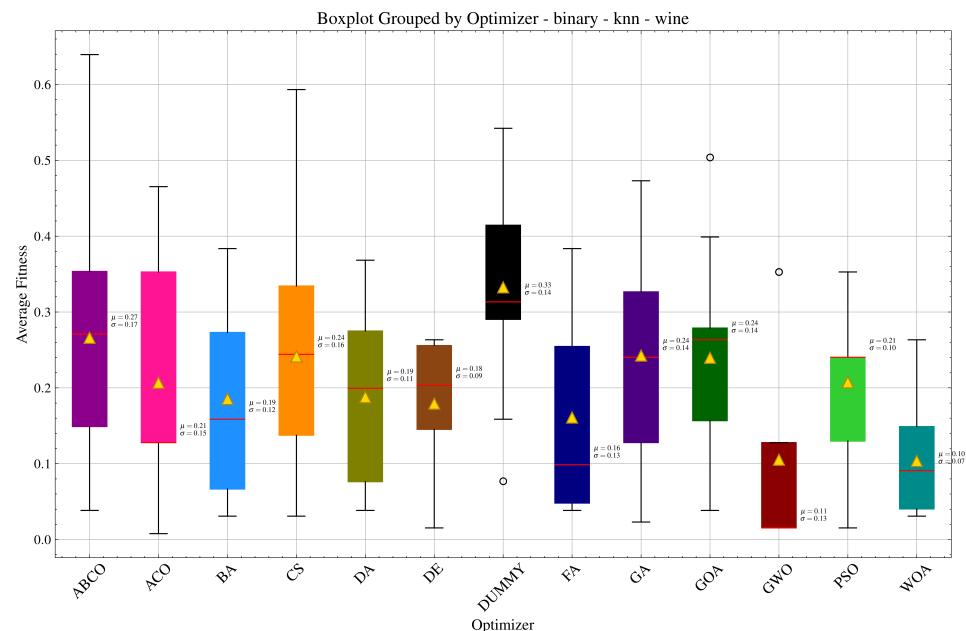


Figura A.35: Boxplot wine - knn - binary

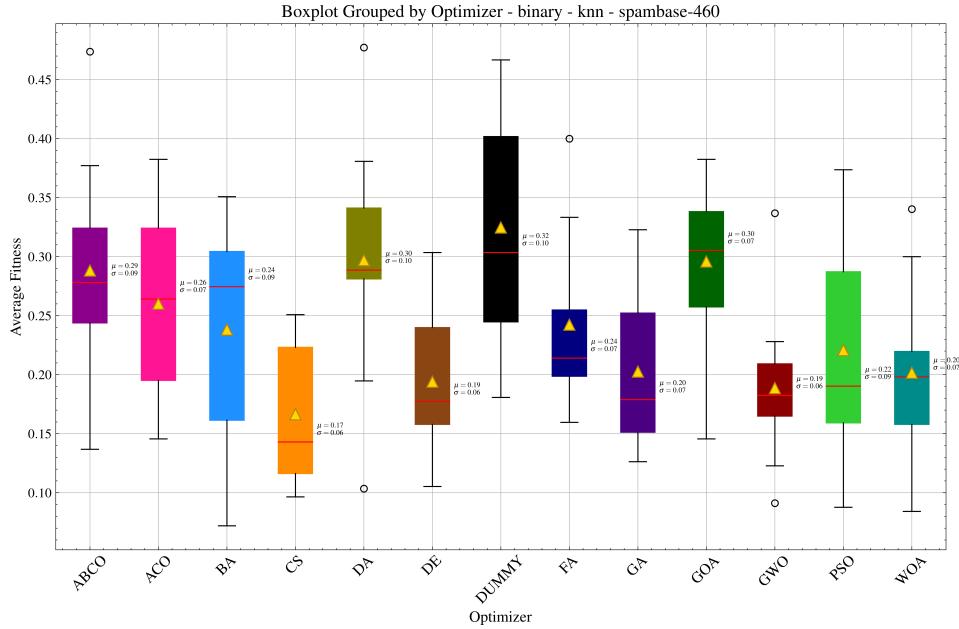


Figura A.36: *Boxplot* spambase-460 - knn - binary

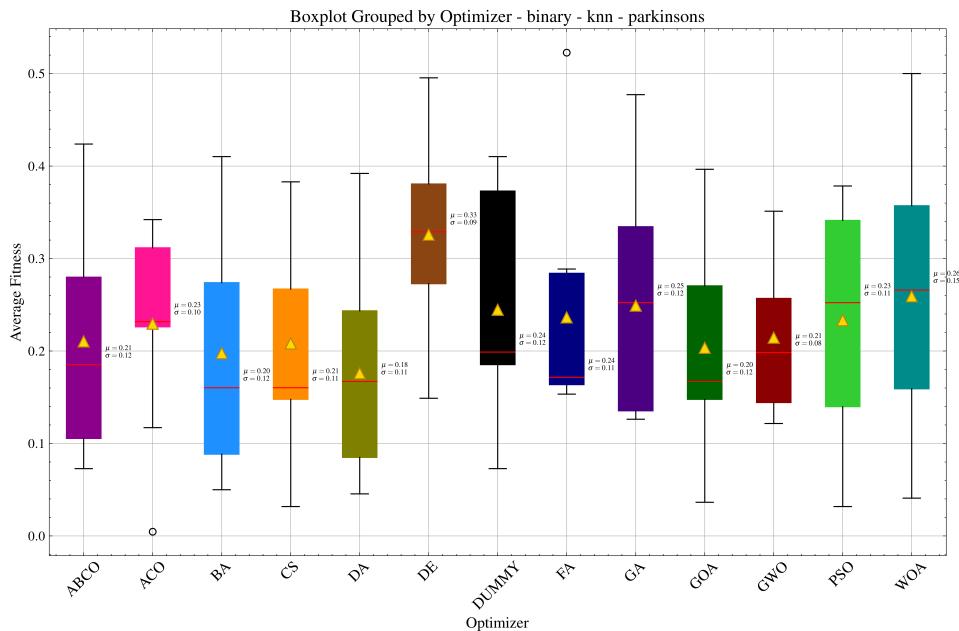


Figura A.37: *Boxplot* parkinsons - knn - binary

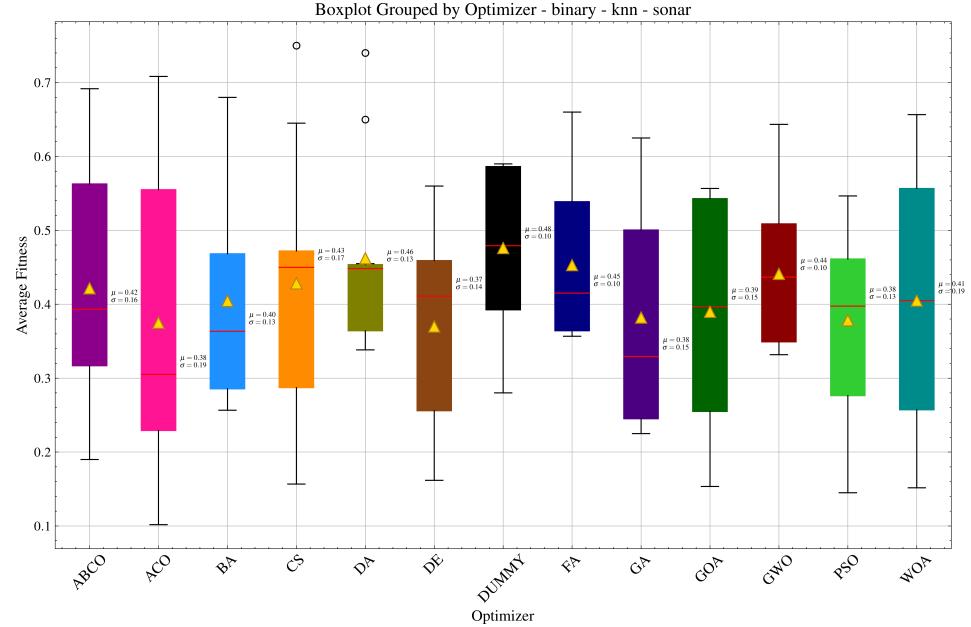


Figura A.38: Boxplot sonar - knn - binary

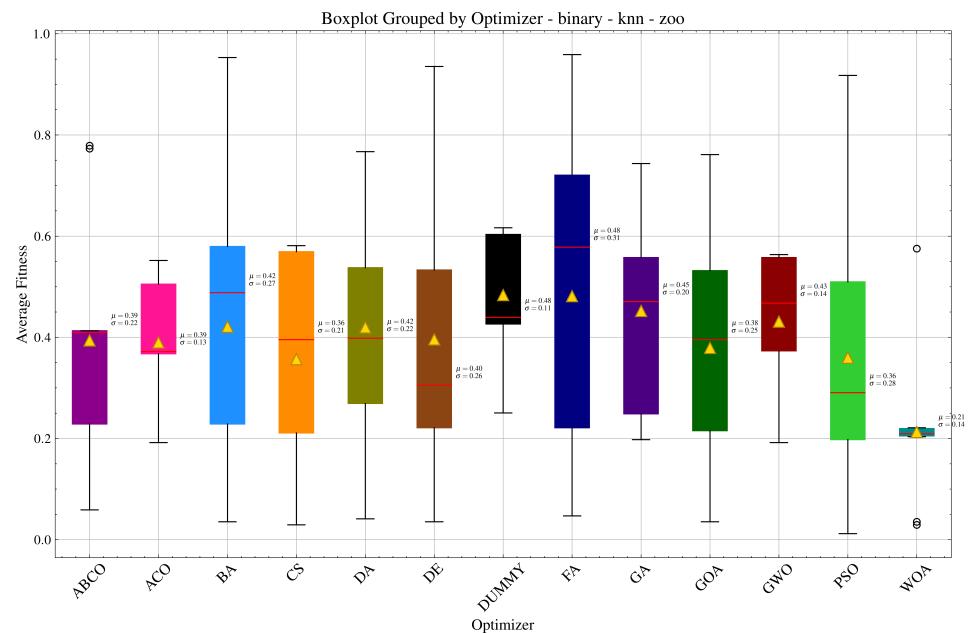
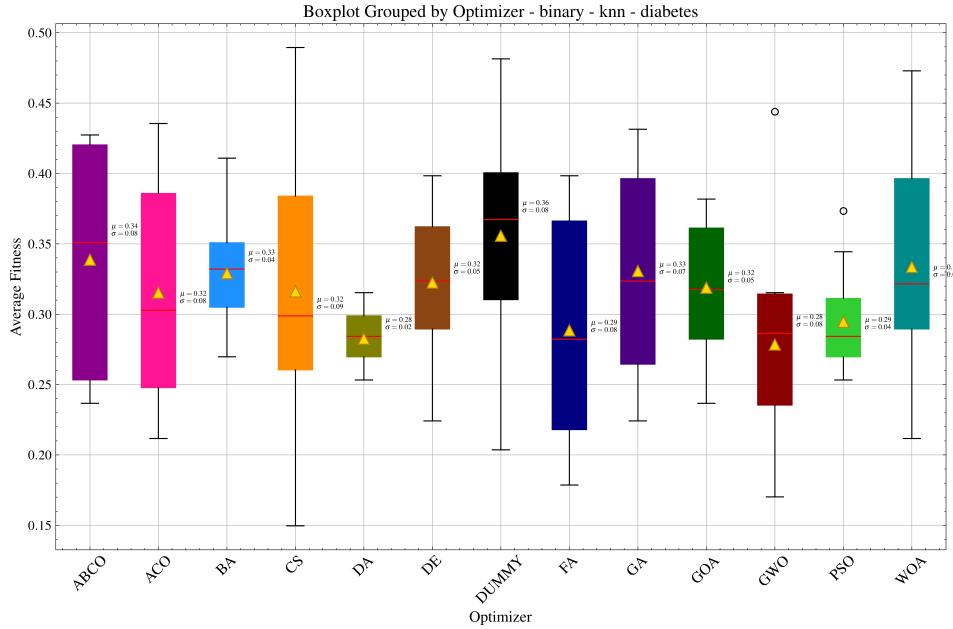
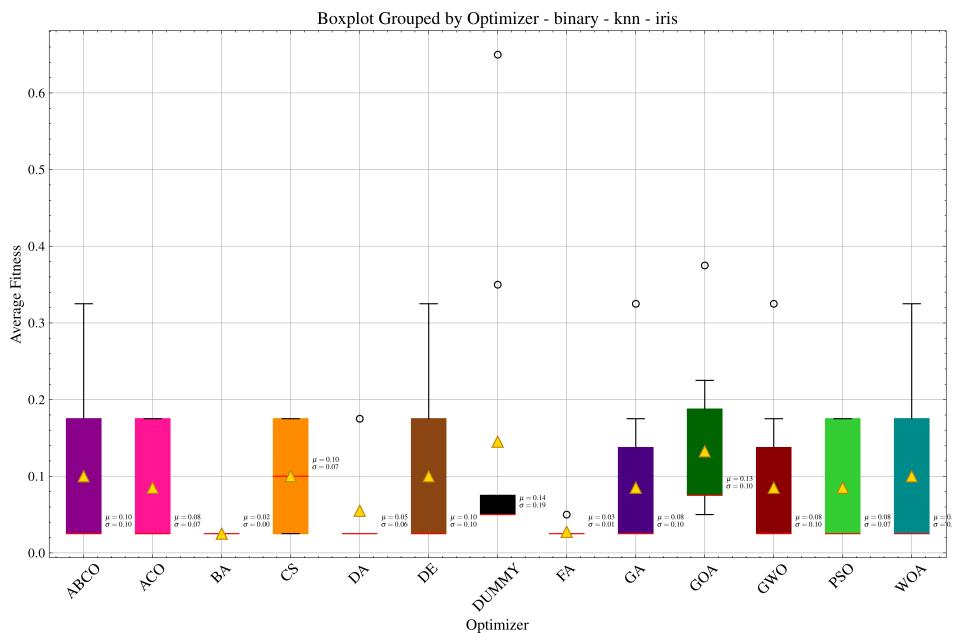


Figura A.39: Boxplot zoo - knn - binary

Figura A.40: *Boxplot* diabetes - knn - binaryFigura A.41: *Boxplot* iris - knn - binary

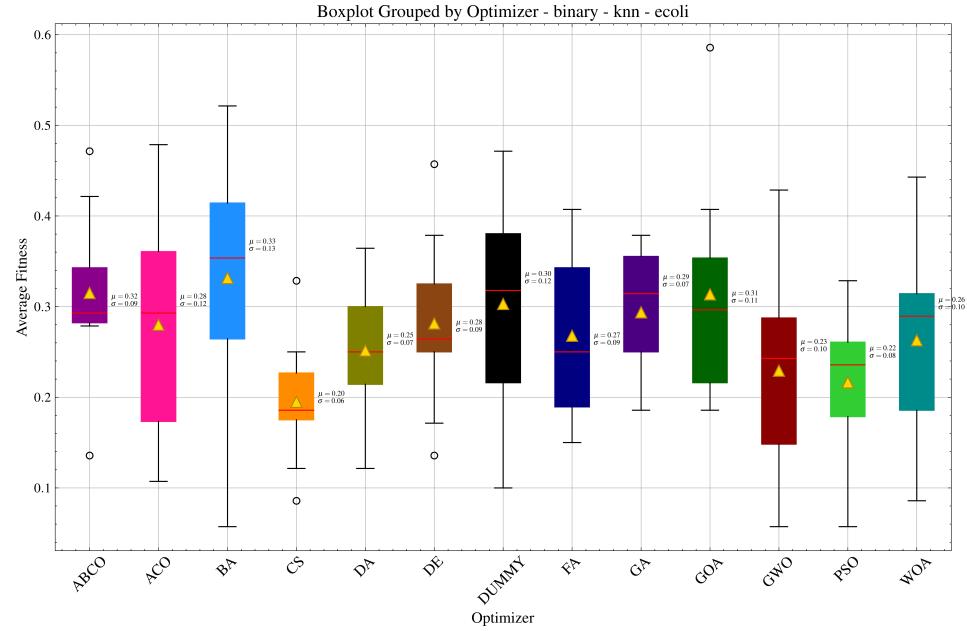


Figura A.42: Boxplot ecoli - knn - binary

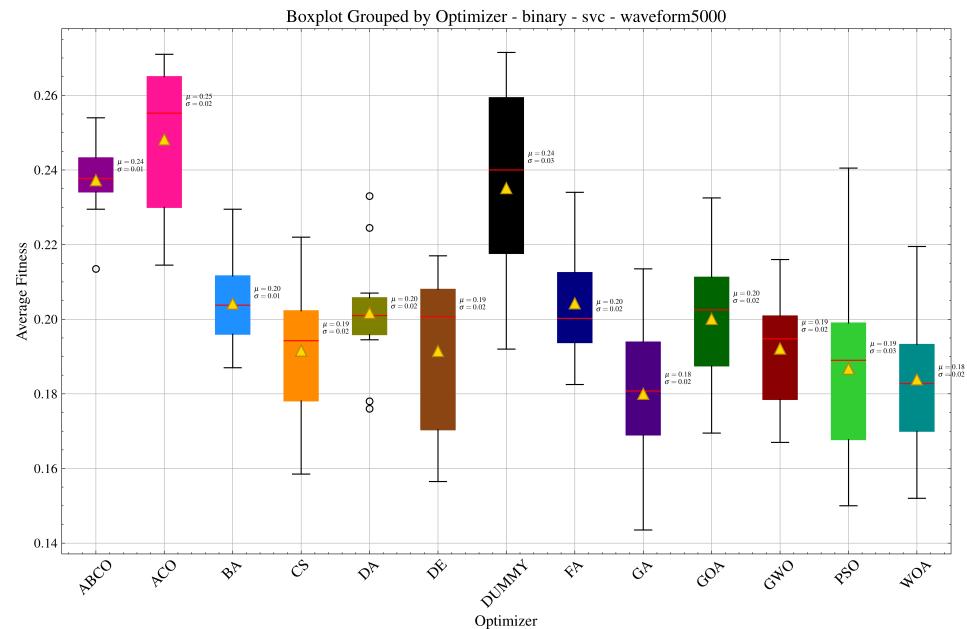
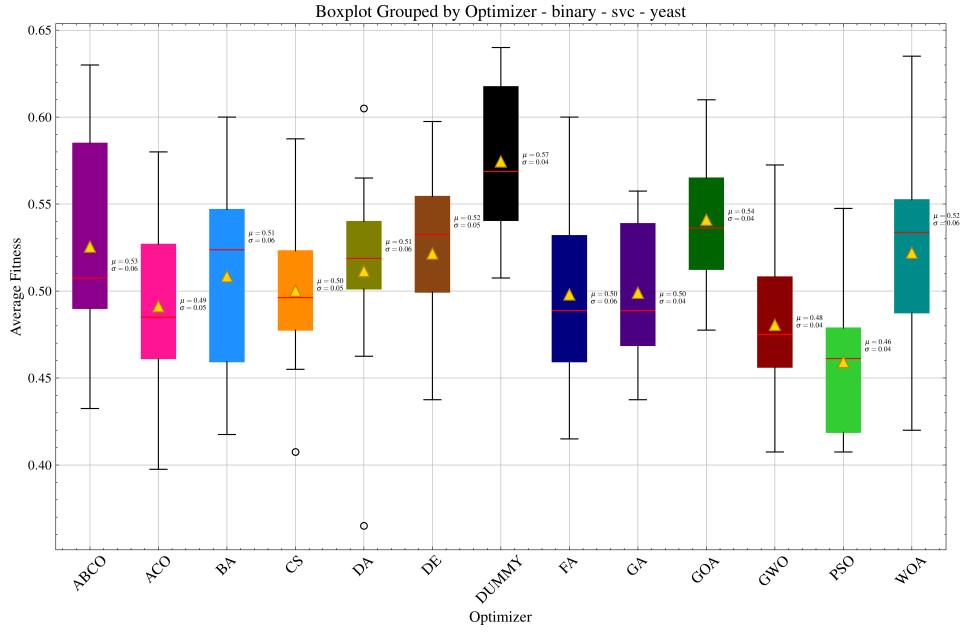
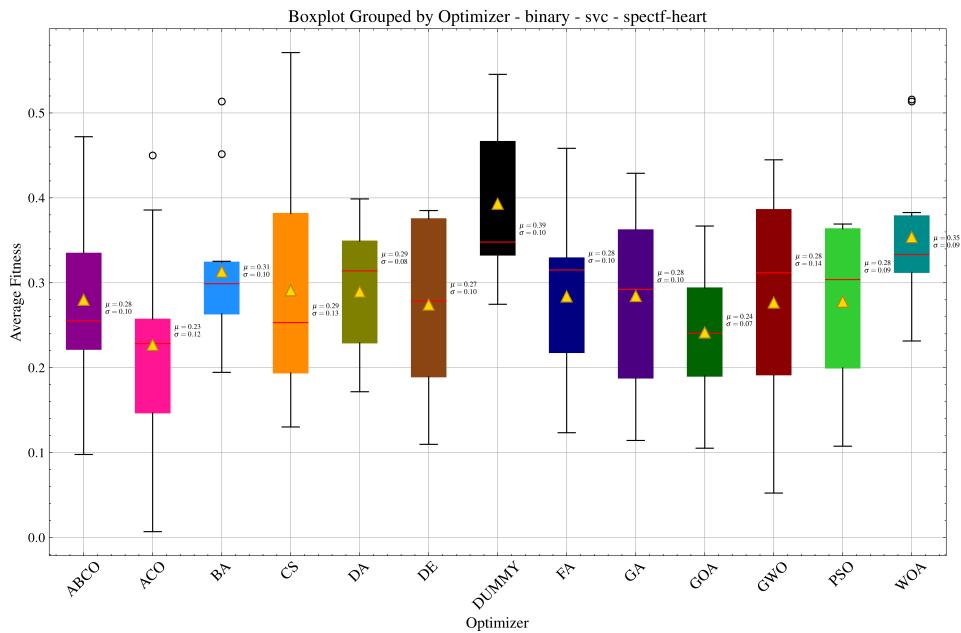


Figura A.43: Boxplot waveform5000 - svc - binary

Figura A.44: *Boxplot yeast - svc - binary*Figura A.45: *Boxplot spectf-heart - svc - binary*

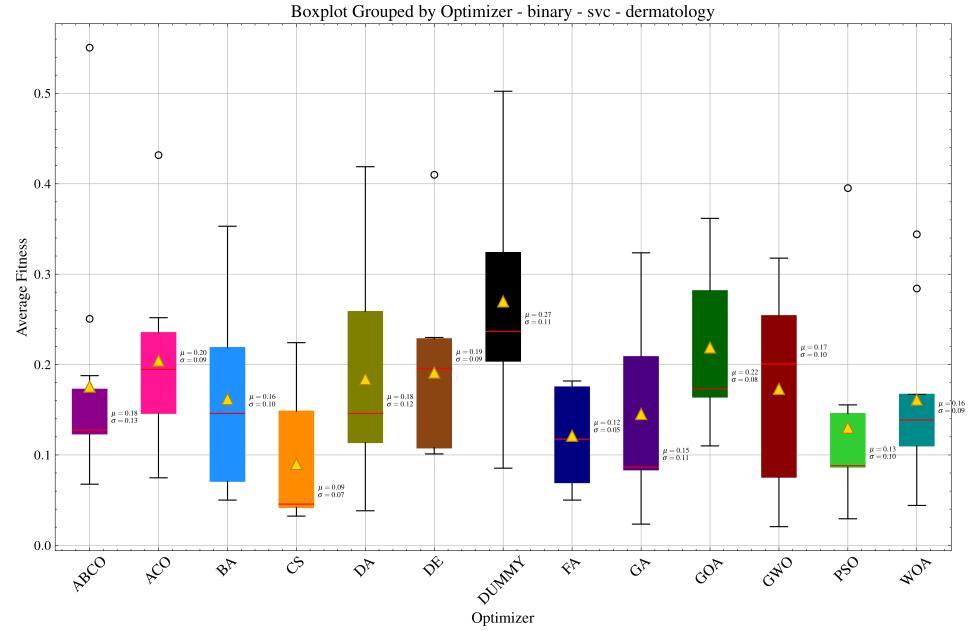


Figura A.46: Boxplot dermatology - svc - binary

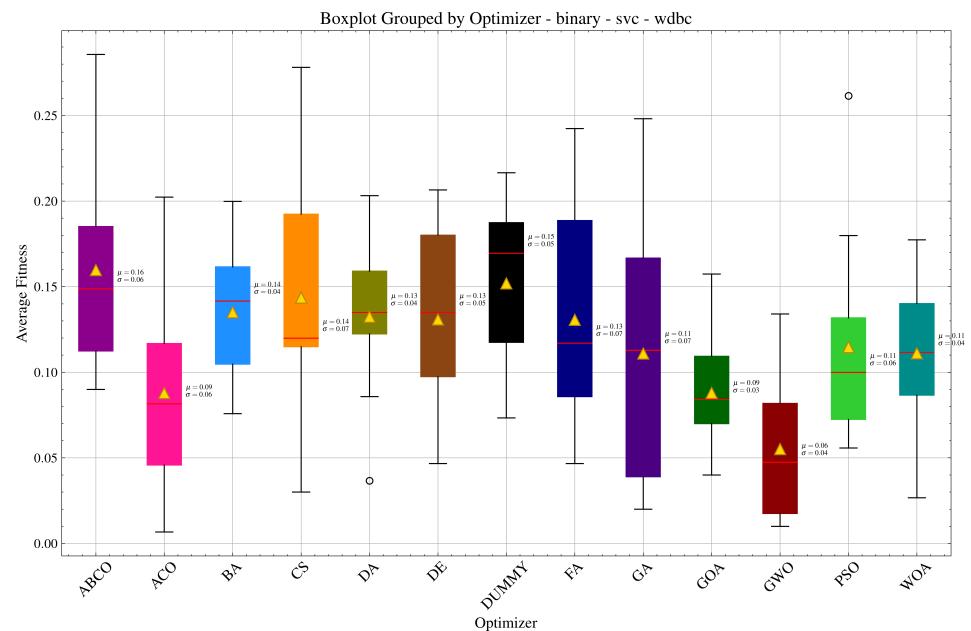
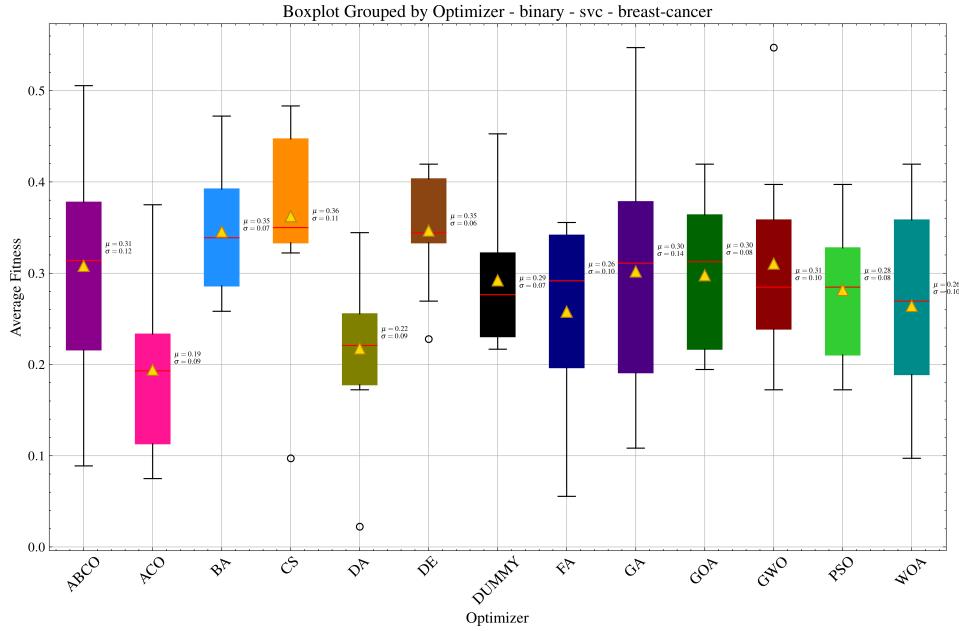
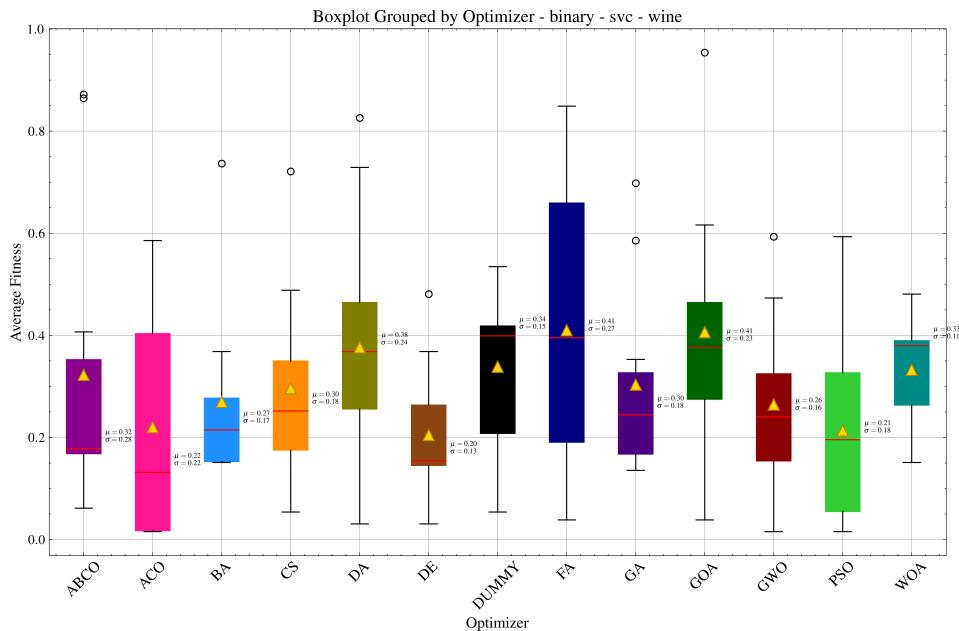
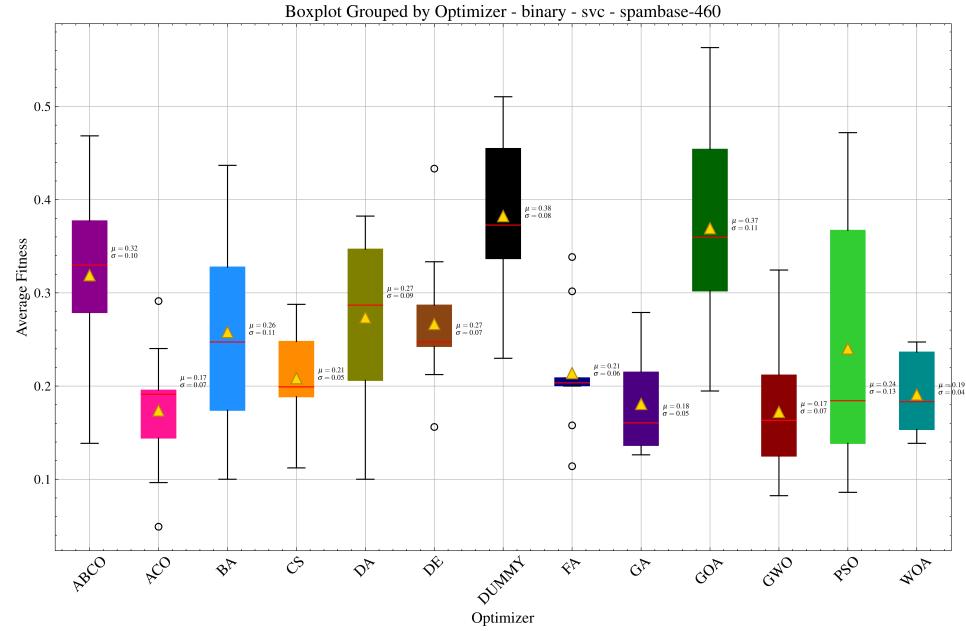
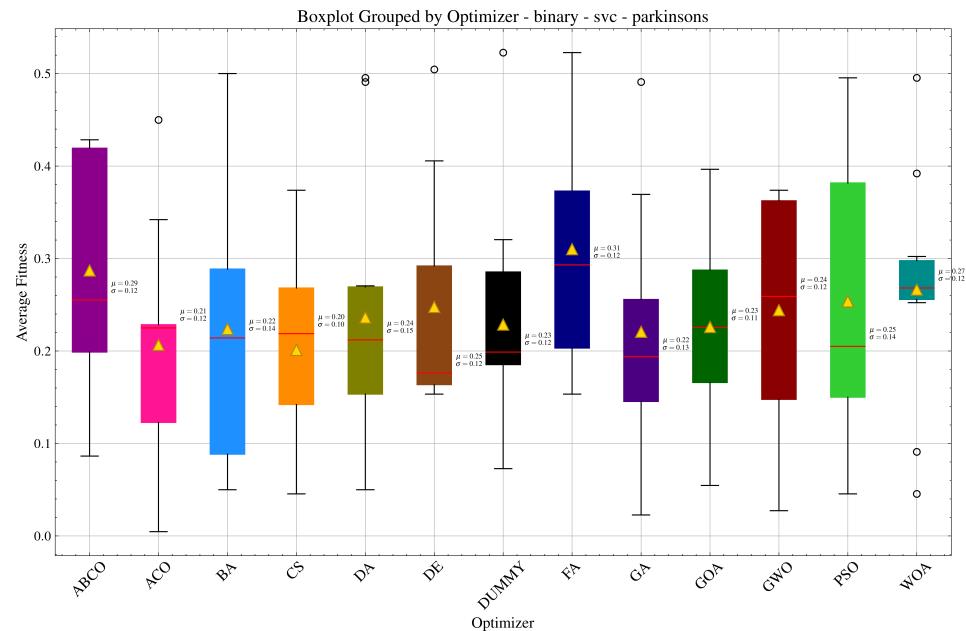


Figura A.47: Boxplot wdbc - svc - binary

Figura A.48: *Boxplot* breast-cancer - svc - binaryFigura A.49: *Boxplot* wine - svc - binary

Figura A.50: *Boxplot* spambase-460 - svc - binaryFigura A.51: *Boxplot* parkinsons - svc - binary

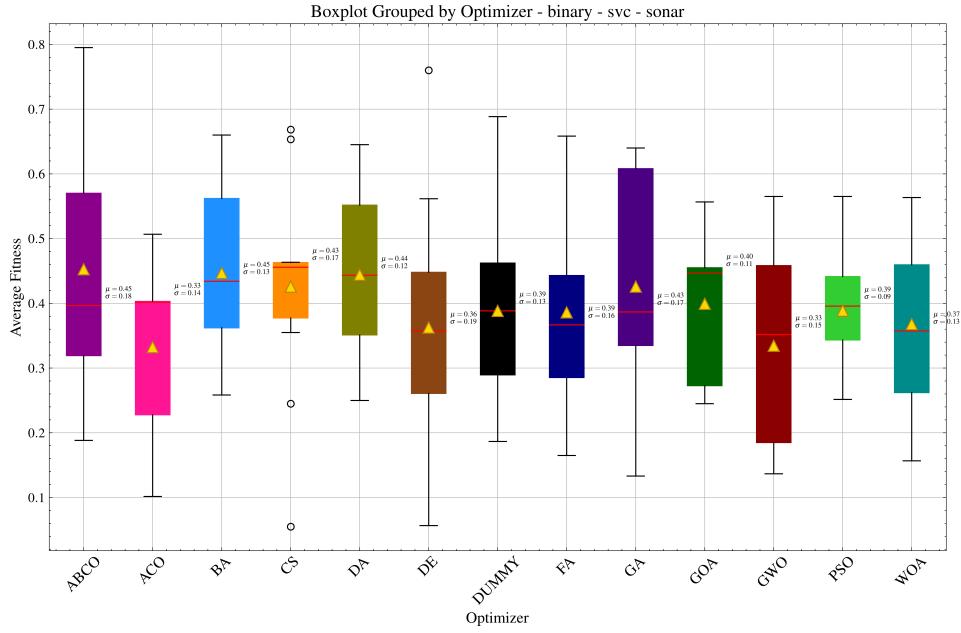


Figura A.52: Boxplot sonar - svc - binary

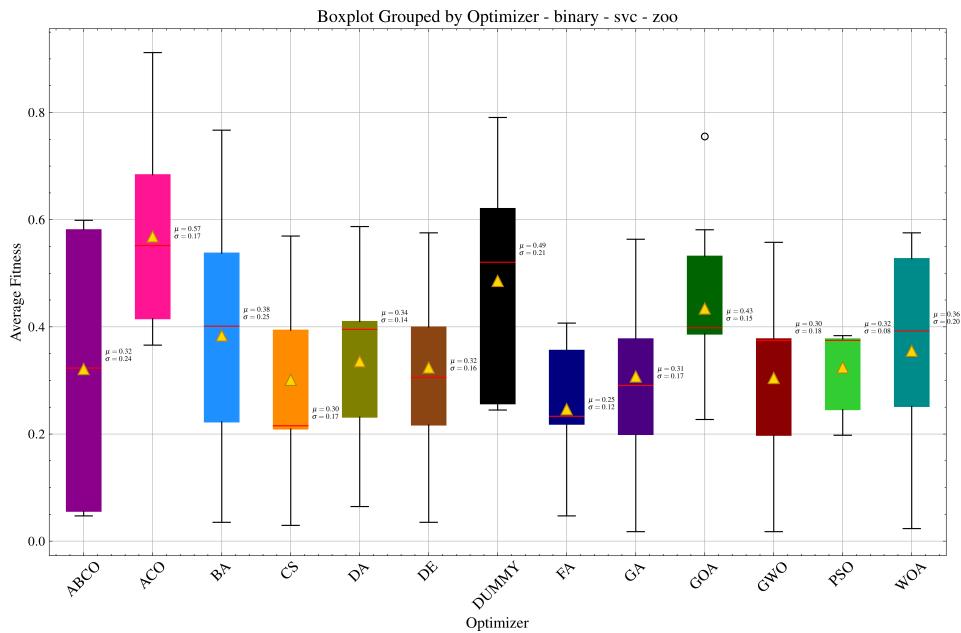


Figura A.53: Boxplot zoo - svc - binary

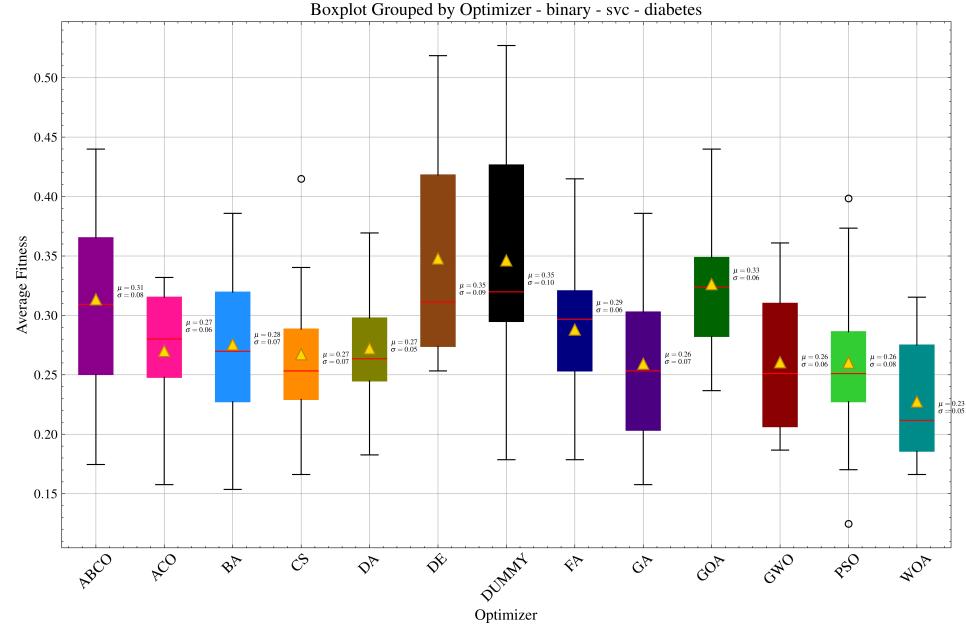


Figura A.54: Boxplot diabetes - svc - binary

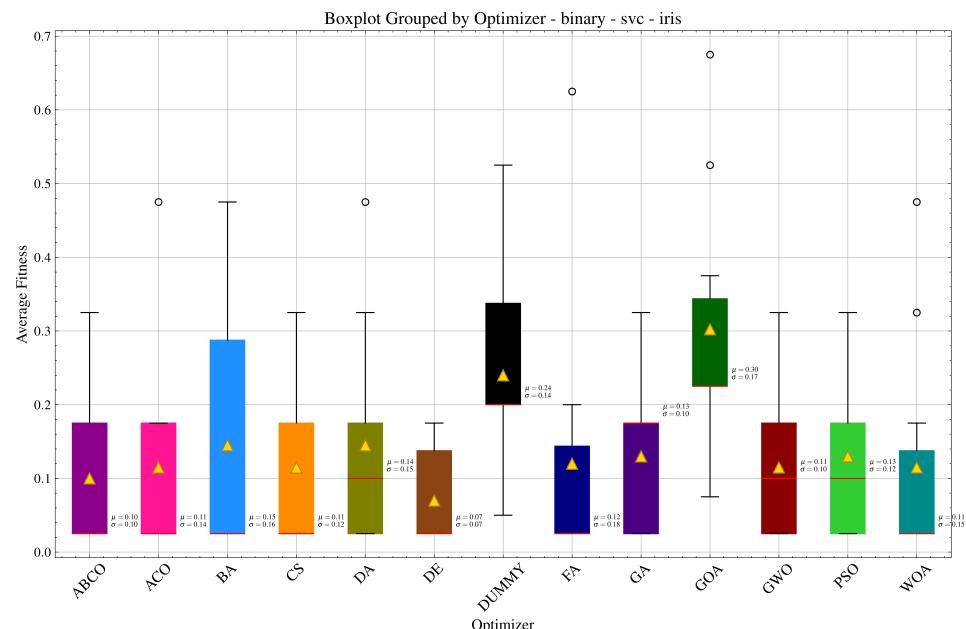


Figura A.55: Boxplot iris - svc - binary

	abco	aco	ba	cs	da	de	dummy	fa	ga	goa	gwo	pso	woa
abco	-	+ (0.083)	+ (0.008)	+ (0.012)	+ (0.151)	+ (0.020)	- (0.018)	+ (0.105)	+ (0.012)	+ (0.252)	+ (0.003)	+ (0.001)	
aco	- (0.083)	-	= (0.514)	= (0.561)	= (0.679)	= (0.934)	- (0.012)	+ (0.443)	= (0.754)	- (0.389)	+ (0.055)	+ (0.081)	
ba	- (0.008)	= (0.514)	-	= (0.851)	- (0.121)	= (0.900)	- (0.001)	= (0.890)	= (0.670)	- (0.147)	+ (0.055)	+ (0.229)	
cs	- (0.012)	= (0.561)	= (0.851)	-	- (0.320)	= (0.851)	- (1.831E-04)	= (0.679)	= (0.890)	- (0.147)	+ (0.135)	= (0.572)	
da	- (0.151)	= (0.679)	+ (0.121)	+ (0.320)	-	+ (0.334)	- (0.005)	+ (0.359)	+ (0.208)	= (0.599)	+ (0.005)	+ (0.191)	
de	- (0.020)	= (0.934)	= (0.900)	= (0.851)	- (0.334)	-	- (0.007)	= (0.950)	+ (0.454)	- (0.229)	+ (0.026)	+ (0.055)	
dummy	+ (0.018)	+ (0.012)	+ (0.001)	+ (1.831E-04)	+ (0.005)	+ (0.007)	-	+ (0.002)	+ (0.002)	+ (6.104E-05)	+ (0.012)	+ (0.001)	
fa	- (0.105)	- (0.443)	= (0.890)	= (0.679)	- (0.359)	= (0.950)	- (0.002)	-	= (0.660)	- (0.330)	+ (0.022)	+ (0.135)	
ga	- (0.012)	= (0.754)	= (0.670)	= (0.890)	- (0.208)	- (0.454)	- (0.002)	= (0.660)	-	- (0.050)	+ (0.038)	- (0.389)	
goa	- (0.252)	+ (0.389)	+ (0.147)	+ (0.147)	= (0.599)	+ (0.229)	- (0.012)	+ (0.330)	+ (0.050)	- (0.010)	+ (0.007)	+ (0.022)	
gwo	- (0.003)	- (0.055)	- (0.055)	- (0.135)	- (0.005)	- (0.026)	- (6.104E-05)	- (0.022)	- (0.038)	- (0.010)	- (0.346)	- (0.064)	
pso	- (0.001)	- (0.081)	- (0.277)	- (0.191)	- (0.030)	- (0.055)	- (6.104E-05)	- (0.135)	- (0.272)	- (0.007)	+ (0.346)	- (0.561)	
woa	- (0.012)	- (0.229)	= (0.572)	= (0.802)	- (0.095)	+ (0.233)	- (0.001)	= (0.639)	+ (0.389)	- (0.022)	+ (0.064)	= (0.561)	
													-

Tabla A.1: P-valores en fitness - knn - binario

	abco	aco	ba	cs	da	de	dummy	fa	ga	goa	gwo	pso	woa
abco	-	+ (0.083)	+ (0.256)	+ (0.005)	+ (0.229)	+ (0.035)	- (0.135)	+ (0.041)	+ (0.012)	= (0.804)	+ (0.002)	+ (0.001)	+ (0.038)
aco	- (0.083)	-	- (0.303)	= (0.950)	- (0.208)	= (0.679)	- (0.007)	- (0.359)	= (0.609)	- (0.073)	+ (0.490)	= (0.887)	= (0.802)
ba	- (0.256)	+ (0.303)	-	+ (0.015)	= (0.777)	+ (0.421)	- (0.003)	+ (0.379)	+ (0.044)	- (0.222)	+ (0.004)	+ (0.007)	+ (0.208)
cs	- (0.005)	= (0.950)	- (0.015)	-	- (0.026)	- (0.451)	- (0.003)	- (0.421)	= (1.000)	- (0.055)	+ (0.364)	= (0.804)	- (0.451)
da	- (0.229)	+ (0.208)	= (0.777)	+ (0.026)	-	+ (0.277)	- (0.008)	+ (0.454)	+ (0.048)	- (0.167)	+ (0.005)	+ (0.033)	+ (0.208)
de	- (0.035)	= (0.679)	- (0.421)	+ (0.451)	- (0.277)	-	- (0.003)	= (0.802)	= (0.599)	- (0.277)	+ (0.109)	+ (0.286)	= (1.000)
dummy	+ (0.135)	+ (0.007)	+ (0.003)	+ (0.003)	+ (0.008)	+ (0.003)	-	+ (0.015)	+ (0.002)	+ (0.107)	+ (3.052E-04)	+ (4.272E-04)	+ (0.001)
fa	- (0.041)	+ (0.359)	- (0.379)	+ (0.421)	- (0.454)	= (0.802)	- (0.015)	-	= (0.820)	- (0.188)	+ (0.121)	+ (0.252)	= (0.599)
ga	- (0.012)	= (0.609)	- (0.044)	-	- (0.048)	= (0.599)	- (0.002)	= (0.820)	-	- (0.073)	+ (0.252)	+ (0.451)	= (0.706)
goa	= (0.804)	+ (0.073)	+ (0.222)	+ (0.055)	+ (0.167)	+ (0.277)	- (0.107)	+ (0.188)	+ (0.073)	-	+ (0.008)	+ (0.026)	+ (0.055)
gwo	- (0.002)	- (0.490)	- (0.004)	- (0.364)	- (0.005)	- (0.109)	- (3.052E-04)	- (0.121)	- (0.252)	- (0.008)	-	= (0.599)	- (0.073)
pso	- (0.001)	= (0.887)	- (0.007)	= (0.804)	- (0.033)	- (0.286)	- (4.272E-04)	- (0.252)	- (0.451)	- (0.026)	= (0.599)	-	= (0.570)
woa	- (0.038)	= (0.802)	- (0.208)	+ (0.451)	- (0.208)	- (0.001)	= (0.599)	= (0.706)	- (0.055)	+ (0.073)	= (0.570)	-	-

Tabla A.2: P-valores en *fitness* - *svc* - binario

	abco	aco	ba	cs	da	de	dummy	fa	ga	goa	gwo	pso	woa
F01	11	1	4	6	9	8	13	12	3	10	2	7	5
F02	8	13	7	9	10	4.5	12	4.5	3	11	1	2	6
F03	12	5	9	6	2	8	13	3	10	7	1	4	11
F04	12	7	13	1	4	8	10	6	9	11	3	2	5
F05	13	5	7	11	10	8	12	1	3	6	4	9	2
F06	9.5	5.5	1	9.5	3	9.5	13	2	5.5	12	5.5	5.5	9.5
F07	5	7	2	4	1	13	10	9	11	3	6	8	12
F08	8	2	6.5	9	12	1	13	11	4	5	10	3	6.5
F09	10	9	7	1	12	3	13	8	5	11	2	6	4
F10	12	13	9	2	10	1	6	7	5	11	3	4	8
F11	11	13	8	7	10	5	12	6	2	4	1	3	9
F12	13	8	4.5	4.5	10	11.5	11.5	9	1	7	2	3	6
F13	12	7.5	5	10	6	4	13	3	11	9	2	7.5	1
F14	5	6	3.5	10	12	3.5	8	8	8	13	2	1	11
F15	6	5	9	2	8	7	13	12	11	4	10	3	1
Mean	9.833	7.133	6.367	6.133	7.933	6.333	11.500	6.767	6.100	8.267	3.633	4.533	6.467

Tabla A.3: Ranking de los algoritmos en *fitness* - knn - binario

	abco	aco	ba	cs	da	de	dummy	fa	ga	goa	gwo	pso	woa
F01	9	1	11	13	2	12	6	3	8	7	10	5	4
F02	8	11	6	1	9	10	13	2	4	12	7	3	5
F03	10	6	8	5	7	13	12	9	2	11	4	3	1
F04	7	13	3	1	8.5	5	12	10	11	8.5	4	6	2
F05	13	10	8	1	12	4	11	6	2	7	5	3	9
F06	2	4.5	10.5	4.5	10.5	1	12	7	8.5	13	5	4.5	4.5
F07	12	2	4	1	7	9	6	13	3	5	8	10	11
F08	13	1	12	9.5	11	3	6	5	9.5	8	2	7	4
F09	11	2	8	5	10	9	13	6	3	12	1	7	4
F10	6	1	11	10	9	3	13	7	8	2	4	5	12
F11	12	13	9.5	5	8	5	11	9.5	1	7	5	3	2
F12	13	2	10	11	9	7.5	12	7.5	4.5	3	1	6	4.5
F13	8	3	5	6	11	1	10	13	7	12	4	2	9
F14	11	3	7	6	8	9	13	4	5	12	2	1	10
F15	5	13	10	2	8	6.5	12	1	4	11	3	6.5	9
Mean	9.333	5.700	8.200	5.400	8.667	6.533	10.800	6.867	5.367	8.700	4.300	5.067	6.067

Tabla A.4: Ranking de los algoritmos en *fitness* - svc - binario

	abco	aco	ba	cs	da	de	dummy	fa	ga	gao	gwo	pso	woa
F01	3.640e-01	2.350e-01	2.960e-01	3.320e-01	3.560e-01	3.540e-01	4.330e-01	4.210e-01	2.640e-01	3.570e-01	2.630e-01	3.410e-01	3.230e-01
F02	1.760e-01	2.880e-01	1.590e-01	1.900e-01	2.000e-01	1.460e-01	2.340e-01	1.460e-01	2.160e-01	1.010e-01	1.210e-01	1.210e-01	1.580e-01
F03	3.390e-01	3.150e-01	3.290e-01	3.160e-01	2.830e-01	3.230e-01	3.560e-01	2.890e-01	3.310e-01	3.190e-01	2.780e-01	2.950e-01	3.340e-01
F04	3.150e-01	2.800e-01	3.310e-01	1.950e-01	2.510e-01	2.810e-01	3.030e-01	2.680e-01	2.940e-01	3.140e-01	2.290e-01	2.160e-01	2.630e-01
F05	3.110e-01	2.340e-01	2.490e-01	3.080e-01	2.710e-01	2.510e-01	3.100e-01	2.020e-01	2.240e-01	2.390e-01	2.270e-01	2.600e-01	2.170e-01
F06	1.000e-01	8.500e-02	2.500e-02	1.000e-01	5.500e-02	1.000e-01	1.450e-01	2.750e-02	8.500e-02	1.320e-01	8.500e-02	8.500e-02	1.000e-01
F07	2.100e-01	2.300e-01	1.980e-01	2.080e-01	1.760e-01	3.260e-01	2.450e-01	2.360e-01	2.490e-01	2.030e-01	2.140e-01	2.330e-01	2.590e-01
F08	4.210e-01	3.750e-01	4.050e-01	4.280e-01	4.620e-01	3.700e-01	4.760e-01	4.530e-01	3.820e-01	3.900e-01	4.410e-01	3.780e-01	4.050e-01
F09	2.880e-01	2.600e-01	2.380e-01	1.660e-01	2.970e-01	1.940e-01	3.250e-01	2.420e-01	2.030e-01	2.960e-01	1.890e-01	2.210e-01	2.020e-01
F10	3.440e-01	3.550e-01	3.200e-01	2.820e-01	3.290e-01	2.790e-01	3.040e-01	3.080e-01	3.030e-01	3.410e-01	2.890e-01	2.990e-01	3.090e-01
F11	2.720e-01	2.850e-01	2.440e-01	2.420e-01	2.640e-01	2.340e-01	2.810e-01	2.390e-01	2.990e-01	2.290e-01	2.060e-01	2.110e-01	2.480e-01
F12	1.770e-01	1.220e-01	1.150e-01	1.490e-01	1.630e-01	1.630e-01	1.300e-01	9.700e-02	1.180e-01	1.050e-01	1.120e-01	1.160e-01	
F13	2.660e-01	2.070e-01	1.850e-01	2.410e-01	1.880e-01	1.790e-01	3.330e-01	1.610e-01	2.430e-01	1.050e-01	2.070e-01	1.040e-01	
F14	5.070e-01	5.110e-01	5.040e-01	5.240e-01	5.460e-01	5.040e-01	5.130e-01	5.130e-01	5.600e-01	4.920e-01	4.560e-01	5.260e-01	
F15	3.940e-01	3.900e-01	4.210e-01	3.570e-01	4.200e-01	3.960e-01	4.830e-01	4.810e-01	4.520e-01	3.790e-01	4.310e-01	3.600e-01	2.130e-01
Best	0	1	1	2	1	2	0	1	1	0	3	1	2

Tabla A.5: Media de *fitness* de los algoritmos - knn - binario

	alco	aco	ba	cs	da	de	dummy	fa	ga	goa	gwo	pso	woa
F01	3.080e-01	1.940e-01	3.460e-01	3.620e-01	2.170e-01	3.470e-01	2.920e-01	2.580e-01	3.020e-01	2.980e-01	3.110e-01	2.820e-01	2.640e-01
F02	1.760e-01	2.040e-01	1.620e-01	9.010e-02	1.840e-01	1.910e-01	2.700e-01	1.210e-01	1.450e-01	2.190e-01	1.730e-01	1.300e-01	1.610e-01
F03	3.130e-01	2.700e-01	2.760e-01	2.670e-01	2.720e-01	3.480e-01	3.460e-01	2.880e-01	2.590e-01	3.260e-01	2.610e-01	2.600e-01	2.270e-01
F04	2.950e-01	3.530e-01	2.680e-01	2.310e-01	3.010e-01	2.820e-01	3.490e-01	3.230e-01	3.450e-01	3.010e-01	2.750e-01	2.920e-01	2.610e-01
F05	2.120e-01	1.710e-01	1.640e-01	1.340e-01	1.870e-01	1.400e-01	1.740e-01	1.570e-01	1.360e-01	1.610e-01	1.460e-01	1.380e-01	1.680e-01
F06	1.000e-01	1.150e-01	1.450e-01	1.150e-01	1.450e-01	7.000e-02	2.400e-01	1.200e-01	1.300e-01	3.020e-01	1.150e-01	1.300e-01	1.150e-01
F07	2.870e-01	2.070e-01	2.240e-01	2.010e-01	2.360e-01	2.480e-01	2.290e-01	3.100e-01	2.210e-01	2.260e-01	2.440e-01	2.540e-01	2.660e-01
F08	4.530e-01	3.330e-01	4.470e-01	4.260e-01	4.440e-01	3.620e-01	3.880e-01	3.860e-01	4.260e-01	3.990e-01	3.340e-01	3.890e-01	3.680e-01
F09	3.190e-01	1.740e-01	2.580e-01	2.080e-01	2.740e-01	2.670e-01	3.820e-01	2.140e-01	1.810e-01	3.690e-01	1.720e-01	2.400e-01	1.910e-01
F10	2.800e-01	2.270e-01	3.130e-01	2.910e-01	2.900e-01	2.740e-01	3.930e-01	2.840e-01	2.850e-01	2.410e-01	2.770e-01	2.780e-01	3.540e-01
F11	2.370e-01	2.480e-01	2.040e-01	1.920e-01	2.020e-01	1.920e-01	2.350e-01	2.040e-01	1.800e-01	2.000e-01	1.920e-01	1.870e-01	1.840e-01
F12	1.600e-01	8.780e-02	1.350e-01	1.440e-01	1.320e-01	1.310e-01	1.520e-01	1.310e-01	1.110e-01	8.800e-02	5.510e-02	1.150e-01	1.110e-01
F13	3.230e-01	2.200e-01	2.700e-01	2.960e-01	3.770e-01	2.050e-01	3.380e-01	4.100e-01	3.040e-01	4.060e-01	2.640e-01	2.140e-01	3.330e-01
F14	5.250e-01	4.910e-01	5.080e-01	5.000e-01	5.110e-01	5.210e-01	5.750e-01	4.980e-01	4.990e-01	5.410e-01	4.800e-01	4.590e-01	5.220e-01
F15	3.210e-01	5.690e-01	3.840e-01	3.010e-01	3.360e-01	3.240e-01	4.860e-01	2.460e-01	3.070e-01	4.340e-01	3.040e-01	3.240e-01	3.550e-01
Best	0	3	0	4	0	2	0	1	0	2	1	1	1

Tabla A.6: Media de *fitness* de los algoritmos - svc - binario

	abco	aco	ba	cs	da	de	dummy	fa	ga	goa	gwo	pso	woa
abco	- (0.015)	- (0.015)	+ (0.402)	= (0.784)	- (0.346)	= (0.529)	- (0.008)	= (0.679)	= (0.649)	- (0.379)	+ (0.414)	+ (0.196)	+ (0.410)
aco	+ (0.015)	-	+ (0.003)	+ (0.038)	+ (0.045)	+ (0.451)	+ (0.026)	+ (0.081)	+ (0.021)	+ (0.012)	+ (0.014)	+ (0.004)	
ba	- (0.402)	- (0.003)	-	- (0.307)	- (0.069)	= (0.875)	- (0.010)	= (0.802)	- (0.061)	- (0.038)	= (0.802)	= (0.616)	= (0.950)
cs	= (0.784)	- (0.038)	+ (0.307)	-	- (0.451)	= (0.944)	- (0.045)	= (0.934)	= (0.551)	= (0.762)	= (0.762)	= (0.720)	= (0.576)
da	+ (0.346)	- (0.045)	+ (0.069)	+ (0.451)	-	+ (0.346)	- (0.233)	+ (0.191)	= (0.934)	= (0.934)	+ (0.121)	+ (0.132)	+ (0.117)
de	= (0.529)	- (0.038)	= (0.875)	= (0.944)	- (0.346)	-	- (0.059)	= (0.762)	- (0.346)	- (0.421)	+ (0.389)	= (0.720)	+ (0.315)
dummy	+ (0.008)	- (0.451)	+ (0.010)	+ (0.045)	+ (0.233)	+ (0.059)	-	+ (0.060)	+ (0.252)	+ (0.155)	+ (0.028)	+ (0.008)	+ (0.048)
fa	= (0.679)	- (0.026)	= (0.802)	= (0.934)	- (0.191)	= (0.762)	- (0.060)	-	- (0.167)	- (0.330)	= (0.802)	= (0.950)	= (0.932)
ga	= (0.649)	- (0.081)	+ (0.061)	= (0.551)	= (0.934)	+ (0.346)	- (0.252)	+ (0.167)	-	= (0.906)	+ (0.906)	+ (0.162)	+ (0.263)
goa	+ (0.379)	- (0.021)	+ (0.038)	= (0.762)	= (0.934)	+ (0.421)	- (0.155)	+ (0.330)	= (0.906)	-	+ (0.233)	+ (0.196)	+ (0.107)
gwo	- (0.414)	- (0.012)	= (0.802)	= (0.762)	- (0.121)	- (0.389)	- (0.028)	= (0.802)	- (0.081)	- (0.233)	-	= (0.572)	= (0.802)
pso	- (0.196)	- (0.014)	= (0.616)	= (0.720)	- (0.132)	= (0.720)	- (0.008)	= (0.950)	- (0.162)	- (0.196)	= (0.572)	-	= (0.934)
woa	- (0.410)	- (0.004)	= (0.950)	= (0.576)	- (0.117)	- (0.315)	- (0.048)	= (0.932)	- (0.263)	- (0.107)	= (0.802)	= (0.934)	-

Tabla A.7: P-valores en accuracy - knn - binario

	abco	aco	ba	cs	da	de	dummy	fā	ga	goa	gwo	pso	woa	
abco	-	- (0.209)	= (0.660)	+ (0.463)	- (0.083)	= (0.890)	- (0.064)	= (0.720)	- (0.156)	- (0.095)	= (0.890)	= (0.934)	= (1.000)	
aco	+	(0.209)	-	+ (0.330)	+ (0.167)	= (0.530)	+ (0.116)	= (0.934)	+ (0.117)	+ (0.359)	= (0.720)	+ (0.079)	+ (0.188)	+ (0.124)
ba	=	(0.660)	- (0.330)	-	+ (0.094)	- (0.162)	+ (0.490)	- (0.151)	+ (0.389)	= (0.689)	- (0.167)	+ (0.421)	+ (0.286)	= (0.551)
cs	- (0.463)	- (0.167)	- (0.094)	-	- (0.026)	= (0.706)	- (0.064)	= (0.826)	- (0.055)	- (0.095)	= (0.706)	= (0.510)	- (0.490)	
da	+	(0.083)	= (0.530)	+ (0.162)	+ (0.026)	-	+ (0.196)	- (0.330)	+ (0.121)	+ (0.327)	- (0.359)	+ (0.149)	+ (0.132)	+ (0.208)
de	=	(0.890)	- (0.116)	- (0.490)	= (0.706)	- (0.196)	- (0.060)	= (0.660)	- (0.402)	- (0.229)	-	= (0.720)	= (0.847)	
dummy	+	(0.064)	= (0.934)	+ (0.151)	+ (0.064)	+ (0.330)	+ (0.060)	-	+ (0.142)	+ (0.303)	= (0.890)	+ (0.055)	+ (0.048)	+ (0.055)
fā	=	(0.720)	- (0.117)	- (0.389)	= (0.826)	- (0.121)	= (0.660)	- (0.142)	-	- (0.182)	- (0.209)	= (0.784)	- (0.470)	= (0.950)
ga	+	(0.156)	- (0.359)	= (0.689)	+ (0.055)	- (0.327)	+ (0.402)	- (0.303)	+ (0.182)	-	- (0.454)	+ (0.233)	+ (0.290)	= (0.514)
goa	+	(0.095)	= (0.720)	+ (0.167)	+ (0.095)	+ (0.359)	+ (0.229)	= (0.890)	+ (0.209)	+ (0.454)	-	+ (0.117)	+ (0.135)	+ (0.083)
gwo	=	(0.890)	- (0.079)	- (0.421)	= (0.706)	- (0.149)	= (1.000)	- (0.055)	= (0.784)	- (0.233)	- (0.117)	-	= (0.660)	= (0.727)
pso	=	(0.934)	- (0.188)	- (0.286)	= (0.510)	- (0.132)	= (0.720)	- (0.048)	+ (0.470)	- (0.290)	- (0.135)	= (0.660)	-	= (0.802)
woa	-	- (0.124)	= (0.551)	+ (0.490)	- (0.208)	= (0.847)	- (0.055)	= (0.950)	= (0.514)	- (0.083)	= (0.727)	= (0.802)	-	

Tabla A.8: P-valores en *accuracy* - SVC - binario

	abco	aco	ba	cs	da	de	dummy	fa	ga	goa	gwo	pso	woa
F01	5.5	1	4	7	11	9	12	13	2.5	8	2.5	10	5.5
F02	6.5	13	6.5	9.5	9.5	4.5	11	2.5	4.5	12	1	2.5	8
F03	7	12	9	7	2	7	10	1	13	4	3	5	11
F04	9	12	13	1	4.5	7.5	7.5	6	10.5	10.5	3	2	4.5
F05	9	11.5	3	13	9	6	11.5	1	4.5	4.5	7	9	2
F06	10.5	6	1.5	10.5	3	10.5	13	1.5	6	6	6	6	10.5
F07	2	11.5	3	5.5	1	13	5.5	7	11.5	4	8	9	10
F08	2.5	8.5	4.5	8.5	13	1	10.5	10.5	6.5	4.5	12	2.5	6.5
F09	9	13	6.5	1	10.5	2	10.5	8	5	12	4	6.5	3
F10	6.5	13	6.5	3	11	1.5	1.5	4	10	12	6.5	9	6.5
F11	8	13	7	10	12	5	11	3.5	1.5	6	3.5	1.5	9
F12	10	12.5	1	3	11	12.5	8.5	5.5	3	5.5	8.5	7	3
F13	10.5	8	5	10.5	5	5	13	3	12	8	2	8	1
F14	3	11	4	10	13	6	2	7	9	12	5	1	8
F15	4	8	8	2	8	6	10.5	12.5	4	10.5	4	1	
Mean	6.867	10.267	5.500	6.767	8.233	6.433	9.200	5.733	7.467	7.533	5.500	5.533	5.967

Tabla A.9: Ranking de los algoritmos en accuracy - knn - binario

	abco	aco	ba	cs	da	de	dummy	fa	ga	goa	gwo	pso	woa
F01	6	1.5	11.5	13	1.5	11.5	3.5	3.5	9	7	10	8	5
F02	4.5	13	4.5	1	8	9	12	2	7	11	10	3	6
F03	9	10	4.5	2	6	13	12	7.5	4.5	11	7.5	3	1
F04	4	13	2.5	1	9	5	11	10	12	6	7	8	2.5
F05	11	13	6	1.5	12	1.5	3	4	6	8.5	8.5	6	10
F06	2	5	10.5	5	10.5	1	12	5	8.5	13	5	8.5	5
F07	8.5	8.5	3.5	2	6	6	1	13	6	3.5	11	11	11
F08	9	6	11	10	12.5	2.5	2.5	5	12.5	8	1	7	4
F09	11	6	7	5	10	9	12	4	3	13	1	8	2
F10	1	6	10.5	7.5	9	4	13	4	10.5	2	4	7.5	12
F11	8	13	4.5	2	9	3	11	6.5	4.5	10	12	6.5	1
F12	4	7.5	9.5	13	9.5	7.5	5.5	5.5	11.5	2	1	11.5	3
F13	6.5	3	4	6.5	11	1	8	12.5	9	12.5	5	2	10
F14	7	12	4	5.5	8.5	8.5	13	3	5.5	11	2	1	10
F15	2.5	13	10	2.5	5.5	5.5	12	1	5.5	11	5.5	8	9
Mean	6.267	8.700	6.900	5.167	8.533	5.867	8.767	5.767	7.667	8.633	6.033	6.600	6.100

Tabla A.10: Ranking de los algoritmos en *accuracy* - svc - binario

	abco	aco	ba	cs	da	de	dummy	fa	ga	goa	gwo	pso	woa
abco	-	+ (0.001)	+ (0.001)	+ (0.001)	+ (0.001)	= (0.820)	+ (1.831E-04)	+ (0.001)	+ (0.015)	+ (0.001)	+ (0.001)	+ (0.001)	+ (0.001)
aco	- (0.001)	-	- (0.001)	- (0.001)	- (0.001)	- (0.001)	- (6.104E-05)	- (6.104E-05)	- (0.001)	- (6.104E-05)	- (0.002)	- (0.001)	- (0.001)
ba	- (0.001)	+ (0.001)	-	+ (0.001)	+ (0.014)	+ (0.010)	- (6.104E-05)	= (0.570)	+ (0.001)	= (0.934)	+ (0.001)	+ (0.001)	+ (0.002)
cs	- (0.001)	+ (0.001)	+ (0.001)	- (0.001)	- (0.149)	- (0.001)	- (6.104E-05)	- (0.001)	+ (0.001)	- (0.029)	+ (0.001)	+ (0.001)	+ (0.002)
da	- (0.001)	+ (0.001)	- (0.001)	+ (0.149)	+ (0.149)	= (0.778)	- (0.001)	- (0.005)	+ (0.001)	= (0.524)	+ (0.001)	+ (0.002)	+ (0.485)
de	- (0.001)	+ (0.001)	- (0.010)	+ (0.001)	+ (0.778)	- (0.778)	- (6.104E-05)	- (6.104E-05)	- (1.831E-04)	+ (0.001)	- (0.359)	+ (0.001)	+ (0.117)
dummy	= (0.829)	+ (6.104E-05)	+ (6.104E-05)	+ (6.104E-05)	+ (0.001)	+ (6.104E-05)	-	-	+ (0.001)	+ (6.104E-05)	+ (6.104E-05)	+ (6.104E-05)	+ (6.104E-05)
fa	- (1.831E-04)	+ (6.104E-05)	+ (6.104E-05)	= (0.570)	+ (0.001)	+ (1.831E-04)	- (0.001)	- (0.001)	- (0.001)	+ (6.104E-05)	+ (0.258)	+ (6.104E-05)	+ (0.002)
ga	- (0.001)	+ (0.001)	- (0.001)	- (0.001)	- (0.001)	- (0.001)	- (6.104E-05)	- (6.104E-05)	- (6.104E-05)	- (6.104E-05)	+ (0.031)	- (0.013)	- (0.001)
goa	- (0.015)	+ (6.104E-05)	= (0.934)	+ (0.934)	+ (0.329)	= (0.524)	+ (0.359)	- (0.001)	- (0.258)	+ (6.104E-05)	-	+ (6.104E-05)	+ (0.201)
gwo	- (0.001)	+ (0.002)	- (0.001)	- (0.001)	- (0.001)	- (0.001)	- (6.104E-05)	- (6.104E-05)	- (0.031)	- (6.104E-05)	-	- (0.002)	- (0.001)
pso	- (0.001)	+ (0.001)	- (0.001)	- (0.002)	- (0.001)	- (0.001)	- (6.104E-05)	- (6.104E-05)	+ (0.013)	+ (0.001)	+ (0.002)	-	- (0.001)
woa	- (0.001)	+ (0.001)	- (0.002)	+ (0.208)	+ (0.208)	- (0.485)	- (0.117)	- (0.002)	+ (0.001)	- (0.201)	+ (0.001)	+ (0.001)	-

Tabla A.11: P-valores en ratio de selección - knn - binario

	abco	abco	aco	ba	cs	da	de	dummy	fa	ga	goa	gwo	pso	woa
abco	- (0.001)	-	+ (0.001)	+ (0.001)	+ (0.001)	+ (0.001)	= (0.847)	+ (1.831E-04)	+ (0.001)	+ (0.010)	+ (0.001)	+ (0.001)	+ (0.001)	
aco	- (0.001)	-	- (0.001)	- (0.001)	- (0.001)	- (0.001)	- (6.104E-05)	- (6.104E-05)	- (0.001)	- (0.001)	- (0.001)	- (0.001)	- (0.001)	
ba	- (0.001)	+ (0.001)	-	- (0.001)	+ (0.001)	+ (0.003)	+ (0.017)	(0.489)	+ (0.001)	= (0.804)	+ (0.001)	+ (0.001)	+ (0.001)	
cs	- (0.001)	+ (0.001)	-	- (0.001)	-	= (0.851)	- (0.004)	- (6.104E-05)	- (6.104E-05)	+ (0.001)	- (0.033)	+ (0.001)	+ (0.001)	- (0.279)
da	- (0.001)	+ (0.001)	-	- (0.003)	-	= (0.851)	- (0.009)	- (6.104E-05)	- (6.104E-05)	+ (0.002)	- (0.244)	+ (0.004)	+ (0.001)	= (0.706)
de	- (0.001)	+ (0.001)	-	- (0.017)	+ (0.004)	+ (0.090)	- (6.104E-05)	- (6.104E-05)	- (0.004)	+ (0.001)	+ (0.001)	+ (0.015)	+ (0.001)	
dummy	- (0.847)	+ (0.001)	+ (6.104E-05)	+ (0.001)	+ (6.104E-05)	+ (6.104E-05)	+ (0.001)							
fa	- (1.831E-04)	+ (6.104E-05)	+ (0.489)	+ (6.104E-05)	+ (6.104E-05)	+ (0.004)	- (6.104E-05)	- (6.104E-05)	- (0.890)	+ (6.104E-05)	+ (0.001)	+ (0.001)	+ (0.001)	
ga	- (0.001)	+ (0.001)	- (0.001)	- (0.002)	- (0.001)	- (0.001)	- (6.104E-05)	- (6.104E-05)	- (0.184)	- (0.002)	+ (0.001)	+ (0.001)	+ (0.001)	
goa	- (0.010)	+ (6.104E-05)	= (0.804)	+ (0.033)	+ (0.244)	+ (0.330)	+ (0.890)	+ (6.104E-05)	- (0.233)	- (3.052E-04)	- (0.002)	- (0.002)	- (0.002)	
gwo	- (0.001)	+ (0.001)	- (0.001)	- (0.004)	- (0.001)	- (0.001)	- (6.104E-05)	- (6.104E-05)	- (0.055)	- (0.184)	- (0.001)	- (0.001)	- (0.001)	
pso	- (0.001)	+ (0.001)	- (0.001)	- (0.001)	- (0.001)	- (0.001)	- (6.104E-05)	- (6.104E-05)	+ (0.184)	- (0.001)	+ (0.001)	+ (0.001)	+ (0.001)	
woa	- (0.001)	+ (0.001)	- (0.279)	+ (0.706)	- (0.015)	- (0.001)	- (0.001)	- (0.001)	- (0.0083)	+ (0.002)	+ (0.001)	+ (0.001)	+ (0.001)	-

Tabla A.12: P-valores en ratio de selección - svc - binario

	abco	aco	ba	cs	da	de	dummy	fa	ga	goa	gwo	pso	woa
F01	13	1	8	7	4.5	9	12	10	3	11	2	4.5	6
F02	12	1	11	6	9	8	13	10	3	7	2	4	5
F03	12	1	8	7	6	9	13	10	2.5	11	2.5	4	5
F04	13	1	10	6	5	7	12	8	3	11	3	3	9
F05	13	1	11	8	6	9	12	10	3	5	2	4	7
F06	5.5	5.5	5.5	5.5	5.5	5.5	12	11	5.5	13	5.5	5.5	5.5
F07	12	1	9	5	10	6	13	11	3	8	2	4	7
F08	13	1	11	8	6	9	12	10	2	5	3	4	7
F09	13	1	11	6	8	9	12	10	2	5	3	4	7
F10	13	1	11	5	7	9	12	10	3	6	2	4	8
F11	13	1	10	5	9	8	12	11	3	6	2	4	7
F12	13	1	11	6	8	9	12	10	3	5	2	4	7
F13	12	1.5	8	5	9	6	13	10.5	4	10.5	1.5	3	7
F14	11	1	8	7	9.5	6	12	5	3.5	13	2	3.5	9.5
F15	12	1	10	5.5	9	7	13	11	4	8	2	3	5.5
Mean	12.033	1.333	9.500	6.133	7.433	7.767	12.333	9.833	3.167	8.300	2.433	3.900	6.833

Tabla A.13: Ranking de los algoritmos en seleccion de caracteristicas - knn - binario

	abco	aco	ba	cs	da	de	dummy	fa	ga	goa	gwo	pso	woa
F01	12	1	7	6	5	8.5	13	10	2	11	3	4	8.5
F02	12	1	10	5	9	6.5	13	11	3	8	2	4	6.5
F03	12	1	8	7	6	9	13	10	3	11	2	4	5
F04	13	1	9	7	5	10	11	8	4	12	2.5	2.5	6
F05	13	1	11	7	8	9	12	10	3	5	2	4	6
F06	5.5	5.5	5.5	5.5	5.5	5.5	12	11	5.5	13	5.5	5.5	5.5
F07	13	1	8	5	6	9	12	11	2	10	3	4	7
F08	13	1	11	7	5	9	12	10	2	4	6	3	8
F09	13	1	11	6	9	8	12	10	2	5	3	4	7
F10	13	1	11	9	5	6	12	10	2	4	7.5	3	7.5
F11	13	1	11	9	6	8	12	10	3.5	5	2	3.5	7
F12	13	1	10	5	8	9	12	11	3	7	2	4	6
F13	12	2	9	6	7	5	13	11	3.5	10	1	3.5	8
F14	11	1	10	6	4	9	12	7.5	5	13	3	2	7.5
F15	12	1	9	5.5	10	7	13	11	4	8	2	3	5.5
Mean	12.033	1.367	9.367	6.400	6.567	7.900	12.267	10.100	3.167	8.400	3.100	3.600	6.733

Tabla A.14: Ranking de los algoritmos en seleccion de caracteristicas - svc - binario

optimizer	Data	dataset	breast-cancer	dermatology	diabetes	ecoli	ionosphere	iris	parkinsons	sonar	spambase-460	spectfheart	waveform-5000	wdbc	wine	yeast	200
abo	Average - acc	74.17%	88.00%	73.23%	77.14%	95.33%	91.67%	77.50%	60.00%	74.74%	79.29%	83.75%	91.74%	71.25%	51.33%	70.00%	
	Average - selected_rate	75.65%	92.50%	90.00%	95.00%	84.55%	92.67%	91.58%	93.64%	91.00%	85.33%	63.85%	87.50%	51.18%	51.18%	51.18%	
	Average - acc	79.17%	71.94%	65.71%	82.00%	90.00%	77.50%	63.33%	81.05%	75.00%	73.45%	90.87%	77.50%	50.00%	38.00%	38.00%	
	Average - selected_rate	6.67%	12.35%	17.50%	44.29%	8.53%	25.00%	4.09%	2.50%	3.33%	1.82%	9.25%	5.67%	17.69%	41.25%	10.59%	
ba	Average - acc	66.67%	88.00%	74.52%	77.86%	88.00%	86.67%	81.25%	57.78%	78.42%	72.86%	84.70%	90.43%	75.00%	52.67%	62.00%	
	Average - selected_rate	45.56%	53.82%	46.25%	68.57%	55.59%	25.00%	55.00%	66.83%	63.86%	68.86%	66.50%	44.62%	82.50%	41.76%	41.76%	
	Average - acc	64.17%	94.67%	75.16%	81.43%	90.00%	90.00%	82.50%	58.89%	82.63%	74.29%	85.35%	88.26%	71.25%	52.33%	70.00%	
	Average - selected_rate	40.00%	42.06%	43.75%	64.29%	44.12%	25.00%	43.64%	55.83%	52.11%	59.77%	59.75%	38.00%	37.69%	71.25%	31.18%	
da	Average - acc	79.17%	85.33%	74.19%	72.86%	84.67%	86.67%	78.75%	56.67%	76.32%	73.57%	83.70%	90.43%	62.50%	50.67%	68.00%	
	Average - selected_rate	30.00%	52.06%	40.00%	57.14%	48.53%	25.00%	45.00%	54.17%	60.53%	51.82%	55.00%	46.33%	39.23%	67.50%	47.65%	
	Average - acc	66.67%	84.00%	66.77%	76.43%	90.00%	95.00%	78.75%	66.67%	76.84%	75.71%	85.25%	90.87%	81.25%	50.67%	68.00%	
	Average - selected_rate	46.67%	47.06%	48.75%	70.00%	49.71%	25.00%	56.36%	62.50%	58.25%	55.68%	58.75%	48.67%	36.15%	77.50%	35.88%	
de	Average - acc	76.67%	79.33%	70.00%	69.29%	89.33%	80.00%	83.75%	66.67%	67.37%	65.71%	83.15%	91.30%	70.00%	46.17%	54.00%	
	Average - selected_rate	82.22%	83.82%	76.25%	72.86%	77.94%	60.00%	82.27%	88.33%	88.77%	84.55%	83.50%	73.67%	68.46%	90.00%	71.76%	
	Average - acc	76.67%	92.67%	73.55%	71.43%	88.67%	90.00%	72.50%	64.44%	83.16%	75.71%	84.00%	91.30%	60.00%	53.00%	78.00%	
	Average - selected_rate	47.78%	55.29%	50.00%	65.71%	55.29%	30.00%	62.73%	66.00%	62.28%	65.45%	60.25%	52.33%	50.00%	75.00%	48.24%	
ga	Average - acc	69.17%	86.67%	74.52%	67.86%	88.00%	88.33%	78.75%	56.67%	84.21%	72.86%	84.75%	90.00%	68.75%	52.33%	68.00%	
	Average - selected_rate	24.44%	25.29%	30.00%	55.71%	27.65%	25.00%	29.55%	35.83%	38.77%	40.23%	42.75%	21.00%	22.31%	19.41%	19.41%	
	Average - acc	73.33%	81.33%	70.97%	75.71%	86.67%	75.00%	81.25%	61.11%	64.74%	78.57%	83.60%	94.78%	60.00%	50.17%	56.00%	
	Average - selected_rate	57.78%	65.00%	62.86%	41.18%	77.50%	57.27%	49.33%	51.93%	48.64%	52.50%	41.00%	46.15%	92.50%	38.24%	38.24%	
gwo	Average - acc	68.33%	82.67%	73.55%	86.07%	90.00%	76.25%	68.89%	85.26%	75.71%	82.65%	95.65%	72.50%	53.83%	68.00%	68.00%	
	Average - selected_rate	25.56%	17.06%	22.50%	54.29%	26.18%	25.00%	50.45%	54.50%	39.65%	58.18%	36.00%	16.00%	16.92%	65.00%	16.47%	
	Average - acc	71.67%	88.67%	74.84%	73.57%	88.00%	88.33%	76.25%	62.22%	77.89%	74.29%	84.00%	90.00%	78.75%	55.50%	66.00%	
	Average - selected_rate	26.67%	28.24%	33.75%	54.29%	29.71%	25.00%	40.00%	48.67%	41.40%	46.36%	42.75%	24.67%	22.31%	58.75%	18.24%	
wca	Average - acc	75.83%	87.33%	79.03%	77.86%	86.00%	90.00%	76.25%	65.36%	84.74%	67.14%	85.90%	92.17%	60.33%	64.00%	64.00%	
	Average - selected_rate	46.67%	47.06%	38.75%	61.43%	42.35%	25.00%	52.73%	58.00%	53.68%	58.18%	57.00%	40.67%	40.00%	75.00%	31.18%	

Tabla A.15: Porcentajes de características seleccionadas y precisión en clasificación para cada algoritmo binario

	abco	aco	ba	cs	da	de	fa	ga	g oa	gwo	pso	woa
F01	9.740e+01	9.980e+00	1.930e+01	1.350e+01	1.020e+01	9.680e+00	2.130e+00	1.700e+01	1.030e+01	3.960e+01	9.340e+00	9.840e+00
F02	4.780e+01	6.540e+00	9.210e+00	7.360e+00	5.280e+00	5.630e+00	1.110e+00	8.650e+00	8.200e+00	2.040e+01	4.920e+00	6.330e+00
F03	1.540e+02	1.560e+01	3.130e+01	2.020e+01	1.500e+01	1.480e+01	3.880e+00	2.700e+01	1.510e+01	6.840e+01	1.470e+01	1.490e+01
F04	1.040e+02	1.030e+01	2.000e+01	1.420e+01	1.060e+01	1.010e+01	1.950e+00	1.750e+01	1.040e+01	4.640e+01	1.010e+01	1.020e+01
F05	5.010e+01	6.540e+00	9.790e+00	7.110e+00	5.620e+00	5.930e+00	1.420e+00	8.720e+00	7.750e+00	1.980e+01	4.930e+00	6.700e+00
F06	7.430e+01	7.540e+00	1.470e+01	1.070e+01	8.160e+00	7.690e+00	1.010e+00	1.330e+01	7.770e+00	2.960e+01	7.310e+00	7.550e+00
F07	4.710e+01	5.560e+00	9.200e+00	6.610e+00	5.130e+00	5.300e+00	1.170e+00	8.190e+00	6.370e+00	1.840e+01	4.570e+00	5.710e+00
F08	4.910e+01	9.990e+00	9.610e+00	7.400e+00	5.400e+00	6.310e+00	1.140e+00	8.570e+00	8.880e+00	1.930e+01	4.760e+00	7.790e+00
F09	5.110e+01	9.310e+00	1.010e+01	7.550e+00	5.630e+00	6.230e+00	1.320e+00	9.050e+00	8.670e+00	2.050e+01	5.000e+00	7.460e+00
F10	4.770e+01	7.480e+00	9.470e+00	7.050e+00	5.590e+00	6.050e+00	1.190e+00	8.390e+00	8.220e+00	2.160e+01	4.920e+00	7.080e+00
F11	4.430e+02	4.390e+01	8.620e+01	6.040e+01	4.410e+01	4.390e+01	8.400e+00	7.700e+01	4.660e+01	1.950e+02	4.270e+01	4.420e+01
F12	5.050e+01	6.550e+00	1.010e+01	7.980e+00	6.000e+00	6.190e+00	1.380e+00	9.140e+00	7.810e+00	1.950e+01	5.350e+00	6.850e+00
F13	8.050e+01	8.400e+00	1.580e+01	1.140e+01	8.560e+00	8.420e+00	1.490e+00	1.430e+01	8.810e+00	3.200e+01	7.990e+00	8.440e+00
F14	2.590e+02	3.120e+01	5.290e+01	3.830e+01	2.750e+01	2.820e+01	4.730e+00	4.860e+01	2.780e+01	1.030e+02	2.670e+01	2.940e+01
F15	4.620e+01	5.140e+00	8.830e+00	6.660e+00	5.110e+00	5.070e+00	1.080e+00	7.990e+00	5.840e+00	1.790e+01	4.400e+00	5.340e+00
Best	0	0	0	0	0	0	0	0	0	0	0	0

Tabla A.16: Media de tiempos - knn - binario

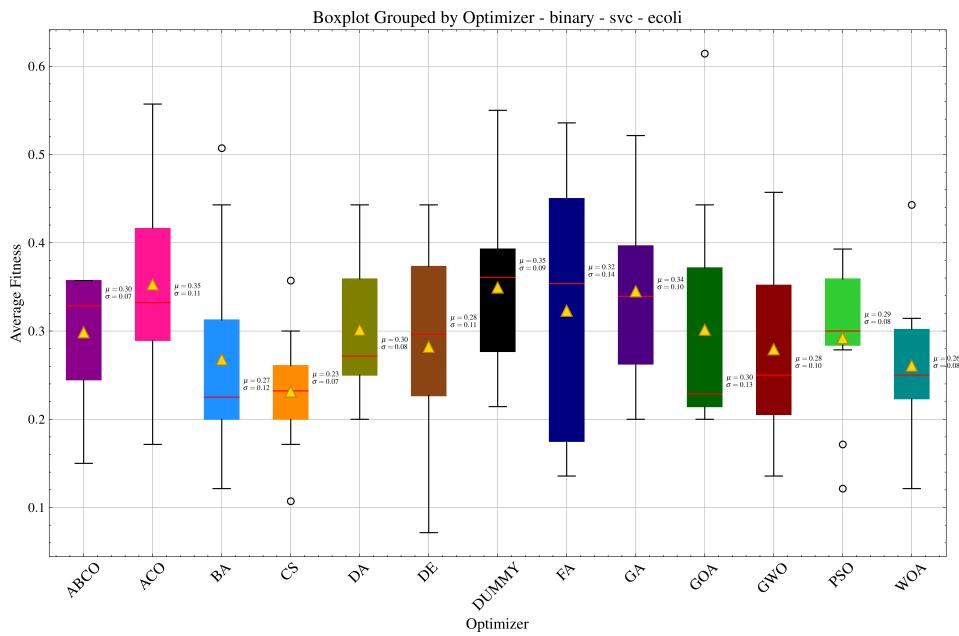


Figura A.56: *Boxplot ecoli - svc - binary*

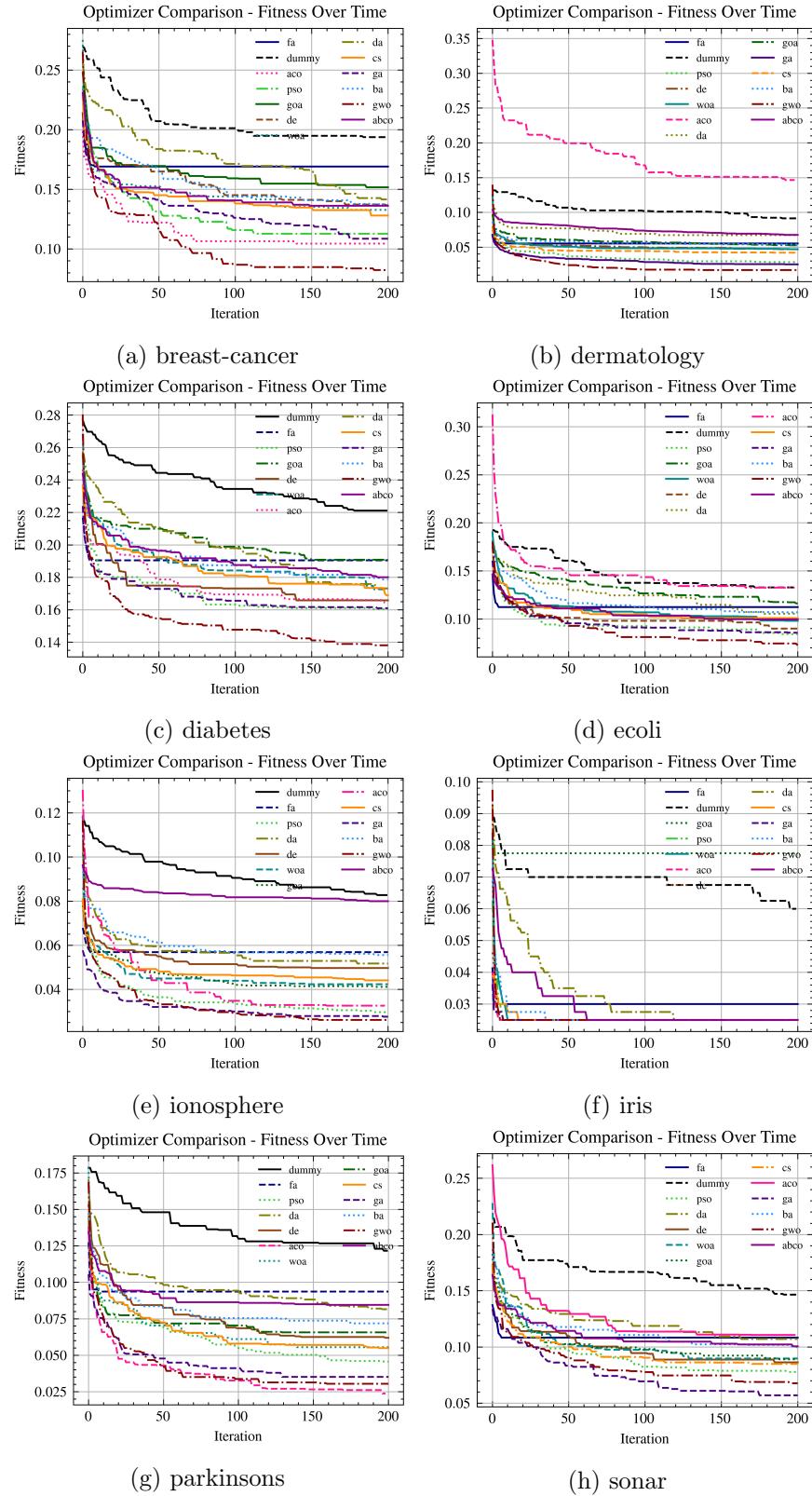


Figura A.57: Convergencia con svc parte 1 - binario

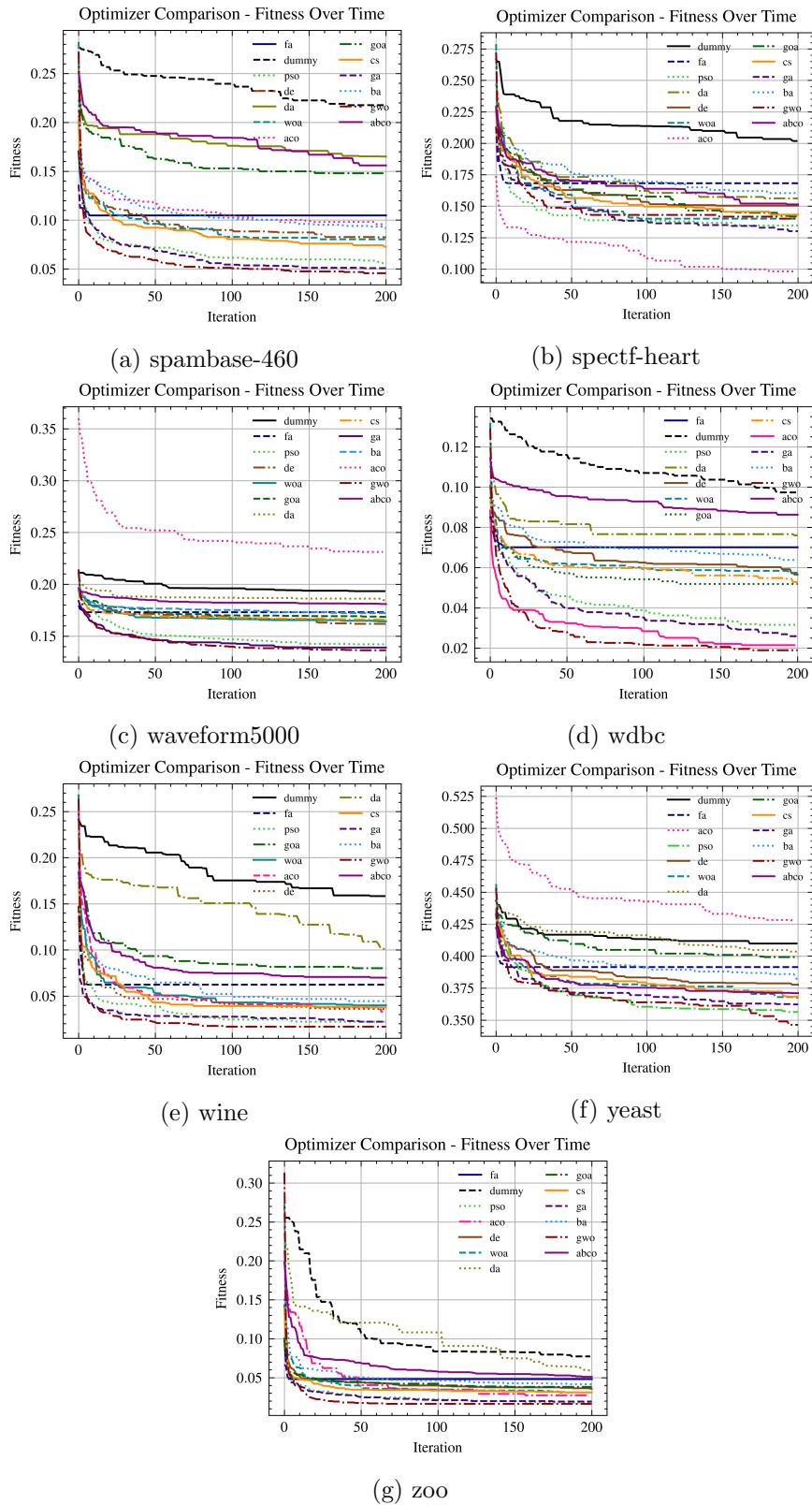


Figura A.58: Convergencia con svc parte 2 - binario

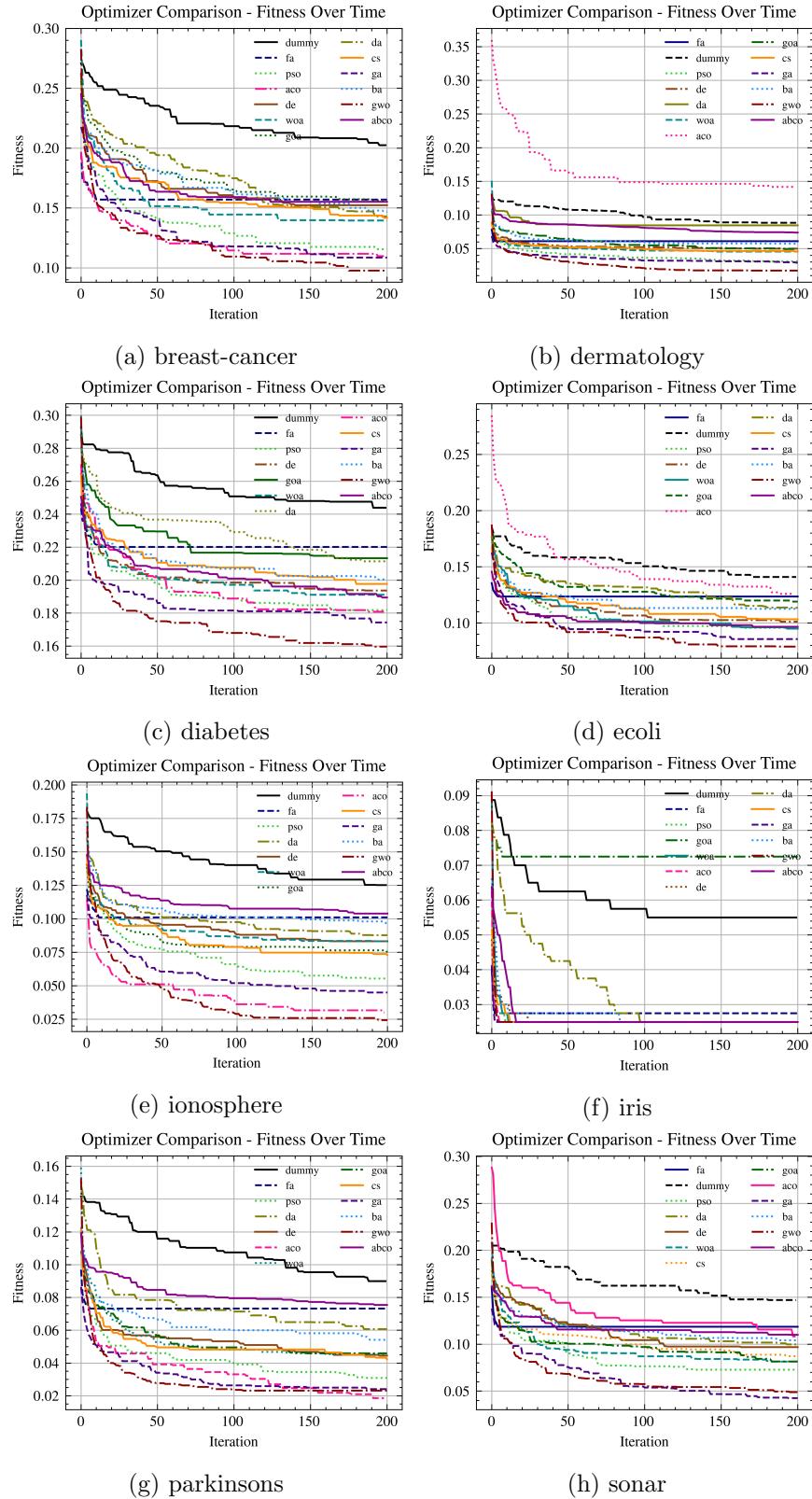


Figura A.59: Convergencia con knn parte 1 - binario

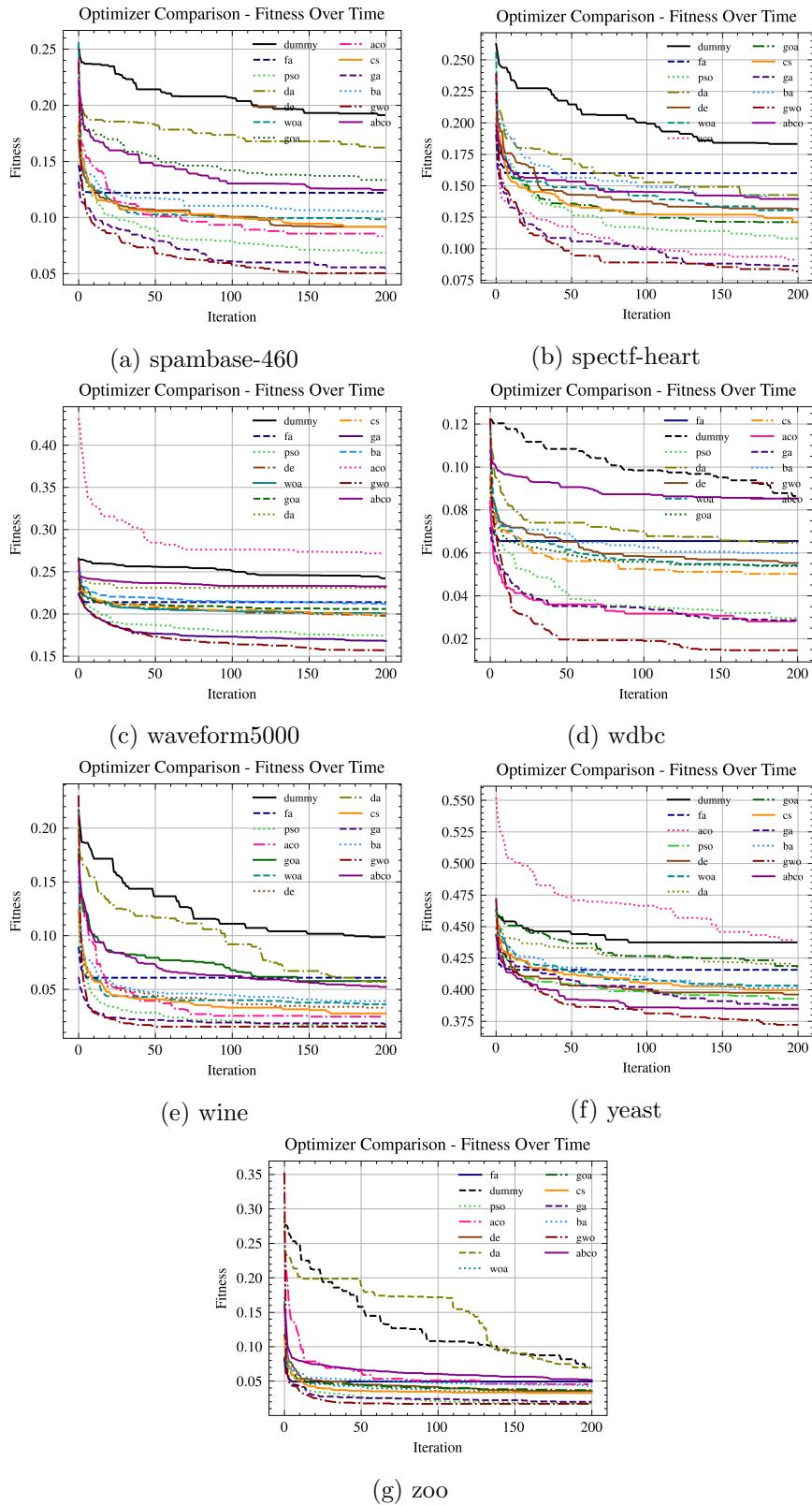


Figura A.60: Convergencia con knn parte 2 - binario