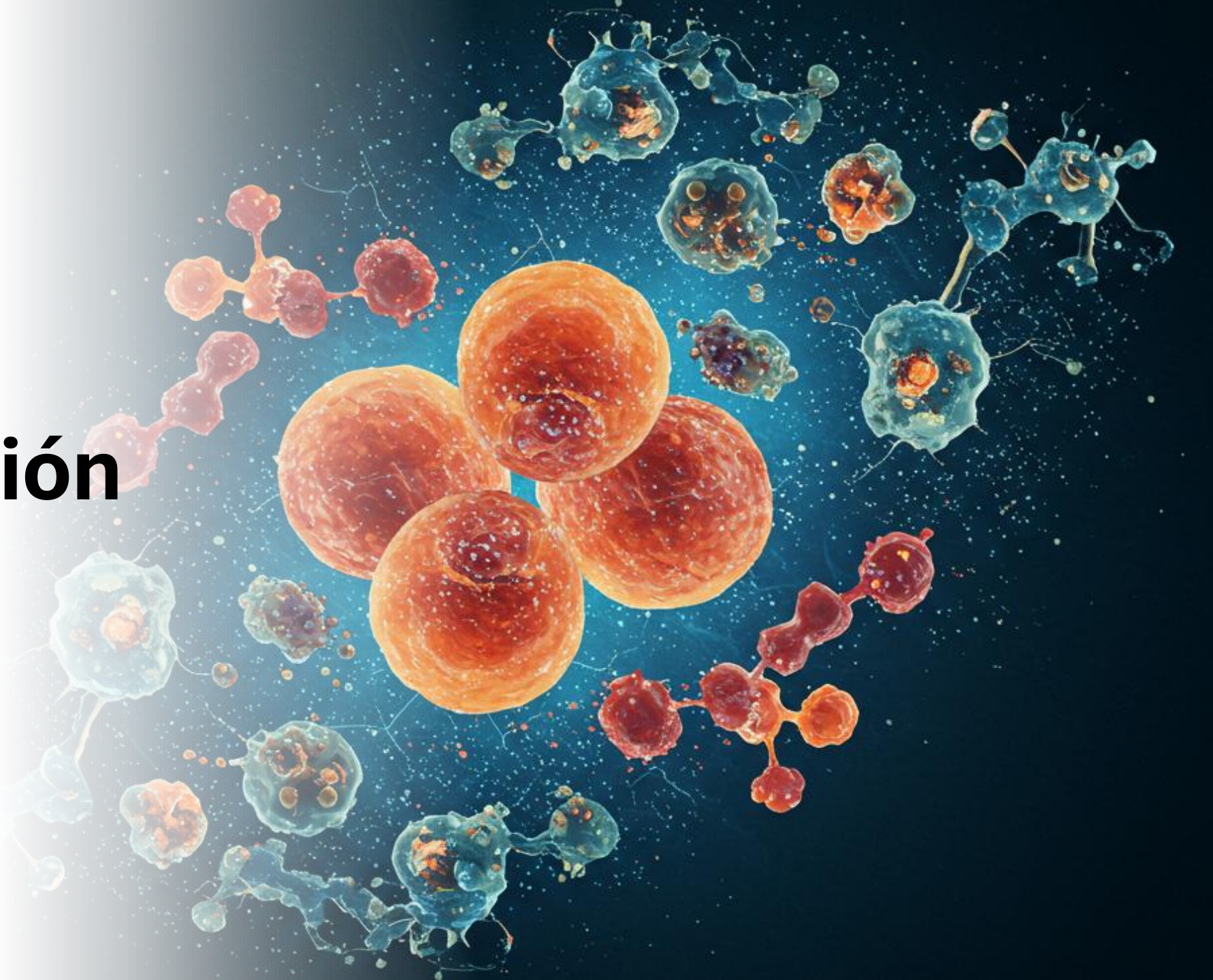
Programación evolutiva

Problemas multiobjetivo

Programación evolutiva



Algoritmos genéticos frente a programación evolutiva

Mientras que los Algoritmos Genéticos (**AG**) buscan optimizar principalmente el genotipo mediante la recombinación de "recetas" codificadas, la Programación Evolutiva (**PE**) se enfoca en la optimización directa del fenotipo de soluciones, confiando primordialmente en la mutación para explorar el espacio de soluciones y adaptarlas.

Característica	Algoritmos Genéticos (GA)	Programación Evolutiva (PE)
Inspiración o énfasis	- Evolución a nivel genético (cromosomas y genes). El Interés primordial es en el genotipo (la codificación interna de la solución).	- Evolución a nivel de especies. Énfasis en la adaptación fenotípica (las características observables de la solución, la solución en sí).
Representación	- Tradicionalmente cadenas binarias, buscando la combinación de "bloques constructivos". Puede usar representaciones indirectas o codificaciones de valor real.	- Comúnmente vectores de números reales (para optimización de parámetros) o máquinas de estados finitos (para comportamientos/estrategias). La representación tiende a ser más directa (fenotípica), manipulando la solución candidata directamente.
Operador Principal	- Cruce (recombinación): Es el corazón del AG, combinando información genética de dos padres para crear descendencia. Se considera el motor principal de la variación.	- Mutación: Es el operador principal, y a menudo el único, para generar nuevas soluciones. La variación se produce por cambios aleatorios en las características de la solución existente.
Rol del Cruce	- Fundamental: para la exploración eficiente del espacio de búsqueda y la explotación de buenos "genes" (características).	- Relegado o ausente: Tradicionalmente no se utiliza el cruce; la innovación y la exploración recaen completamente en la mutación.
Estrategia de Selección	- Se enfoca en seleccionar a los padres más aptos para la reproducción (ej., selección por ruleta, torneo).	- A menudo se centra en la selección de supervivencia o competición directa entre los hijos generados y, a veces, los padres, para decidir qué individuos pasarán a la siguiente generación (los mejores sobreviven).
Aplicaciones Típicas	- Problemas de optimización combinatoria, problemas donde la recombinación de subcomponentes es beneficiosa (ejemplo, programación de horarios, toma de decisiones, planificación de tareas).	- Optimización de parámetros en sistemas complejos (continuos). Evolución de comportamientos, estrategias o controladores (ejemplo: toma de decisión en agentes autónomos).
Énfasis Teórico	- A menudo se basa en la hipótesis de los bloques constructivos, las buenas soluciones se construyen a partir de la recombinación de partes subóptimas.	- Su fundamento es la adaptación fenotípica, donde la variación es producida directamente en las características de la solución y la selección favorece los cambios que mejoran el rendimiento.

Algoritmos genéticos frente a programación evolutiva

Programación evolutiva

1. **Inicialización**: se genera una población inicial de *N* individuos usualmente de forma aleatoria. Está definida una función de aptitud u objetivo: $f(x_i)$.

2. **Evaluación:** se aplica la función de aptitud $f(x_i)$ que asigna un valor de aptitud a cada individuo x_i. Esto permite medir la calidad de la solución.

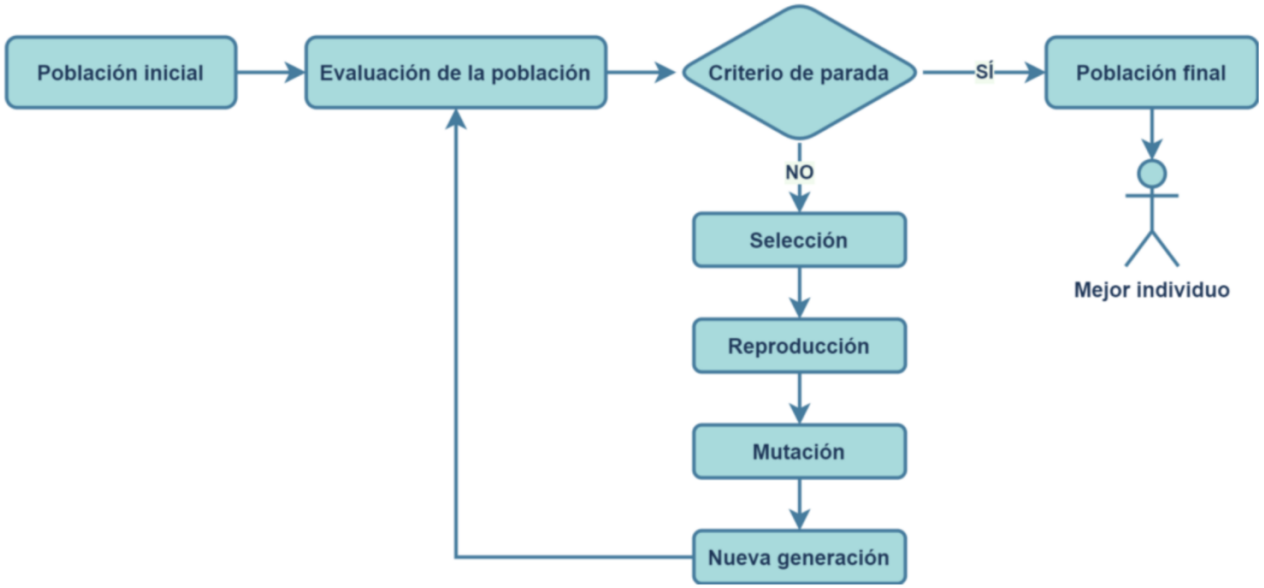
3. **Selección:** se seleccionan los individuos más aptos de la población según un criterio de selección (mediante torneo o proporcional a la aptitud). Los individuos más aptos deben tener mayor probabilidad de ser seleccionados.

4. **Mutación:** se aplican operadores de mutación a los individuos seleccionados para generar descendientes. La mutación introduce variaciones aleatorias en las soluciones, lo cual permite la exploración del espacio de búsqueda.

5. **Reemplazo:** los descendientes reemplazan a los individuos menos aptos de la población, manteniendo el tamaño de la población constante.

6. **Iteración:** Se repiten los pasos 2 al 5 hasta que se alcanza un criterio de terminación, como: un número máximo de generaciones, un nivel de aptitud satisfactorio, o el esfuerzo computacional establecido como límite.

Algoritmo Genético



Programación evolutiva, aprendizaje basado en reglas

El aprendizaje basado en reglas (**RBML**) es un enfoque de aprendizaje automático que utiliza un conjunto de reglas para representar el conocimiento aprendido. Cada regla consta de una condición (**antecedente**) y una acción (**consecuente**) de la forma "**SI**... **ENTONCES**...". Cuando la condición de una regla se cumple, la acción asociada se ejecuta. Una regla se puede representar como:

SI c*ondición* ENTONCES c*onclusión*

Buscar las reglas cuya condición se cumple para los datos de entrada y aplicar su conclusión. Si varias reglas se cumplen, se utilizan mecanismos de resolución de conflictos para determinar la salida final.

Programación evolutiva, aprendizaje basado en reglas

- Sistemas Clasificadores de Aprendizaje (LCS)

  Son una clase de algoritmos de aprendizaje basado en reglas que desarrollan y emplean un conjunto de reglas SI-ENTONCES para comprender y interactuar con entornos complejos y adaptativos. Su propósito principal es el modelado, la clasificación, la predicción y la toma de decisiones secuencial, donde las reglas establecen una relación causal o clasificatoria entre múltiples variables independientes observadas ({V1, V2, V3}) y una variable dependiente, que puede ser una clase específica o una acción a ejecutar (como en {V1, V2, V3} → Clase/Acción).

  Se utilizan eficazmente en sistemas donde el comportamiento óptimo debe aprenderse a medida que el sistema interactúa con su entorno, como en control de robótica, juegos, optimización de recursos o en cualquier escenario que demande un aprendizaje continuo y adaptación frente a la incertidumbre.

Programación evolutiva, aprendizaje basado en reglas

- Minería de Reglas de Asociación (ARM)

  La Minería de Reglas de Asociación (ARM) es un enfoque de RBL diseñado principalmente para descubrir patrones "Si-Entonces" y relaciones interesantes entre variables dentro de grandes conjuntos de datos. A diferencia de los sistemas clasificadores, las reglas de ARM no predicen una clase o acción, sino que vinculan la ocurrencia de un conjunto de elementos con la ocurrencia de otro conjunto de elementos o una variable independiente diferente (por ejemplo, {V1, V2, V3} → V4).

  Su utilidad primordial radica en identificar comportamientos frecuentes o coexistencias ocultas, como "si un cliente compra leche y pan, entonces es probable que compre mantequilla". Se aplica ampliamente en el sector minorista para el análisis de cestas de compra y recomendación de productos, en la medicina para descubrir asociaciones entre síntomas y enfermedades, o en la detección de fraudes para identificar patrones de transacción inusuales.

Programación evolutiva, aprendizaje basado en reglas

- Sistemas Inmunológicos Artificiales (AIS)

  Los Sistemas Inmunológicos Artificiales (AIS) son una rama del aprendizaje basado en reglas que se inspira en el funcionamiento y los principios del sistema inmune biológico para resolver problemas computacionales.

  Su aplicación fundamental es la detección de anomalías o intrusiones, distinguiendo de manera efectiva entre lo que se considera "propio" (datos normales y esperados) y lo "no propio" (patrones anómalos, desviaciones o ataques).

  Para lograrlo, los AIS aprenden y generan múltiples "anticuerpos" o detectores, que son reglas o patrones que caracterizan colectivamente el comportamiento normal o anómalo, basándose en un umbral de afinidad.

  Se utilizan comúnmente en ámbitos de ciberseguridad para la detección de malware, virus e intrusiones en redes, en diagnóstico médico para identificar células anómalas, en detección de fraudes financieros o en la monitorización industrial para identificar fallos en maquinaria, donde la identificación temprana de desviaciones es crítica.

Utilidades de Python para PE - DEAP

DEAP es adaptable a la complejidad de los sistemas de aprendizaje basados en reglas. permite diseñar un algoritmo genético para RBML de forma muy granular:

- Se debe definir la estructura de la regla (podría ser una clase o una tupla).
- Se define una función para crear una regla aleatoria.
- Se define la forma en que los conjuntos de reglas (individuos) se construirán (como una lista de reglas).
- Se proporcionas una función para crear un conjunto inicial de reglas para un individuo.
- Se registras cómo se creará una población inicial de estos conjuntos de reglas.
- Y se tiene que registrar las funciones de evaluación (para calcular el fitness de cada conjunto de reglas basándose en su rendimiento), cruce y mutación específicas para reglas.

Utilidades de Python para PE - DEAP

En el contexto de los sistemas RBML, donde el objetivo es evolucionar un conjunto óptimo de reglas que puedan, por ejemplo, clasificar datos o predecir resultados, DEAP proporciona el andamiaje necesario para:

1. Representar Reglas y Conjuntos de Reglas: cada "individuo" en la población de DEAP se convierte en un *conjunto de reglas* (o un único clasificador basado en reglas, dependiendo de cómo se diseñe).

2. Definir la Aptitud (Fitness): cómo de "buenas" son estas reglas. Para RBML, la aptitud suele ser la precisión de clasificación, la cobertura de los datos, la complejidad del conjunto de reglas, etc.

3. Operadores Evolutivos Específicos: DEAP permite definir cómo se "mutan" las reglas (cambiar una condición, agregar/quitar una cláusula) o cómo se "cruzan" conjuntos de reglas (intercambiar reglas entre dos conjuntos).

4. Algoritmos de Búsqueda: Implementar fácilmente algoritmos como Algoritmos Genéticos (GA) para buscar el conjunto de reglas más efectivo en un espacio de búsqueda grande.

Utilidades de Python para PE - DEAP

Explicación de DEAP, módulos principales (“from deap import base, creator, tools, algorithms”)

base:

- Contiene las clases base y las herramientas fundamentales. Aquí se encuentra base.Fitness para definir la aptitud de un individuo y base.Toolbox un contenedor de funciones que se registran para ser utilizadas en el algoritmo.
- Rol en RBML: define la estructura básica de cómo se medirá la "bondad" de un conjunto de reglas (Fitness) y proporciona el "lugar" central (Toolbox) para registrar todas las funciones personalizadas que crean y manipulan las reglas.

creator:

- Permite crear clases en tiempo de ejecución y definir la estructura de los individuos (por ejemplo, una lista, un árbol, un diccionario) y su fitness (maximizar o minimizar uno o varios objetivos).
- Rol en RBML: se usa para especificar qué forma tendrá un "individuo" (un conjunto de reglas, probablemente una list de objetos Rule) y cómo se medirá su rendimiento (su Fitness, por ejemplo, "maximizar la precisión de clasificación").

tools:

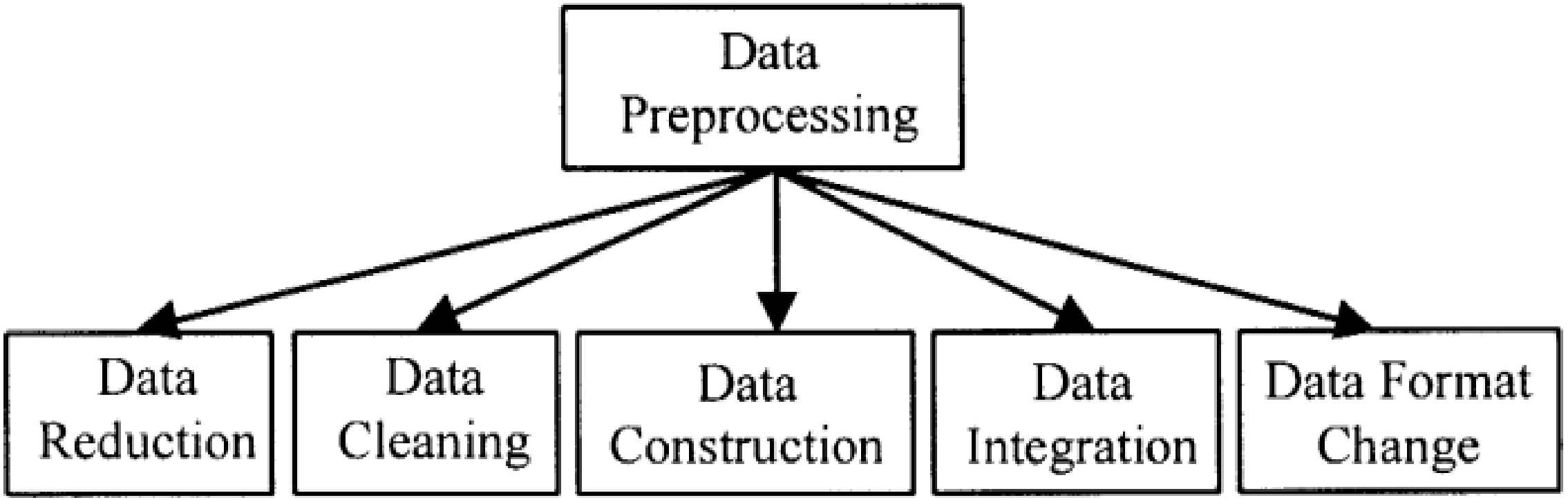
- Contiene una vasta colección de operadores predefinidos que son comunes en los algoritmos evolutivos: inicialización, selección, cruce, mutación
- Rol en RBML: a menudo se usan operadores de tools para seleccionar los mejores conjuntos de reglas para la siguiente generación.

algorithms:

- Proporciona los algoritmos evolutivos de "alto nivel" ya implementados. Estos algoritmos orquestan el flujo completo del proceso evolutivo.
- Rol en RBML: una vez que has definido tus individuos, su aptitud y tus operadores, usas un algoritmo de este módulo para ejecutar la evolución que buscará los mejores conjuntos de reglas durante un número de generaciones.

Métodos Evolutivos en Problemas de Selección de Instancias

La selección de instancias es una técnica de preprocesamiento de datos que consiste en seleccionar un subconjunto de instancias, relevantes de un conjunto de datos de entrenamiento, esta técnica se guía por un criterio de irrelevancia o ruido para descartar datos. El propósito es reducir el tamaño del conjunto de datos, de modo que por ejemplo en un problema de reconocimiento de patrones se llegue a una solución con la misma efectividad que se lograría al usar todos los datos.



Problemas Multiobjetivo

En muchos problemas de optimización del mundo real, se deben optimizar múltiples objetivos simultáneamente. Estos problemas se conocen como problemas multiobjetivo (MOP). A diferencia de los problemas de optimización mono objetivo, los MOP no tienen una única solución óptima, sino un conjunto de soluciones que pueden incluso representar diferentes compromisos e incluso conflictos entre los objetivos.

$$MOP = \begin{cases} \text{minimizar } (o\ maximizar) F(x) = \left(f_1(x), f_2(x), \dots, f_k(x)\right) \\ x \in X \\ restricciones\ en\ X\ \left\{x | g_i(x) \geq 0, 0 \leq i \leq m;\ h_i(x) = 0, 0 \leq i \leq e\ ;\ otras\right\} \end{cases}$$

Donde:

- $k \geq 2$, es el número de funciones objetivo.
- $x = \{x_1, \dots, x_n\}$ es el vector de variables de decisión.
- $y = \{y_1, \dots, y_k\}$, $y_i(x) = f_i(x)$ valor de una solución en el espacio de soluciones factibles.
- X es el espacio de soluciones factibles.
- Y es el espacio objetivo. Donde usualmente $Y \subseteq R^k$.
- $F(x) = \{f_1(x), \dots, f_k(x)\}$, es el conjunto de funciones objetivo.
- $g_i(x)$ y $h_i(x)$ restricciones que existen en X.

Problemas Multiobjetivo

Característica	*Problema Multimodal*	*Problema Multiobjetivo*
Objetivos a Optimizar	Uno	Dos o más
Naturaleza de "Múltiple"	Múltiples óptimos (locales/globales) en una sola función objetivo	Múltiples funciones objetivo a considerar
Meta de Optimización	Encontrar el óptimo global (o varios buenos óptimos)	Encontrar el frente de Pareto (conjunto de compromisos)
Solución Típica	Una única solución (o pocas)	Un conjunto de soluciones
Conflicto	No hay conflicto entre objetivos (solo hay uno)	Objetivos a menudo en conflicto

Problemas Multiobjetivo

Diseño de Producto/Ingeniería

Problema Multimodal (Single-objective Multimodality):

Contexto: Un equipo de ingeniería está diseñando una pieza mecánica y su único objetivo es maximizar la resistencia a la fatiga del material.

Naturaleza "Múltiple": Durante la fase de optimización, descubren que existen varias combinaciones distintas de aleación y tratamiento térmico que resultan en una resistencia a la fatiga máxima (valores pico). Por ejemplo, una aleación A con tratamiento térmico X da una alta resistencia, pero también una aleación B con un tratamiento térmico Y diferente da una resistencia similar e igualmente alta. Estas combinaciones representan "óptimos locales" o "regiones óptimas" en el espacio de diseño para la única función objetivo de resistencia.

Meta y Solución: La meta es encontrar la configuración específica (aleación y tratamiento) que dé la resistencia globalmente más alta. La solución típica sería una única combinación o un puñado de combinaciones equivalentes que alcancen el máximo.

Conflicto: No hay conflicto porque solo se busca mejorar una característica (resistencia a la fatiga).

Problema Multiobjetivo:

Contexto: El mismo equipo de ingeniería está diseñando una pieza mecánica, pero ahora sus objetivos son maximizar la resistencia a la fatiga Y minimizar el costo de producción Y minimizar el peso.

Naturaleza "Múltiple": Hay tres funciones objetivo distintas a considerar (resistencia, costo, peso).

Meta y Solución: Estos objetivos suelen estar en conflicto (una pieza más resistente y ligera podría ser más cara). La meta es encontrar un "frente de Pareto" de soluciones, es decir, un conjunto de diseños donde no se puede mejorar un objetivo sin empeorar al menos uno de los otros. Por ejemplo, podrían tener una solución muy resistente pero cara, otra barata, pero pesada, y varias intermedias. La solución es un conjunto de estas opciones de compromiso.

Conflicto: Sí, los objetivos están a menudo en conflicto.

Problemas Multiobjetivo

Optimización de Rutas/Logística

Problema Multimodal (Single-objective Multimodality):

Contexto: Una empresa de transporte quiere encontrar la ruta que minimice el tiempo de viaje para un camión desde el almacén A hasta el punto de entrega B.

Naturaleza "Múltiple": Debido a la complejidad de la red vial (calles de un solo sentido, intersecciones, diferentes límites de velocidad, posibles desvíos), puede haber múltiples rutas distintas que tomen aproximadamente el mismo tiempo mínimo total para llegar al destino. Una ruta podría ser por autopista más larga pero rápida, y otra por calles secundarias, más corta, pero con más semáforos, ambas resultando en tiempos totales muy similares o idénticos. Cada una de estas rutas representa un "óptimo local" en términos de tiempo.

Meta y Solución: La meta es encontrar la ruta más rápida posible. La solución es tipicamente la ruta "única" o "pocas" rutas que tienen el tiempo de viaje más bajo globalmente.

Conflicto: No hay conflicto, solo se busca el tiempo mínimo.

Problema Multiobjetivo:

Contexto: La empresa de transporte quiere encontrar la ruta que minimice el tiempo de viaje Y minimice el costo del combustible Y minimice el número de peajes.

Naturaleza "Múltiple": Esto implica tres funciones objetivo distintas.

Meta y Solución: Una ruta más rápida podría usar más combustible o tener más peajes. Una ruta para ahorrar combustible podría ser más lenta. La meta es encontrar un conjunto de rutas de compromiso (frente de Pareto) que ofrezcan diferentes balances entre tiempo, costo de combustible y peajes. La solución será un conjunto de rutas entre las cuales el operador puede elegir la que mejor se adapte a sus prioridades en un momento dado.

Conflicto: Sí, los objetivos de tiempo, costo de combustible y peajes suelen estar en conflicto.

Dominancia

Para la optimización de un problema con múltiples objetivos, no existe una única solución, sino un conjunto de soluciones no dominadas, no existen soluciones mejores en todos los objetivos del problema; a este conjunto de soluciones se les denomina frente de Pareto

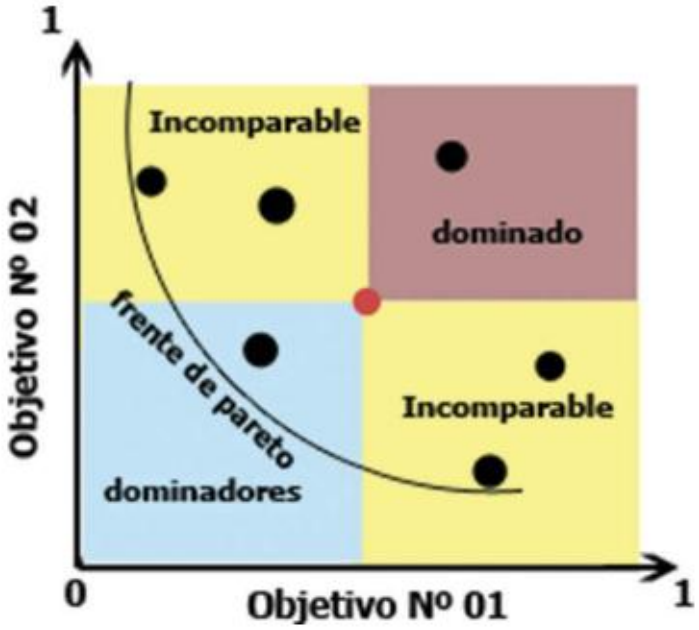
El concepto de dominancia de Pareto es fundamental en la optimización multiobjetivo. Se dice que una solución x_1 domina a otra solución x_2 si y solo si:

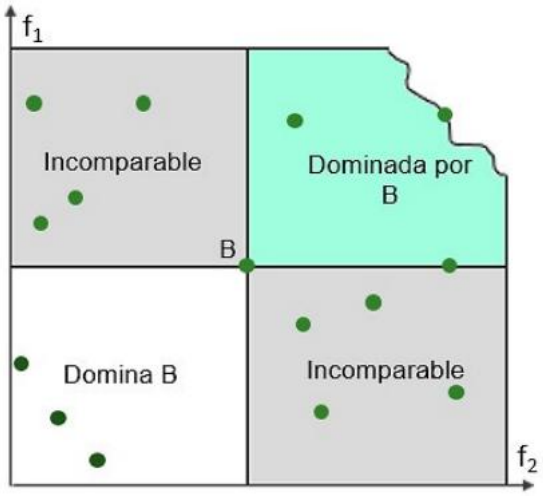
1. x_1 es al menos tan buena como x_2 en todos los objetivos.
2. x_1 es estrictamente mejor que x_2 en al menos un objetivo.

Matemáticamente, ***si tenemos un problema de minimización*** (en maximización sería ≥, >, ⋡) con m objetivos, la solución x_1 domina a la solución x_2 si:

$$x_1 \preccurlyeq x_2 \begin{cases} f_i(x_1) \leq f_i(x_2) \; para \; todo \; i = 1,2,\dots,m \\ y \\ f_j(x_1) < f_j(x_2) \; para \; al \; menos \; un \; j \in \{1,2,\dots,m\} \end{cases}$$

Dominancia - minimización





La clave para identificar la relación de dominancia es comparar las soluciones objetivo por objetivo. La dirección de esta comparación (mayor o menor) depende de si la meta es minimizar o maximizar cada objetivo. El Frente de Pareto, como la curva negra en la primera gráfica, está formado por el conjunto de soluciones que no son dominadas por ninguna otra solución en el espacio de objetivos.

Dados dos puntos (o soluciones) A y B en el espacio de objetivos (es decir, los valores de los objetivos para esas soluciones):

- A domina a B si:
  - A es al menos tan buena como B en *todos* los objetivos.
  - A es estrictamente mejor que B en *al menos uno* de los objetivos.
- B domina a A si B cumple las mismas condiciones respecto a A.
- A y B son incomparables si ninguna solución domina a la otra. Esto ocurre cuando A es mejor que B en algunos objetivos, pero B es mejor que A en otros (hay un *trade-off*).

Problemas multiobjetivo con pymoo

pymoo es un framework completo y extensible para la investigación y aplicación de la optimización evolutiva en Python, proporcionando una robusta arquitectura para definir, ejecutar y analizar problemas de optimización. Es útil para la optimización multiobjetivo (MOO) y mono objetivo, con un enfoque principal en los algoritmos evolutivos (EA). Su diseño es modular y extensible, facilitando la definición de problemas, algoritmos, operadores y criterios de terminación. https://pymoo.org/. Usos Principales:

- Definición y Resolución de Problemas de Optimización Multiobjetivo: permite modelar problemas complejos con múltiples objetivos conflictivos y encontrar un conjunto de soluciones de pareto óptimas.

- Benchmark de Algoritmos: facilita la comparación y evaluación de diferentes algoritmos de optimización sobre conjuntos de problemas estándar.

- Diseño de Algoritmos Personalizados: gracias a su diseño modular, permite a los investigadores y desarrolladores implementar fácilmente nuevos algoritmos, operadores o variantes combinando los componentes existentes.

- Análisis de Resultados: Ofrece herramientas y la estructura de resultados adecuada para el análisis de frentes de Pareto, convergencia y diversidad de soluciones.

- Optimización uniobjetivo: aunque se centra en multi-objetivo, pymoo es perfectamente capaz de resolver problemas con un solo objetivo y/o con restricciones.