UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

PROYECTO FIN DE CARRERA

EXPERIMENTACIÓN CON ALGORITMOS GENÉTICOS PARA LA BÚSQUEDA DE FUNCIONES DE DISTANCIA EN RNBR

Autor: Javier López Herradón
Tutores: José María Valls Ferrán
Ricardo Aler Mur

ÍNDICE

CAPÍTULO 3: FUNDAMENTOS DE REDES DE NEURONAS (RNBR)

CAPÍTULO 4: DESCRIPCIÓN DEL SISTEMA EMPLEADO

CAPÍTULO 5: EXPERIMENTACIÓN

CÁPITULO 6: CONCLUSIONES Y FUTURAS LÍNEAS DE INVESTIGACIÓN

1. INTRODUCCIÓN

En este primer capítulo se van a exponer los objetivos generales hacia los cuales está orientado el proyecto y se desglosarán haciendo un breve resumen del contenido de los mismos, los capítulos de los que está compuesto, para posteriormente analizar de manera mas extensa cada uno de ellos.

Desde que existe el ser humano, de manera habitual, le han ido surgiendo problemas en las distintas áreas del conocimiento. Estos problemas consisten fundamentalmente en la mejora de soluciones de las cuales se obtiene algún beneficio, ya sea a nivel particular o común.

La optimización es la disciplina que centra sus estudios en este tipo de problemas y sus posibles alternativas.

Si nos fijamos bien, en la naturaleza desde siempre ha existido el principio de supervivencia, que como bien sabemos consiste en que sobreviven los individuos mas aptos y a partir de ellos surgen nuevos individuos con características semejantes, es a lo que llamamos evolución.

La computación evolutiva es la aplicación que mediante la implementación de algoritmos inspirados en ese principio de supervivencia llevan a simular en computadores un proceso de evolución.

Los algoritmos evolutivos tienen como objetivo principal “evolucionar individuos”, estos individuos normalmente representan soluciones a un determinado problema de optimización, de tal manera que los individuos irán evolucionando generación tras generación basándose en el principio de supervivencia del que ya hemos hablado, es decir evolucionan los mas aptos.

En cada nueva generación se podrá comprobar como los individuos que la conforman poseen características mejores que los individuos de la generación anterior, acercándose con cada iteración a la solución óptima del problema.

1.1.- Resumen y objetivos del proyecto

Cuando se habla de redes de neuronas, se debe saber que son un sistema de computación que utilizando en una serie de muestras o ejemplos es capaz de aprender para posteriormente, basándose en ese aprendizaje, estar preparado para reaccionar de forma óptima ante otros estímulos.

Existe gran variedad de redes de neuronas, en este caso, nos vamos a centrar en las redes de neuronas de base radial (RNBR), las cuales utilizan funciones de activación de base radial como lo es la función normal (gaussiana).

Habitualmente se utilizan distancias euclídeas, que son simétricas en todas las dimensiones del espacio de entrada y por lo tanto no existe ponderación alguna que dé más importancia a unas entradas sobre otras.

El objetivo del proyecto es el de experimentar en diferentes dominios, utilizando distancias euclídeas generalizadas de forma que los resultados puedan compararse con los obtenidos con RNBR que utilizan distancias euclídeas clásicas. Las distancias generalizadas permiten ponderar entradas y así dar mayor relevancia a unas sobre otras y establecer las dependencias que puedan existir entre ellas. La función de distancia generalizada que se utiliza en el proyecto sigue la estructura de la distancia de Mahalanobis y vendrá representada mediante una matriz de distancias.

Basándose en la implementación de un algoritmo genético de un proyecto anterior, los valores de la matriz de distancias, son hallados de forma que son adecuados y se adaptan al problema para el que se quiera utilizar la red de neuronas. Gracias a estos valores calculados se intenta conseguir que la red de neuronas obtenga el mínimo error posible.

El Algoritmo genético va a ir probando distintas matrices de distancia utilizando la red de neuronas para su evaluación, de manera que poco a poco irá obteniendo cada vez mejores matrices con las que se consiga que la red de neuronas cometa menor error. La red de neuronas, fue modificada para que fuese capaz de utilizar funciones gaussianas con la distancia de distancias en lugar de la distancia euclídea como ocurre normalmente.

Al experimentar con diferentes dominios, se comparará si el sistema obtiene mejores o peores resultados que las redes de neuronas de base radial y se harán comparativas de los resultados obtenidos mediante la modificación de diversos parámetros.

La implementación del sistema nos permite realizar distintos tipos de pruebas pudiendo decidir si la matriz de distancias empleada será diagonal o simétrica, si el problema será de clasificación o aproximación, si usa validación cruzada o no, permitiendo en el primer caso seleccionar el número de particiones. También permite modificar el tamaño de la población, el número de generaciones posibles, si se utiliza elitismo o no…

Modificando estos parámetros y otros, se irán realizando pruebas y se irán comparando los resultados para cada uno de los dominios que se analizan en este proyecto: Diabetes, Ripley, Rectas 45, Rectas 0, Aleatorio y Aleatorio girado.

1.2.- Estructura del proyecto

Se exponen a continuación, a grandes rasgos, los contenidos de cada uno de los capítulos que conforman la estructura del proyecto:

- **Capítulo 1º : Introducción**

  En este primer capítulo se han expuesto los objetivos principales del proyecto y de forma resumida los conceptos que mas se van a emplear a partir de este momento como algoritmos genéticos, redes neuronales de base radial o funciones de distancia y que en capítulos posteriores se irán desarrollando para una mejor comprensión del sistema empleado.

Capítulo 2º : Algoritmos genéticos

El segundo capítulo comienza con una introducción a los Algoritmos genéticos, que son empleados como sistema de búsqueda y optimización, para posteriormente repasar su evolución desde los años 50 hasta nuestros días y se comparará su funcionamiento con el de la evolución en la vida real.

Se hará hincapié en los distintos tipos de codificación existentes así como en los distintos operadores genéticos y diferentes métodos de selección.

- **Capítulo 3º : Fundamentos de redes de neuronas**

El tercer capítulo presenta una comparativa entre el concepto de neurona biológica y neurona artificial y como en el caso de los algoritmos genéticos hace un repaso a la evolución histórica de las redes de neuronas.

Posteriormente se centra en las redes de neuronas de base radial, exponiendo sus características principales, arquitectura y los distintos tipos de aprendizaje.

Para terminar hace referencia al uso y características de las distancias de Mahalanobis y los distintos tipos de distancias euclídeas.

- **Capítulo 4º : Descripción del sistema empleado**

En el cuarto capítulo, se expone de manera general el funcionamiento del sistema que se va a emplear a la hora de llevar a cabo la experimentación con los dominios.

Se describe la estructura de alto nivel del sistema y se incluyen diagramas explicativos y descripciones de aquellos aspectos necesarios para la comprensión del funcionamiento del sistema.

Finalmente se hace una descripción de los ficheros que utiliza el sistema para su correcto funcionamiento, detallando la función de cada uno de los parámetros del fichero de configuración.

- **Capítulo 5º : Experimentación**

El quinto capítulo, es el más extenso de todo el proyecto, en él se explican y detallan los resultados de la experimentación realizada sobre los dominios propuestos.

Para cada dominio se realiza una experimentación previa para calcular el número óptimo de neuronas a utilizar y posteriormente se hacen dos pruebas, la primera utilizando matrices diagonales y la segunda utilizando matrices simétricas.

Aunque no es del todo exacto, en ocasiones se ha denominado "matriz de Mahalanobis" a la matriz simétrica, puesto que así se la denominaba en el proyecto fin de carrera en el que está basado este proyecto.

Además se presentan gráficas de la evolución el algoritmo genético para cada una de las particiones en caso de que se haya utilizado validación cruzada, se exponen las matrices utilizadas y la evolución de la tasa de acierto del sistema comparándola con la obtenida utilizando distancia euclídea.

- **Capítulo 6º : Conclusiones y futuras líneas de investigación**

  En el último capítulo se hace una recopilación de los resultados obtenidos en el capítulo anterior y se exponen las conclusiones generales sobre como se ha comportado el sistema tras la fase de experimentación.

  Finalmente se incluyen las mejoras que podrían hacerse en el sistema y las futuras líneas de investigación.

2. ALGORITMOS GENÉTICOS

A lo largo de este capítulo se expondrán de forma detallada los aspectos generales en lo que se refiere a los algoritmos genéticos (AG) y que constituyen una parte fundamental del proyecto.

Se estudiará su uso como sistema de búsqueda y optimización, su evolución histórica y las similitudes y diferencias fundamentales al relacionar la terminología de los algoritmos genéticos en relación a la empleada en la genética biológica.

Se explicará de forma mas detallada el concepto de operador genético, los tipos de codificación y algunos fundamentos teóricos que ayudarán a comprender mejor que es un algoritmo genético.

2.1 – Introducción a los algoritmos genéticos

2.1.1 – Concepto y razones para su estudio

Si hubiera que responder de forma sencilla a qué son los algoritmos genéticos (AG), se diría que son algoritmos de búsqueda que se basan en la genética y la selección natural.

Los AG fueron desarrollados por John Holland [Holland, 75], que basándose en los principios básicos de la naturaleza crearon algoritmos de optimización que posteriormente eran utilizados con éxito para la resolución de gran cantidad de problemas.

Charles Darwin [Darwin, 59] describió en su teoría de la evolución, en la que se basan los principios básicos de los algoritmos genéticos que:

- Existe una población de individuos con diferentes propiedades y habilidades.
- Existe una limitación sobre el número de individuos que componen una determinada población.
- La naturaleza crea nuevos individuos con propiedades similares a los individuos existentes.
- Los mejores individuos se seleccionan de manera más habitual para la reproducción de acuerdo con la selección natural.

2.1.2 – Evolución histórica

La razón principal de que en las décadas de los 50 y 60, algunos científicos estudiaran los sistemas evolutivos era la idea de que la evolución podía ser utilizada como herramienta de optimización para problemas de ingeniería.

Los sistemas evolutivos tienen como idea común la de que, para un problema dado construir una población de soluciones candidatas utilizando para ello operadores que se basan en la selección natural y en la variación genética. Los principales campos que comprenden la computación evolutiva son los algoritmos genéticos y la programación evolutiva, junto a las estrategias evolutivas.

A finales de los años 60, Rechenberg [Rechenberg, 73] escribió “Estrategias de la Evolución” (Evolution Strategies). En él, describe un método utilizado para optimizar los valores de los parámetros de algunos dispositivos. Desde su publicación el campo de la utilización de estrategias basadas en la evolución se ha convertido en un área activa de investigación. La mayor parte de estas estrategias fueron desarrolladas de forma independiente del campo de los algoritmos genéticos.

La programación evolutiva [Fogel, 66], es otra técnica evolutiva que se basa en que cada una de las soluciones candidatas están representadas como una máquina de estado finita, en las cuales se muta de manera aleatoria sus diagramas de transición de estados. Esta técnica fue desarrollada por Fogel, Owens y Wash en 1966.

John Holland ideó en los años 60 los algoritmos genéticos y durante finales de los 60 y gran parte de los 70 los desarrolló junto con sus estudiantes de la Universidad de Michigan [Holland, 75]. El fin de Holland no era diseñar algoritmos para resolver problemas específicos como hacen las estrategias y la programación evolutiva, sino que Holland estudió el fenómeno de la adaptación como ocurre en la naturaleza y desarrolló métodos en los que los mecanismos de la adaptación natural podían ser utilizados en sistemas de computación. En “Adaptación en Sistemas Naturales y Artificiales” de Holland, se presentó el algoritmo genético como una abstracción de la evolución genética.

El algoritmo genético de Holland es un método que sirve para pasar de una población de cromosomas (cadenas de ceros y unos) a una nueva población, usando junto con los operadores inspirados en la genética de cruce, mutación e inversión una especie de selección natural.

Cada cromosoma está compuesto de genes (bits), de manera que cada uno de los genes es una instancia de una alelo (0 ó 1). El operador de selección escoge dentro de la población los cromosomas que van a ser utilizados para reproducirse, de modo que basándose en la media, los cromosomas que se ajusten mejor a esta media terminarán produciendo más hijos.

La principal novedad que introdujo Holland fue un algoritmo que se basa en una población donde existe cruce, inversión y mutación. El cruce intercambia parte de dos cromosomas de manera bastante parecida a la recombinación entre organismos con dos cromosomas simples (haploides). La mutación cambia de manera aleatoria los valores del alelo en alguna de las partes del cromosoma. La inversión invierte el orden de una sección contigua del cromosoma.

En años posteriores este tipo de algoritmo se fue difundiendo cada vez más y los investigadores estudiaron distintos métodos de computación evolutiva. A día de hoy, se utiliza el término “algoritmo genético” para describir algo bastante alejado de la idea que tuvo Holland.

2.1.3 – Genética biológica

En el contexto de los algoritmos genéticos, la terminología biológica que va a ser empleada a lo largo del proyecto está inspirada en la biología real. Los organismos vivos presentes en la naturaleza están constituidos por células y cada una de ellas contiene cadenas de ADN (uno o más cromosomas). De manera conceptual un cromosoma puede ser dividido en genes, cada uno de ellos codifica una proteína.

En términos generales se puede decir que un gen se codifica como si fuese un rasgo (color de piel). Las diferentes disposiciones del rasgo se denominan alelo. Por ejemplo, existe un gen que codifica el color de los ojos. El gen puede tomar diversos valores (alelos), tantos como colores de ojos haya. Cada gen, dentro del cromosoma, ocupa una posición particular.

Algunos organismos, en cada una de sus células tienen muchos cromosomas. El conjunto de todos los cromosomas del organismo es a lo que se llama genoma. El genotipo es el término que hace referencia al conjunto particular de genes que se encuentran en el genoma. Si dos individuos tienen genomas idénticos, se dice que tienen el mismo genotipo. El genotipo da lugar mediante el desarrollo del organismo al fenotipo, que hace referencia a las características físicas y psíquicas del individuo (altura, color de ojos, inteligencia…).

Los organismos pueden ser Diploides (sus cromosomas se conforman en pares) o Haploides. En la naturaleza, la gran parte de las especies que se reproducen de forma sexual son diploides, ya que cada uno tiene 23 pares de cromosomas en cada célula del cuerpo. En este grupo se incluyen los seres humanos. Durante la reproducción se produce una recombinación, que consiste en que en cada antecesor, los genes son intercambiados entre cada par de cromosomas para formar un gameto, y es entonces cuando los gametos de los dos padres crean un conjunto completo de cromosomas diploides. En la reproducción sexual haploide, los genes son intercambiados entre los dos antecesores con cromosomas de una sola hebra. En algunas ocasiones, los hijos sufren una mutación, en la que uno de los nucleótidos es cambiado.

La adecuación o fitness de un organismo, la definimos de manera normal como la probabilidad de que el organismo viva para poder reproducirse o una función del número de sucesores que tenga el organismo (fertilidad). La teoría de la evolución por selección natural afirma que la adaptación de las especies a su entorno se produce gracias al cambio en el código genético de los individuos a lo largo de las eras geológicas. Dicho cambio se produce gracias a diversos tipos de mutación. De alguna manera, la evolución genera individuos bien adaptados, o lo que es lo mismo, que maximizan su fitness. Este proceso de maximización u optimización es lo que se pretende imitar en un algoritmo genético.

En términos de un algoritmo genético, cromosoma hace referencia a una solución candidata a un problema y que normalmente está codificado como una cadena de bits. Los genes son cada bit o bloques de bits adyacentes que codifican un número particular de la solución candidata. Así pues, una población en un algoritmo genético es un conjunto de cromosomas, o lo que es lo mismo, de cadenas de bits. La población irá

cambiando con el transcurso de las generaciones para optimizar una función que denominaremos función de *fitness*.

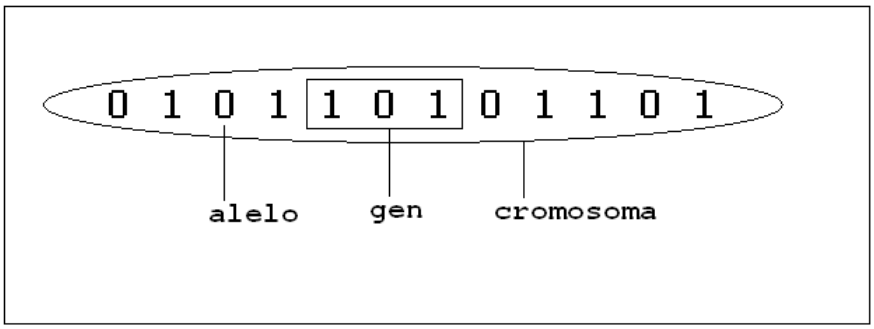


Figura 2.1.3 – 1 Alelos, genes y cromosomas

2.1.4 – Ejemplo de algoritmo genético

A continuación se va a describir el algoritmo básico que constituye el núcleo del algoritmo a partir del cual se pueden construir la mayor parte de los algoritmos genético. Dado un problema definido, para ser resuelto mediante una representación de soluciones como cadena de bits, un algoritmo sencillo trabaja de la siguiente manera:

En primer lugar, se comienza con una población generada aleatoriamente de cromosomas (o cadenas de bits). A continuación se calcula el valor de la función de adecuación *f(x)* (o *fitness*) para cada cromosoma x en la población. Posteriormente han de repetirse los siguientes pasos hasta que se hayan generado n hijos:

- Seleccionar un par de cromosomas progenitores desde la población actual (La probabilidad de ser seleccionado aumenta de manera proporcional a su adecuación). Esta selección se realiza con reemplazamiento, lo que significa que el mismo cromosoma puede ser seleccionado una o varias veces como progenitor.

- Con una probabilidad de cruce p_c, en un punto dado se cruza el par de cromosomas para formar dos hijos. En el caso de que no se produjese ningún cruce, ambos hijos serían copias exactas de sus respectivos padres.

- Se mutan los dos hijos con una probabilidad de mutación en cada posición y se colocan los cromosomas resultantes en la nueva población.

Finalmente se intercambia la población actual con la nueva población y se vuelve a repetir el proceso a partir del cálculo del valor de la función de adecuación para cada cromosoma x en la población nueva. A cada una de las iteraciones de este proceso la llamamos generación. Un algoritmo genético típico se itera de 50 a 500 generaciones. Al conjunto de generaciones se le llame ejecución. Al terminar la ejecución existen

normalmente uno o varios cromosomas que están adecuados de manera destacada en la población y que son elegidos como solución al problema inicial.

Más adelante se explicará en detalle como se llevan a cabo la mutación y el cruce.

2.2 – Codificación de problemas para algoritmos genéticos

Para tener éxito a la hora de utilizar un algoritmo genético, así como en cualquier problema de búsqueda y aprendizaje, hay que asegurarse de que el modo en el que se codifica cada solución candidata es el correcto, ya que aunque no es el factor principal es uno de los ejes del método utilizado.

La mayor parte de las aplicaciones con algoritmos genéticos emplean cadenas de bits que son de longitud fija y en un orden fijado para la codificación de soluciones candidatas. En los últimos años se ha realizado experimentación con otros tipos de codificaciones.

2.2.1 – Codificación binaria

Las codificaciones que se dan más habitualmente son las binarias (cadenas de bits), esto es debido en parte a que en su primer trabajo, Holland y sus estudiantes se centraron en este tipo de codificación y las prácticas en algoritmos genéticos siguieron este camino. La teoría de los algoritmos genéticos que existe actualmente se basa, en su mayor parte, en la suposición de tener una codificación binaria con una longitud y un orden fijo. Estas teorías pueden extenderse a aplicar codificaciones no binarias, pero no están tan bien desarrolladas como la teoría original.

Se debe tener en cuenta también que las heurísticas sobre los valores apropiados de los parámetros (probabilidad de cruce y mutación…) han sido normalmente desarrolladas bajo una codificación binaria. A lo largo de los años han ido apareciendo muchas extensiones al esquema binario básico, entre otros cabe destacar la codificación gris (gray coding) de Bethke [Bethke, 80] y Caruana, [Caruana, 88], y el esquema de codificación binaria de diploides de Hillis.

Holland [Holland, 75] le dio a la utilización de codificaciones binarias una justificación teórica al comparar dos codificaciones que aproximadamente tenían la misma información. La primera codificación, por ejemplo, tenía un número pequeño de alelos y largas cadenas y la segunda un gran número de alelos y cadenas cortas. Un ejemplo de cada una sería:

Codificación 1: Cadenas de bits de longitud 100
Codificación 2: Cadenas de números decimales de longitud 30

Tras estudiar ambas codificaciones, Holland argumentó que la primera de ellas permitía un grado mas alto de paralelismo implícito que la segunda, ya que una instancia de la primera contenía mas esquemas que la segunda (2^{100} vs. 2^{30}).

Aunque como ya hemos visto este tipo de codificación tiene sus ventajas, también hay que saber que entre sus desventajas se encuentran el que este tipo de codificaciones no son naturales y presentan varios problemas, además de ser propensas a ciertas ordenaciones arbitrarias.

2.2.2 – Codificación mediante caracteres y valores reales

En el caso de algunas aplicaciones, la forma más natural de codificación es el uso de un alfabeto de algunos caracteres o números reales para formar cromosomas. El esquema de Holland implica que se obtiene peor rendimientos con esos algoritmos genéticos que utilizando codificación binaria. Actualmente se considera que hay que utilizar la codificación que resulte más natural para el problema. En ocasiones esa codificación será la binaria [Janikow, 91] [Wright, 91] y en otros casos la solución se codificará como una lista de números reales.

2.2.3 – Codificación en árbol

Los esquemas de codificaciones tienen bastantes ventajas, entre ellas la de que permiten acotar el espacio de búsqueda, ya que en principio cualquier tamaño de árbol puede formarse vía cruce o mutación. Pero no todo son ventajas al hablar de la posibilidad de acotación ya que muchas veces genera algunas dificultades potenciales.

Los árboles normalmente crecerán de forma incontrolada, previniendo la formación mas soluciones candidatas estructuradas y jerárquicas, además los árboles resultantes tienden a ser bastante grandes por lo que pueden llegar a ser difíciles de simplificar y comprender.

Los experimentos que tratan de evaluar la utilidad de las codificaciones en árbol comparándolos con otros tipos de codificación se encuentran en fase de comienzo entre los investigadores del desarrollo de aplicaciones genéticas.

2.3 – Operadores genéticos

Un algoritmo genético en su forma más simple, está compuesta por tres operadores genéticos: reproducción, cruce o crossover y mutación.

En los siguientes apartados vamos a estudiar de forma general el funcionamiento del operador de reproducción, y de una manera más extensa y detallada los operadores de cruce y mutación y así ver sus variantes más comunes.

También estudiaremos otros aspectos teóricos de interés y necesarios para la correcta comprensión de los operadores y diferentes métodos de selección.

2.3.1 – Tipos de operadores genéticos

2.3.1.1 – Operadores genéticos (I): Reproducción

La reproducción consiste en que un individuo pase a la siguiente generación sin modificación. Así pues, en la nueva generación habrá individuos que vienen directamente de la generación anterior, además de individuos modificados mediante mutación y cruce, como se explica a continuación.

2.3.1.2 – Operadores genéticos (II): Cruce (Crossover)

El operador de Cruce (en inglés: crossover) elige una posición del cromosoma e intercambia subsecuencias antes y después de esta posición entre dos cromosomas para así crear dos cromosomas sucesores. Aproximadamente, el operador cruce intenta realizar la función de la recombinación biológica entre dos organismos con dos cromosomas sencillos (haploides).

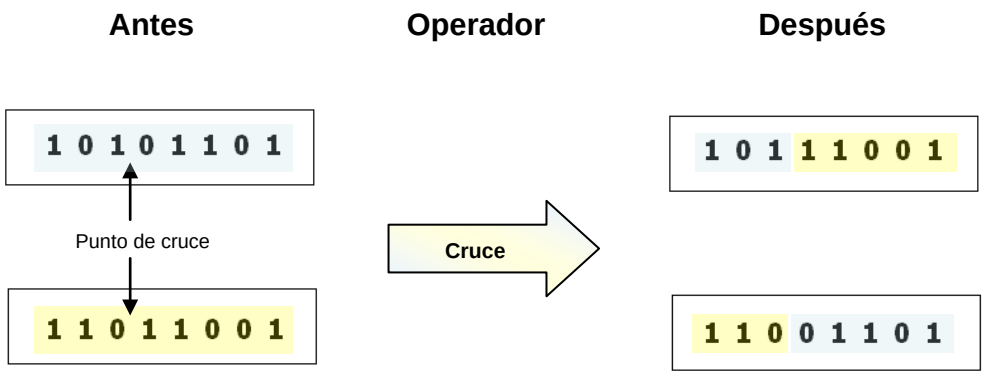


Figura 2.3.1.2-1: Ejemplo operador de cruce por un punto.

El uso de un operador de cruce es un rasgo principal distintivo entre algoritmos genéticos. Existen varios operadores de cruce: cruce unitario, cruce de dos puntos, cruce uniforme parametrizado…etc.

- Cruce unitario

  Es la forma más simple del operador de cruce, ya que se escoge una posición de cruce de forma aleatoria y las partes de los padres se intercambian para formar dos hijos.

  La idea principal es recombinar bloques de construcción o esquemas de las diferentes cadenas.

  El cruce unitario tiene algunos defectos, entre ellos el de que generalmente no se pueden combinar todos los esquemas posibles o que seguramente los esquemas con longitudes de definición largas tras realizar una operación de cruce unitario se destruirán, es lo que se denomina sesgo posicional (los esquemas que pueden ser creados o destruidos por un cruce dependen en gran medida de la posición que ocupen en los bits del cromosoma).

  El operador de cruce unitario asume que los esquemas cortos son funcionales para la construcción de bloques de cadenas, pero normalmente no se conoce qué ordenación de bits pueden agrupar la funcionalidad relacionada.

  Otra desventaja del operador de cruce unitario es que puede ser que no hay ninguna forma de poner todos los bits con funcionalidad relacionada juntos en una misma cadena, ya que hay bits determinados que pueden ser cruciales.

  El operador de cruce unitario tiene la tendencia a respetar los esquemas cortos y trata algunas posicionas de manera preferente ya que los segmentos intercambiados entre dos padres siempre contienen los extremos de las cadenas.

- Cruce por dos puntos

  Con la intención de reducir el sesgo posicional y el efecto del cruce unitario en los extremos de las cadenas, muchos investigadores utilizan el cruce de dos puntos, que consiste en seleccionar dos posiciones de manera aleatoria e intercambiar los segmentos existentes entre ellos.

  El cruce por dos puntos puede combinar más esquemas que el cruce unitario. Los segmentos que son intercambiados no contienen necesariamente los extremos de las cadenas, pero como en casos anteriores van a seguir existiendo esquemas que el cruce por dos puntos no va a ser capaz de combinar.

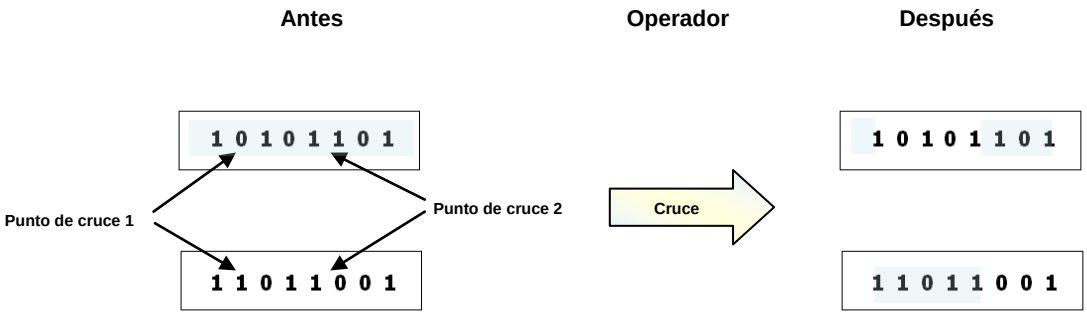


Figura 2.3.1.2-2: Ejemplo de operador de cruce por dos puntos.

- Otros tipos de operadores de cruce

  Los investigadores han experimentado en algoritmos genéticos con operadores con diferente número de puntos de cruce.

  Muchos opinan que el operador de cruce mas eficiente es el del cruce uniforme parametrizado, en el cual se producen un intercambio de bits en cada posición con una probabilidad p, normalmente de entre 0, 5 y 0,8. No tiene sesgo posicional, algunos esquemas contenidos en posiciones distintas en los padres pueden ser recombinados por los hijos.

No existe una respuesta clara a la pregunta de cuál es el tipo de operador de cruce que se debe emplear a la hora de resolver un problema, ya que el éxito o el fracaso del mismo va a depender de forma algo complicada de la función de adecuación que se utiliza, de la codificación y de otros aspectos del algoritmo genético.

En las aplicaciones de los algoritmos genéticos más comunes se suele emplear el operador de cruce por dos puntos o el parametrizado con una probabilidad de cruce de entre 0,7 y 0,8.h

2.3.1.3 – Operadores genéticos (III): Mutación

El operador de mutación cambia de forma aleatoria algunos bits de un cromosoma. Una mutación puede ocurrir en cada posición de una cadena con una probabilidad bastante baja normalmente del orden 1 de cada 1000 veces.

Figura 2.3.1.2-3: Ejemplo operador de mutación.

Bajo el punto de vista de la comunidad científica que investiga el funcionamiento de los algoritmos genéticos, el operador de cruce es la principal herramienta de variación en los algoritmos junto al operador de mutación que asegura nuevas poblaciones ante la falta de variabilidad en una posición en particular.

Esto es contraria a ideas tradicionales de otros métodos de computación evolutivos, en los cuales la mutación tiene una papel mas secundario al ser considerado solo una fuente de variación. En algunas versiones actuales se incluye a la mutación como una forma de cruce.

Se han realizado algunos estudios para comparar la influencia que puede ejercer una mutación respecto al cruce, ya que se está cogiendo aprecio al papel desempeñado por la mutación en los algoritmos genéticos.

Las conclusiones que se pueden sacar entre todos ellos es que lo más importante es encontrar un equilibrio entre selección, cruce y mutación. La importancia tanto del cruce como de la mutación puede ser relativa a lo largo de la ejecución del algoritmo genético.

2.3.2 – Esquemas

Cuando se utilizan algoritmos genéticos, puede que existan similitudes importantes entre las cadenas que tengan un alto grado de adecuación, de forma que éstas pueden ser utilizadas como guías a la hora de realizar una búsqueda. A esto es a lo que se denomina esquema.

Para una mayor comprensión por parte del lector, y puesto que no es un tema directamente relacionado con este proyecto, en el *Anexo A* se expone todo lo relacionado con este tema, por si fuera del interés del mismo.

2.3.3 – Métodos de selección

La forma de realizar la selección es una de las decisiones que se deben tomar para utilizar un algoritmo genético. Se debe escoger la forma en la que se seleccionan los individuos que crearán los hijos de la siguiente generación y cuantos hijos crearán cada uno de ellos. El objetivo de la selección será hacer todo lo posible para que se

seleccionen los individuos con el mayor grado de adecuación, para que sus hijos puedan alcanzar incluso un grado de adecuación mayor.

La selección debe estar equilibrada con la variación proporcionada con el cruce y la mutación, ya que una selección demasiado fuerte hará que los individuos con un mayor grado de adecuación local terminen copando la población, reduciéndose así el grado de diversidad que es necesario para que se produzca un cambio y un progreso. Sin embargo una selección demasiado débil puede hacer que la evolución sea demasiado lenta.

En los siguientes apartados vamos a ir viendo alguno de los métodos de selección existentes y a describirlos de forma breve.

2.3.3.1 – Selección proporcional a la adecuación

El método de selección proporcional a la adecuación es el algoritmo original de Holland y en él, el valor esperado de un individuo, que es el número esperado de veces que un individuo será seleccionado para reproducirse, es el valor de la función de adecuación del individuo dividido entre la adecuación media de la población:

$$\frac{f(Xi)}{\sum_{j=1}^{n} \overline{f(Xj)}}$$

Ecuación 2.3.3.1-1

, donde n es el número de individuos de la población.

Para la implementación del algoritmo de Holland el método mas común es el de muestreo de la ruleta, en el cual ha cada individuo se le asigna una porción proporcional al grado de adecuación del individuo. La ruleta se lanza n veces, siendo n el número de individuos de la población. En cada una de las tiradas, el individuo afortunado es seleccionado del conjunto de padres para la siguiente generación.

Este método obtiene el número esperado de hijos para cada individuo, sin embargo en poblaciones relativamente pequeñas usadas en algoritmos genéticos, el número actual de hijos esperados está bastante alejado del valor esperado.

En1987, James Baker [Baker, 87] propuso un nuevo método de muestreo, que es conocido con el nombre de Muestreo Estocástico Universal (SUS), en inglés Stochastic Universal Sampling, con la intención de minimizar la dispersión de los valores a la que hemos hecho referencia anteriormente. En términos generales este método consiste en lanzar la ruleta solo una vez pero con n indicadores que están igualmente espaciados y que son empleados para seleccionar n padres.

El método del muestreo estocástico universal no resuelve la mayoría de los problemas que suceden con la selección proporcional a la función de adecuación, ya que

normalmente en la búsqueda, la varianza de la adecuación de la población es bastante alta y un pequeño número de individuos tiene mucho mayor grado de adecuación que los demás. La convergencia prematura consiste en que con selección proporcional a la adecuación, ellos y sus descendientes se multiplicarán rápidamente en la población, llegando a desplazar a otros que aunque en primera instancia son poco idóneos, en generaciones futuras podrían llevarnos a la solución.

En conclusión, la selección proporcional a la adecuación tiene el fallo en que se centra de manera clara en explotar las cadenas que desde un principio tienen un mayor grado de adecuación, rechazando sin tener la posibilidad de dar marcha atrás las que tienen menor grado de adecuación pudiendo ser éstas las que en generaciones sucesivas alcancen el mayor grado de adecuación, siendo esa la verdadera solución al problema.

2.3.3.2 – Escalado sigma

Para tratar de resolver el problema del método de selección proporcional a la adecuación, los investigadores de algoritmos genéticos han experimentado con varios métodos de escalado, que consisten en hacer algoritmos genéticos menos susceptibles a la convergencia prematura, es decir al rechazo de las cadenas que en un principio parecen tener el menor grado de adecuación.

El método de escalado a sigma desarrollado por Forrest [Forrest, 85] en 1985 y Goldberg [Goldberg, 89] en 1989 guarda la presión de selección relativamente constante, es decir el grado en el cual los individuos con una adecuación muy alta se tienen en cuenta para tener hijos, a lo largo de la ejecución del algoritmo dependiendo siempre de la varianza del grado de adecuación de la población.

En un escalado a sigma, el valor esperado de un individuo está en función de su grado de adecuación, de la media de la población y de la desviación estándar de la misma. Cuando se comienza la ejecución de este método, normalmente la desviación estándar de las adecuaciones es alta y el escalado a sigma lo corrige disminuyendo dichas desviaciones. Por el contrario, a medida que se ejecuta el método, cuando la población ha convergido más y la desviación estándar es mucho menor, el método sigma realiza una corrección para destacar a los mejores individuos, permitiendo continuar a la evolución.

2.3.3.3 – Elitismo

Kenneth De John [De Jong, 75] en 1975 introdujo por primera vez el elitismo, que no es otra cosa que un complemento para muchos de los métodos de selección que fuerzan al algoritmo genético a retener a los mejores individuos de cada una de las generaciones. Los individuos que son retenidos pueden perderse si no son seleccionados para la reproducción o se ven afectados por un cruce o una mutación. En líneas generales los investigadores están de acuerdo en que el elitismo mejora de forma considerable el rendimiento de un algoritmo genético.

2.3.3.4 – Selección de Boltzmann

El escalado a sigma tiene la desventaja de que solamente guarda la presión de la selección más constante durante la ejecución del algoritmo. Sería bueno encontrar un método que permitiese guardar distintos grados de presión en diferentes momentos de la ejecución.

Es decir, sería bueno que al principio de la ejecución ser un poco menos estrictos a la hora de seleccionar los individuos para la reproducción dependiendo de su grado de adecuación. Con esto se conseguiría que individuos con un grado de adecuación menor se reproduzcan y en generaciones posteriores generen individuos con mayor adecuación.

Más tarde, si puede ser bueno irse centrando en los individuos con el grado de adecuación mayor, asumiendo que la diversidad inicial con poca selección ha permitido a la población evolucionar a la población por el espacio correcto de búsqueda.

Una idea parecida a esto es la Selección de Bolzmann, en la cual se produce una constante variación de la temperatura controlando la tasa de selección de acuerdo a una programación establecida anteriormente.

La temperatura empieza alta, mientras que la presión de selección es baja, por lo que cada uno de los individuos tiene una probabilidad razonable para poder reproducirse.

La temperatura se va reduciendo de manera gradual, lo que hace que a la vez se vaya elevando la presión de selección, permitiendo al algoritmo genético a partir de ese momento estrechar la mejor parte del espacio de búsqueda mientras se mantenga el grado de diversidad adecuado. [Goldberg, 90] [De la Maza, 91] [De la Maza, 93] [Prügel-Bennett, 94]

2.3.3.5 – Selección por ranking

La selección proporcional a la adecuación es usada normalmente en los algoritmos genéticos, debido a que forma parte de la proposición original de Holland, pero evidentemente y como ya hemos visto en apartados anteriores para algunas aplicaciones la selección proporcional a la adecuación sencilla requiere de muchos ajustes para que funcione de un modo mas o menos correcto. En los últimos años se han llevado a cabo aproximaciones que se alejan de la selección y que cada vez se están volviendo más comunes.

Este es el caso de la selección por ranking que es un método alternativo, que surge con el propósito de prevenir la convergencia prematura [Baker, 85] al igual que hemos visto en métodos anteriores. En el método de selección por ranking los individuos de la población se ponen en un ranking en función de su grado de adecuación, por lo que el valor esperado para cada individuo depende de su puesto en el ranking.

Con este método no se necesita escalar las adecuaciones ya que van a quedar ocultas las diferencias absolutas entre ellas, lo que tiene algunas ventajas como que el uso de adecuaciones absolutas puede llevar a problemas de convergencia, pero también desventajas puesto que a veces vendría bien saber que el grado de adecuación de un

individuo está muy cercano a su competidor. La selección por ranking trata de evitar coger la mayoría de los hijos de un pequeño grupo de individuos con una adecuación muy alta, por lo que esto hace que reduzca la presión de la selección cuando la varianza es alta.

Otra desventaja que puede aparecer en la selección por ranking es que disminuir la presión de la selección significa que al algoritmo genético en algún caso pueda ser demasiado lento para encontrar los individuos con mayor grado de adecuación. Sin embargo, en otros casos, la diversidad que proporciona el método puede llevarnos a un mayor éxito en la búsqueda, que si usáramos el método de selección proporcional a la adecuación que tiene el problema de la convergencia prematura.

2.3.3.6 – Selección por torneo

Como ya hemos visto en apartados anteriores, los métodos de selección que hemos visto hasta ahora presentan diferentes problemas.

En el caso de la selección proporcional a la adecuación, ésta requiere dos pasos a través de la población en cada una de las generaciones, en el primero se obtiene la adecuación media (en el caso del escalado sigma se obtiene la desviación estándar) y en el segundo se obtiene el valor esperado para cada uno de los individuos. En el método de selección por ranking, se consume demasiado tiempo ya que se requiere ordenar toda la población de acuerdo a su rango.

Estudiemos ahora el funcionamiento del método de selección por torneo. Éste método en términos de presión de la selección es similar a la selección por ranking, pero es mas eficiente en términos computacionales y mas fácil paralelizable.

El funcionamiento, de manera resumida, del método de selección por torneo consiste en la selección de dos o mas individuos y se elegirá para la reproducción el que mayor grado de adecuación tenga. Todos, incluido el elegido, se devuelve a la población original teniendo la posibilidad de volver a ser seleccionados. Este proceso se repite tantas veces como el tamaño de la población. Por ejemplo, si la población es de 50 individuos, habrá que realizar 50 torneos, para completar la siguiente generación. [Goldberg, 91]

2.3.3.7 – Selección por Steady-State o estado estacionario.

La mayor parte de los algoritmos genéticos son generacionales, lo que significa que cada una de las nuevas generaciones está formada por los hijos de la generación previa, por lo que algunos hijos pueden ser similares a los padres. En algunos esquemas, como los del elitismo, las generaciones posteriores pueden superponerse en algún grado, es decir, alguna parte de la generación anterior se mantiene en la nueva. [De Jong, 75]

La fracción de nuevos individuos en cada generación es conocida como diferencia generacional. En el caso del método de selección por Steady-State propuesto por Syswerda [Syswerda, 89] [Syswerda, 91] y De Jong, [De Jong, 93] solo unos pocos individuos son reemplazados en cada generación. Normalmente una pequeña cantidad de los peores individuos son reemplazados por los hijos resultantes de las operaciones de cruce y mutación de los mejores individuos.

2.4 – Parámetros de los algoritmos genéticos

A la hora de implementar un algoritmo genético, otra de las decisiones importantes que se deben tomar es que valores se va a dar varios parámetros como el tamaño de población, la tasa de cruce o la tasa de mutación. Normalmente estos parámetros están relacionados de forma no lineal y no pueden ser optimizados de forma simultánea. No existen convenio alguna entre la comunidad científica que determinen que valores son los mejores para dar a esos parámetros, por lo que normalmente se utilizan los que mejor hayan funcionado en experimentaciones anteriores. Se exponen a continuación algunos de los estudios más importantes con sus correspondientes resultados:

- En 1975, De Jong realizó uno de los primeros estudios sistemáticos de cómo podían afectar los parámetros a un algoritmo genético utilizando una corta colección de funciones de test. Este estudio reveló, que los mejores valores para el parámetro del tamaño de la población estaban situados entre 50 y 100 individuos, el mejor ratio de cruce era 0,6 por cada par de padres y la mejor tasa de mutación 0,001 por bit. Estos son los valores mas aceptados.

- En 1986, Grefensette se dio cuenta de que los algoritmos genéticos podían ser utilizados como procedimiento a la hora de optimizar los parámetros de otro algoritmo genético. En este caso, los resultados obtenidos fueron un tamaño de población de 30 individuos, una tasa de cruce de 0,95 y una tasa de mutación de 0,01. Estos valores no pueden ser considerados válidos generalmente, ya que se demostró que solo eran óptimos para diversas funciones de adecuación.

- En 1989, Schaffer, Caruana, Eshelman y Das, tuvieron un año de forma sistemática una CPU testeando un rango muy grande de combinación de varios parámetros. Los resultados que obtuvieron fueron muy parecidos a los conseguidos en 1986 por Grefensette, una población de entre 20 y 30 individuos, una tasa de cruce de entre 0,75 y 0,95 y una tasa de mutación de entre 0,005 y 0,01.

3. FUNDAMENTOS DE REDES DE NEURONAS (RNBR)

En este capítulo expondremos los aspectos fundamentales de las Redes de Neuronas de Base Radial (a partir de ahora, RNBR) que son las que se van a emplear para realizar la parte de experimentación del proyecto.

Así mismo veremos los fundamentos básicos de las redes de neuronas artificiales para entender mejor las ventajas, limitaciones y aplicaciones de las distintas arquitecturas que existen de neuronas artificiales.

3.1 – Redes de Neuronas

3.1.1 – Biológicas y artificiales

El cerebro humano se compone de billones de neuronas que están interconectadas entre sí formando redes con unas funcionalidades específicas.

Existen varios modelos que están basados en el comportamiento real del cerebro, en cómo está estructurado y en como funciona el sistema nervioso humano, son los denominados modelos conexionistas.

La neurona es una célula viva, que desde el punto de vista fisiológico está constituida por los mismos elementos, independientemente de su tamaño o forma. Si buscamos en un diccionario el concepto de "neurona" veremos que ésta se compone de una parte central denominada Soma o Cuerpo Celular, de la cual surge una rama principal denominada Axón, y otras ramas más cortas llamadas Dendritas.

El axón, se escinde en su extremo en millares de ramificaciones que alcanzan las dendritas de otras neuronas, a esta estructura de conexión que se da entre ellas se la conoce como sinapsis.

En las siguientes figuras podemos observar la estructura de una neurona biológicamente hablando y en la que se distingue fácilmente las tres partes principales que la conforman (Soma, Axón y Dendritas) y el momento en el cual se produce la sinapsis entre dos neuronas:

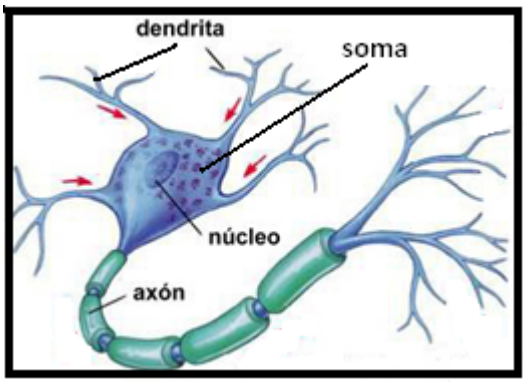


Figura: 3.1.1 – 1 Neurona biológica

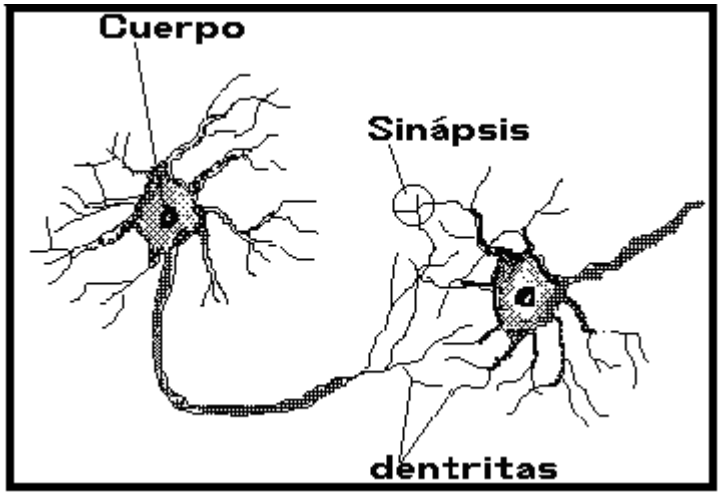


Figura 3.1.1 – 2: Sinapsis entre neuronas

Las dendritas actúan como receptores de las señales que proceden de neuronas adyacentes, mientras que es a través del axón por donde se transmite la actividad neuronal generada a otras células nerviosas, a órganos sensitivos o a las fibras musculares, y al revés.

En 1888, Santiago Ramón y Cajal (Premio Nóbel) estudió éste complejo mecanismo de comunicación entre neuronas y demostró por primera vez que los axones procedentes de las células nerviosas terminan libremente, relacionándose con las dendritas y con los cuerpos de otras neuronas por contacto y no por continuidad.

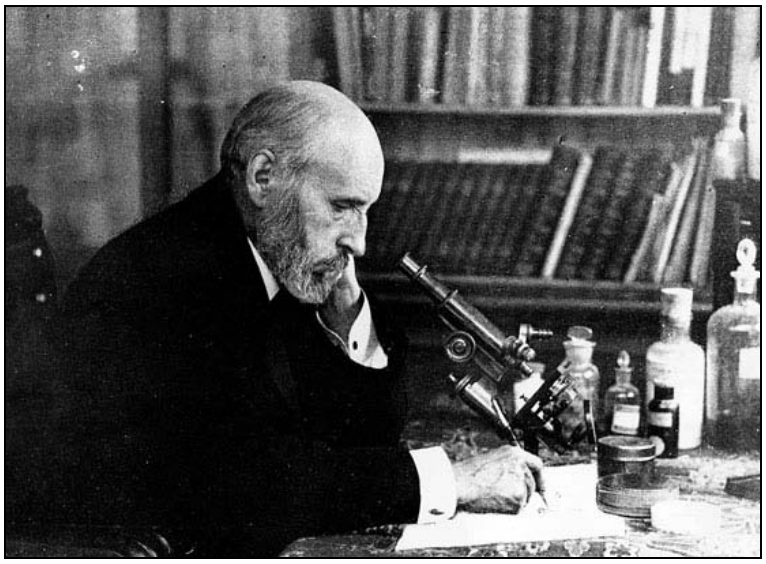


Figura 3.1.1 – 3: Santiago Ramón y Cajal

Las señales que son recibidas por una neuronas son combinadas y en función de la estimulación que esta combinación provoca, la neurona alcanzará un cierto nivel de activación que si llega a estimular a la misma, provocará su activación, lo que conlleva una serie de impulsos y a determinar a qué velocidad se transmiten estos impulsos a lo largo de la neurona.

Modificando como unas neuronas influyen sobre otras o lo que es lo mismo variando la efectividad de la sinapsis se produce el aprendizaje en las redes de neuronas biológicas, por eso se dice que el conocimiento implícito en la red (memoria) queda definido en las conexiones, es decir, en la sinapsis.

Las redes de neuronas artificiales (Reas) toman como modelo las neuronas biológicas, aunque por supuesto su modelización implica simplificar el sistema.

Igual que en el cerebro, el elemento fundamental de una RNA es la neurona y que queda definida como unidad del proceso.

Una RNA se encuentra formada por un conjunto de unidades altamente interconectadas, del mismo modo que el cerebro realiza una tarea particular. Estas unidades son implementadas mediante componentes electrónicos o simuladas en una computadora mediante un software.

3.1.2 – Evolución histórica

La inteligencia artificial es la rama de la informática dedicada al desarrollo de agentes racionales no vivos, donde agente puede ser considerado como cualquier cosa que es capaz de recibir entradas, procesarlas y proporcionar una salida, lo que en la vida real consideraríamos como cualquier ser vivo es capaz de recibir un estímulo, procesar la percepción y actuar en consecuencia.

Los primeros trabajos realizados sobre inteligencia artificial siguieron dos modelos o caminos bien diferenciados:

El modelo ***simbólico-deductivo***, que es el más conocido, se basa en el modelado de la racionalidad por medio de un sistema de reglas y manipulación simbólica.

El otro modelo es el denominado ***conexionista o inductivo***, *q*ue se basa en el desarrollo de modelos que se inspiran en la redes neuronales biológicas. Es verdad, que cada vez se va teniendo mayor conocimiento del comportamiento del cerebro, de su funcionamiento y su estructura física y lógica, pero existe un problema que limita la interpretación de los resultados bajo este modelo.

El problema radica en que los modelos conexionistas son implementados en computadoras que están basados en la arquitectura de Von Neumann que para su funcionamiento utilizan algoritmos o lo que es en definitiva una descripción secuencial del proceso de tratamiento de la información.

El comportamiento de los sistemas nerviosos biológicos es simulado a través de las redes de neuronas artificiales mediante la combinación de varios elementos simples de computación en sistemas que están fuertemente interconectados con el objetivo de que surja la inteligencia como resultado del aprendizaje del sistema. Por ello basándose en lo que se conoce actualmente de las neuronas y su comportamiento, se implementan redes neuronales artificiales que se pueden aproximar al funcionamiento real de una red de neuronas biológicas.

W.McCulloch y W.Pitts, 1943

En lo que se refiere a la evolución histórica de los estudios realizados sobre los modelos inspirados en las redes neuronales biológicas, diremos que sus comienzos datan de 1943, cuando W. McCulloch y W. Pitts [McCulloch, 43] propusieron una teoría general del procesado de información basada en redes de elementos binarios de decisión, que aunque en un estado todavía primitivo, podrían ser consideradas como neuronas.

D. Hebb, 1949

En 1949, D. Hebb publicó el libro *The organization of Behavior* [Hebb,49] introdujo dos ideas fundamentales, la de que una percepción o concepto se representa en el cerebro por un conjunto de neuronas que se activan a la vez y otro concepto, que ya nombramos anteriormente, el de que la memoria se localiza en las conexiones entre neuronas (sinapsis).

Hebb también proporcionó una regla de aprendizaje basada en el refuerzo de las conexiones físicas entre neuronas cuando éstas se activan y que es conocida como la regla de Hebb y en la que posteriormente se basaron varios de los algoritmos de aprendizaje.

V. Neumann, 1956

En 1956, Von Neumann describe su arquitectura de computadoras, con lo que en este punto se comienza a profundizar las grandes diferencias que se dan entre cerebros biológicos y la arquitectura clásica de las computadoras. [Neumann, 58]

F. Rossenblatt, 1960

En la década de los 60, F. Rossenblatt y sus colaboradores presentaron el concepto de Perceptrón [Rossenblatt, 58], que hace referencia a la percepción automática y que consiste en un modelo de naturaleza lineal que simplifica los mecanismos biológicos del procesamiento de la información sensorial. Este modelo tiene limitaciones para resolver problemas no lineales, lo que provocó que se fuera perdiendo paulatinamente el interés por las redes de neuronas.

En esta época y paralelamente al Perceptrón se presentaron varios modelos como el Adaptative Linear Elemente (Adeline) o el Multiple Adeline (Madeline) donde utilizaban la regla delta, conocida como la regla de aprendizaje por mínimos cuadrados.

M. Minsky y S.Papert, 1969

A finales de los años 60, M.Minsky y S.Papert [Minsky, 69], agravaron ésta pedida de interés por las redes de neuronas al criticar el Perceptrón de Rossenblatt demostrando que aún conectando un gran número de ellos, no era posible solucionar el problema de la linealidad.

D.E. Rumelhart, J.L. McClelland, G.E. Hinton y R.J. Williams, 1986

En la década de los 80 se produjo un resurgir del interés por el Perceptrón al aparecer el algoritmo BP (Algoritmo de aprendizaje de retropropagación), el cual es esencial en las redes de neuronas denominadas Perceptrón Multicapa y que se caracterizan por estar formadas por más de dos capas. [Rumelhart, 86]

El algoritmo BP permitió entrenar los modelos que se basaban en el Perceptrón Multicapa que solventaba las limitaciones del Perceptrón simple.

Además de la aparición del algoritmo de aprendizaje de retropropagación otras de las razones del resurgir del interés por el Perceptrón y por las RNAs fueron sin duda la aparición de ordenadores muchos más rápidos que los que existían en los años 50 y 60. Incluso los ordenadores de tipo serial ya permitían experimentar con modelos intrínsecamente paralelos. De hecho, gracias a los protocolos de comunicaciones locales y remotas, es posible distribuir los cálculos entre varios ordenadores seriales, consiguiendo así un entorno distribuido.

El modelo de computación en paralelo es de los más utilizados para lograr un mayor aumento de la velocidad en los cálculos científicos, pero a pesar de que existen muchos ordenadores paralelos e incluso con diversos tipos de arquitectura, no es sencillo desarrollar algoritmos que sean eficientes a la hora de construir un paradigma para la computación paralela en las redes de neuronas.

T. Kohonen, 1986

Simultáneamente al algoritmo BP, T. Coñeen [Kohonen, 82] propuso un mecanismo mediante el cual una red de neuronas puede organizarse por sí misma, de manera que las neuronas se disponen en una región determinada dependiendo de su actividad, lo que da lugar a lo que se conoce como mapas autoorganizados.

D.S. Bromead y D. Lowe, 1988

A finales de la década de los 80 apareció el concepto de redes de neuronas de base radial (RNBR) gracias a D.S. Bromead y D.Lowe [Bromead, 88]. Estas redes de neuronas surgieron como alternativa al Perceptrón Multicapa.

G.A Carpenter y S. Grossberg, 1990

A principio de los años 90, G.A. Carpenter y S. Grossberg establecieron la teoría de la resonancia adaptativa.

3.1.3 – Características de las redes de neuronas

Las características principales de una red de neuronas (RN) es que son sistemas distribuidos no lineales, sistemas tolerantes a fallos, adaptables, capacidad de establecimiento de relaciones no lineales entre datos y la posibilidad de implementación en VLSI.

Veamos resumidamente cada una de las características de una red de neuronas.

Las redes neuronales son ***sistemas distribuidos no lineales***, ya que puesto que una neurona es un elemento no lineal, una interconexión entre elementos no lineales tampoco lo será. Esta característica permite la simulación de sistemas lineales y caóticos y que no se podría realizar con los sistemas clásicos lineales.

Al ser un sistema distribuido, una red neuronal, permite el fallo de algunos elementos individuales, en nuestro caso de neuronas, sin alterar por ello la respuesta total del sistema. Que las redes neuronales sean ***sistemas tolerantes a fallos*** las hace bastante interesantes bajo el punto de vista de que actualmente los ordenadores generalmente son sistemas secuenciales, lo que significa que ante un fallo en una secuencia, el sistema global no funcione.

Las Redes neuronales pueden modificar loa parámetros de los que depende su funcionamiento dependiendo de los cambios que se puedan producir en el entorno de trabajo, por eso una de sus características es la ***adaptabilidad***. A pesar de esta capacidad de adaptación hay que tener en cuenta que tampoco puede ser muy grande ya que provocaría que el sistemas se volviese inestable.

Son capaces de ***establecer relaciones no lineales entre datos***.

Estos sistemas pueden ser aplicados en sistemas de tiempo real, pudiendo llegar a simular sistemas biológicos mediante elementos de silicio, gracias a la posibilidad de ***implementación en VLSI*** (Very Large Scale Integration)

3.2 – La neurona artificial

Dentro de una red de neuronas artificiales, la unidad elemental o el elemento mas básico de la misma es la propia neurona, a la que se denomina como un dispositivo simple de cálculo, que a partir de unas entradas que proceden del exterior o de otras neuronas, proporciona una respuesta o salida.

En la siguiente figura se pueden ver los elementos de los que está constituida una neurona artificial:

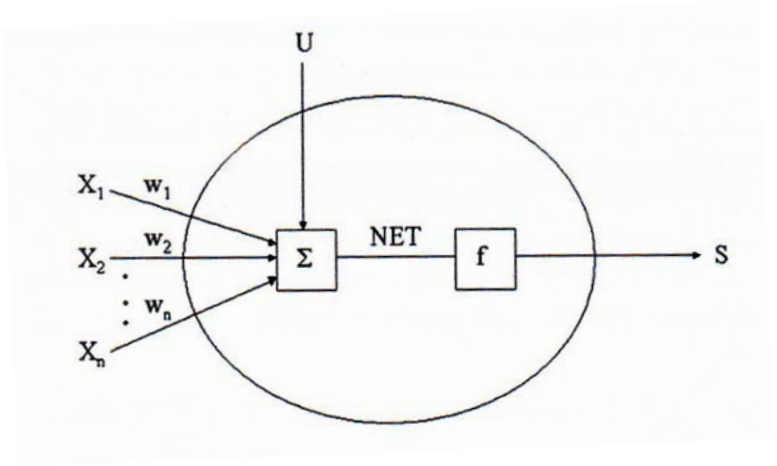


Figura 3.2-1: Neurona artificial

Donde:

- $X_1,\ldots, X_n$, son el conjunto de entradas.
- $w_1,\ldots,$ wn son los pesos sinápticos de la neurona, que representan la intensidad de interacción entre la entrada y la neurona.
- **S = f(NET)** es la regla de propagación, que proporciona el valor a la salida de la neurona, en función de sus pesos y entradas.
- f es la *función de activación* que proporciona el estado de activación actual de la neurona.

En función del tipo de salida, cada neurona normalmente suele recibir un nombre específico [Müller, 90]:

- Las neuronas estándar, son aquellas cuya salida solo puede tomar los valores 0 ó 1 y son conocidas con el nombre de ***neuronas de tipo McCulloch-Pitts.***
- Aquellas neuronas, cuyo valor de salida es -1 ó 1 se suelen denominar ***neuronas de tipo Ising***.

- Y las neuronas, que pueden generar diversos valores discretos de salida (-1, 0, +1, +3…) se las denomina ***neuronas de tipo Potts***.

Nota: En nuestro caso, las neuronas de la capa oculta toman valores reales en el intervalo [0,1] y las neuronas de salida pueden tomar cualquier valor real.

A partir de este momento nos referiremos durante todo el proyecto al modelo de neurona básico (McCulloch-Pitts).

La regla de propagación nos permite obtener, a partir de entradas y los pesos, el valor de salida de la neurona. La función más habitual es de tipo lineal y su base es la suma ponderada de las entradas con los pesos:

$$NET = X_1 * w_1 + X_2 * w_2 + \cdots + X_n * w_n = \sum_{i=1}^{n} X_i * w_i$$

Ecuación 3.2.1-1: Función de propagación lineal

La función de propagación lineal se puede interpretar como el producto escalar de los vectores de entrada y pesos. Las conexiones, mediante sus pesos asociados (w_i) alteran la efectividad con la que la señal es transmitida. Algunas conexiones dejan pasar la señal con facilidad, mientras otras no.

Existen otros tipos de funciones de propagación que se basan en distancias entre vectores, como es el caso de la distancia euclídea, la mas usada en redes realimentadas y en mapas de Kohonen, y reglas que no son de tipo lineal que dan lugar a un tipo de neuronas que se denominan ***neuronas sigma-pi,*** [Rumehart, 86].

El **Umbral U** es el término que utilizan normalmente las neuronas artificiales y que puede ser entendido como un peso asociado a una conexión entre un valor constante e igual a 1 y la neurona en cuestión.

Este umbral U se interpreta de manera que la neurona generará una respuesta si el valor de la suma de entradas por los pesos de cada conexión es mayor del umbral que se ha establecido. En este caso, el cálculo de las entradas de la neurona se expresa de la siguiente forma:

$$NET = X_1 * w_1 + X_2 * w_2 + \cdots + X_n * w_n + U = \sum_{i=1}^{n} X_i * w_i + U$$

Ecuación 3.2.1-2: Función de propagación lineal con umbral

La neurona artificial recibe todas las entradas, y emite una señal de salida si la entrada total supera el valor del umbral.

La salida o activación de la neurona para una determinada entrada X puede determinarse por:

$$f(x) = \begin{cases} 1, si \sum_{i=1}^{n} X_i . w_i \geq U \\ 0, si \sum_{i=1}^{n} X_i . w_i < U \end{cases}$$

Ecuación 3.2.1-3: Función de activación respecto al umbral

Este modelo se puede extender modificando la función de activación de la neurona, puesto que en la mayor parte de los modelos esta función es monótona, creciente y continua, como se puede ver habitualmente en las neuronas biológicas.

En la siguiente tabla podemos observar las funciones de activación más empleadas (***función sigmoidal*** y ***función tangente hiperbólica***):

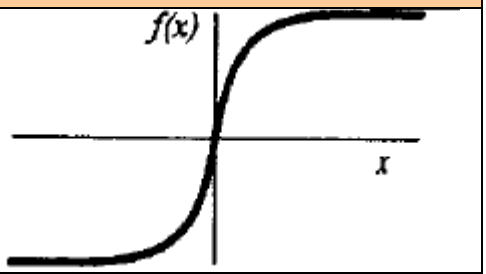
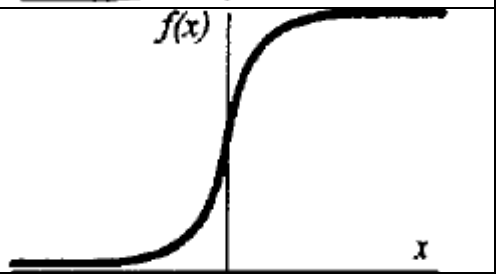
Función de activación	Función matemática	Rango	Gráfica
Sigmoidal	$f(x) = \frac{1}{1 + e^{-x}}$	**[0,+1]**	
Tangente Hiperbólica	$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$	**[-1,+1]**	

Tabla 3.2.1-1: Funciones de activación sigmoidales

Además de las dos funciones de activación anteriores, se pueden utilizar otras funciones de activación mas simples como la ***función identidad***, que se puede generalizar al caso de una función lineal cualquiera, la ***función escalón***, que se emplea en la red de Hopfield discreta y por el perceptrón simple o ***la función lineal a tramos*** que se puede considerar como una función lineal saturada en sus extremos, en los que se obtendría la máxima respuesta que se puede dar.

Otro tipo de función de activación es la ***función gaussiana***, que al igual que la función sigmoidal y la función tangente hiperbólica es la que se utiliza en la redes de neuronas de base radial. Observemos en la siguiente tabla estas cuatro nuevas funciones de activación:

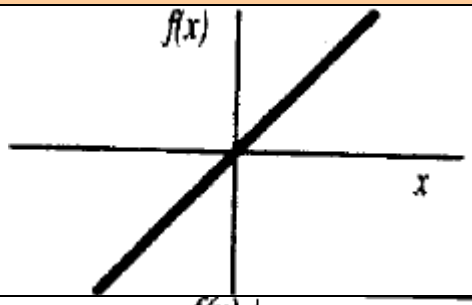
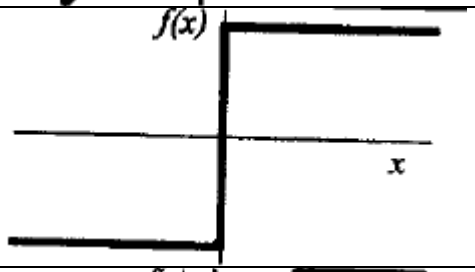
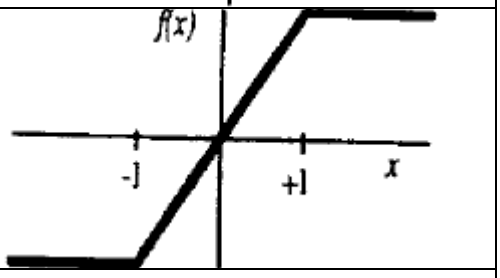
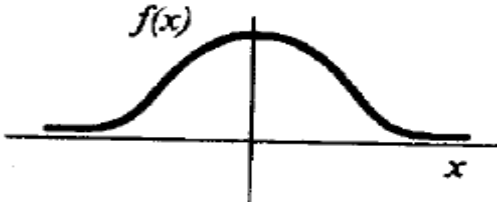
Función de activación	Función matemática	Rango	Gráfica
Identidad	$f(x) = x$	$[-\infty, +\infty]$	
Escalón	$f(x) = \begin{cases} f1/x > 0 \\ -f1/x \leq 0 \end{cases}$	**{-1,+1}**	
Lineal a tramos	$f(x) = \begin{cases} -1, & si\ x < l \\ x, si + l \leq x \leq -l \\ +1, si\ x > +l \end{cases}$	**[-1,+1]**	
Gaussiana	$f(x) = e^{\frac{-x^2}{2}}$	**[0,+1]**	

Tabla 3.2.1-2: Otras funciones de activación

3.3 – Redes de neuronas de base radial

Fue en 1989, cuando Moody y Darken [Moody, 89] [Poggio, 90] propusieron las redes de neuronas de base radial (RNBR).

Estas redes de neuronas son multicapas con conexiones hacia delante, pero a diferencia de otras, debido a que son simples, generales y rápidas en el aprendizaje son cada vez más utilizadas y tienen más aplicaciones.

3.3.1 – Características básicas

Las redes de neuronas de bases radial (RNBR's) son redes multicapa que construyen sus modelos con funciones de activación que son diferentes tanto en la capa oculta como la de salida.

El modelo clásico de las redes de neuronas de base radial está construido por una arquitectura rígida de **tres capas**: capa de entrada, capa oculta y capa de salida.

Las neuronas de la **capa de entrada** transmiten las señales de entrada a las neuronas ocultas sin realizar ningún tipo de procesamiento. Las señales son transmitidas mediante conexiones que no llevan pesos asociados.

La **capa oculta** de una RNBR es única y cada una de las neuronas de la capa se activa en una región distinta del espacio de entrada y realizan una transformación local de las señales que reciben de la capa de entrada.

Las neuronas de la **capa de salida** (Broomhead y Lowe, 1988) se implementan como una combinación lineal de las salidas proporcionadas por la capa oculta.

En el siguiente gráfico se puede observar cada una de las capas de una RNBR

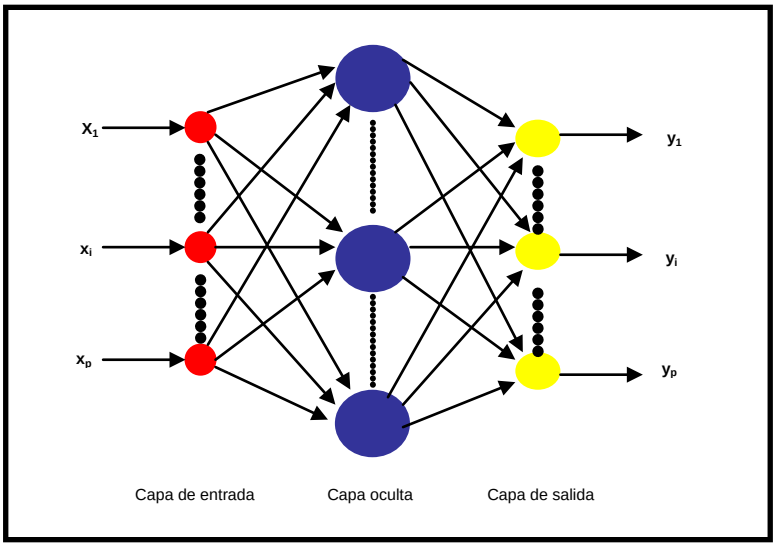


Figura 3.3.1 – 1: Estructura de una RNBR

Las funciones de base radial definen hiperesferas o hiperelipses que dividen el espacio de entrada.

Para llevar a cabo el entrenamiento de la red se debe completar dos fases, en la primera se determinarán las formas de las funciones de la capa oculta y en la segunda se pasan los pesos de la capa oculta a la capa de salida.

Las redes neuronales de base radial son consideradas como redes con una alta eficiencia en la fase de entrenamiento, a pesar de que no suelen ser usadas en aplicaciones que impliquen un volumen alto de patrones de entrenamiento.

Las RNBR son redes de aprendizaje rápido y son muy buenas en la aproximación de funciones.

3.3.2 – Funciones de base radial

Las funciones de base radial se caracterizan porque su respuesta se incrementa o decrementa monótonamente respecto a la distancia con su punto central o centroide, y a la vez los puntos que se encuentran a la misma distancia del centroide generarán salidas idénticas.

FUNCIONES DE BASE RADIAL		
Función	**Ecuación**	**Parámetros**
Gaussianas	**Ecuación 3.3.2-1** $\phi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right)$	*con parámetro de normalización δ>0*
Multi-Cuadrática	**Ecuación 3.3.2-2** $\phi(r) = (r^2 + \sigma^2)^{1/2}$	*con parámetro δ>0*
Multi-Cuadrática generalizada	**Ecuación 3.3.2-3** $\phi(r) = (r^2 + \sigma^2)^{\beta}$	*con parámetros δ>0 1 > β > 0*
Multi-Cuadrática inversa	**Ecuación 3.3.2-4** $\phi(r) = (r^2 + \sigma^2)^{-1/2}$	*con parámetro δ > 0*
Multi-Cuadrática inversa generalizada	**Ecuación 3.3.2-5** $\phi(r) = (r^2 + \sigma^2)^{-\beta}$	*con parámetros δ>0 1 > β > 0*
Cúbica	**Ecuación 3.3.2-6** $\phi(r) = r^3$	

Tabla 3.3.2 – 1: Funciones de base radial

En la tabla anterior (3.3.2-1) se pueden ver algunas de las funciones base que son más utilizadas.

Las funciones Gaussianas y Multi-cuadráticas tienen salidas significativamente distintas de cero solo para un rango acotado de valores alrededor del centro, por eso se dice que tienen forma local.

El factor de escala mide la anchura de la función Gaussiana y constituye una medida del radio de influencia de la función; está representado por el parámetro de normalización δ.

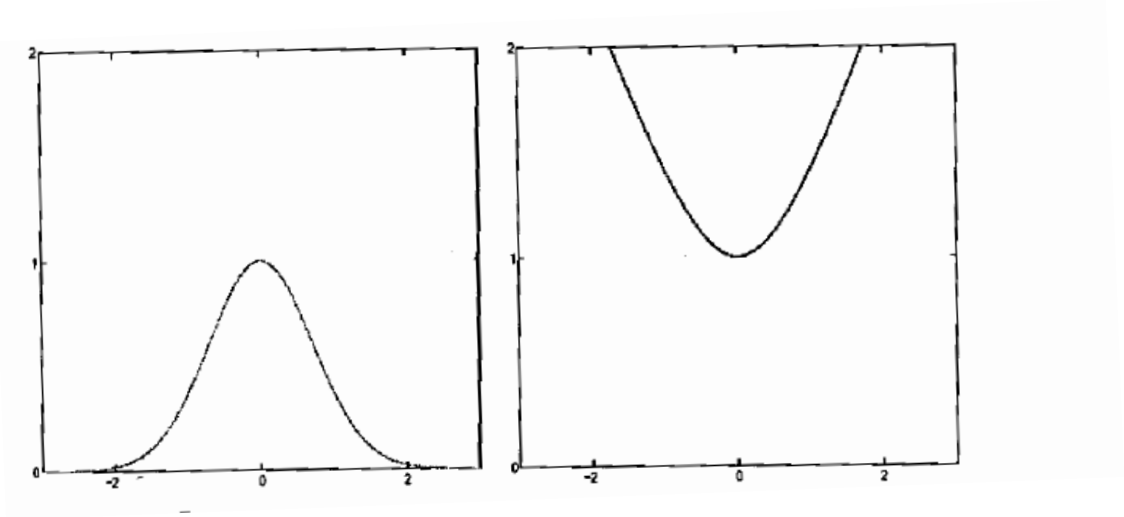


Figura 3.3.2 – 2: Funciones gaussiana y multicuadrática

3.3.3 – Arquitectura de las RNBR

Anteriormente, ya se expuso que la arquitectura de una red neuronal de base radial cuenta con tres capas de neuronas (de entrada, oculta, de salida).

Vamos a considerar los siguientes parámetros:

- n = número de neuronas en la capa de entrada (dimensión del vector de entrada)
- m = número de neuronas en la capa oculta
- c = número de neuronas de la capa de salida (dimensión del vector de salida).

Las redes neuronales de base radial operan en función a la distancia que separa el vector de entrada con el vector de centros (centroide), a esta distancia se le aplica una función radial gaussiana.

A diferencia de otros tipos de redes, las RNBR son de respuesta localizada lo que significa que solo responden con una intensidad apreciable cuando el vector de entradas presentado y el centroide de la neurona pertenecen a una zona próxima en el espacio de entradas.

El vector de entradas a la red neuronal va a ser:

$$\mathbf{x = (x_1,\ldots\ldots,x_i,\ldots..x_n)}$$

Se van a proporcionar Np vectores de entrada durante el periodo de aprendizaje de la red, a los que a partir de ahora vamos a llamar patrones de entrenamiento.

Los vectores de salida de la capa oculta van a ser:

$$\mathbf{y = (y_1,\ldots\ldots,y_j,\ldots,y_m)}$$

Y los vectores de salida serán:

$$\mathbf{o = (o_1,\ldots,o_k,\ldots..o_c)}$$

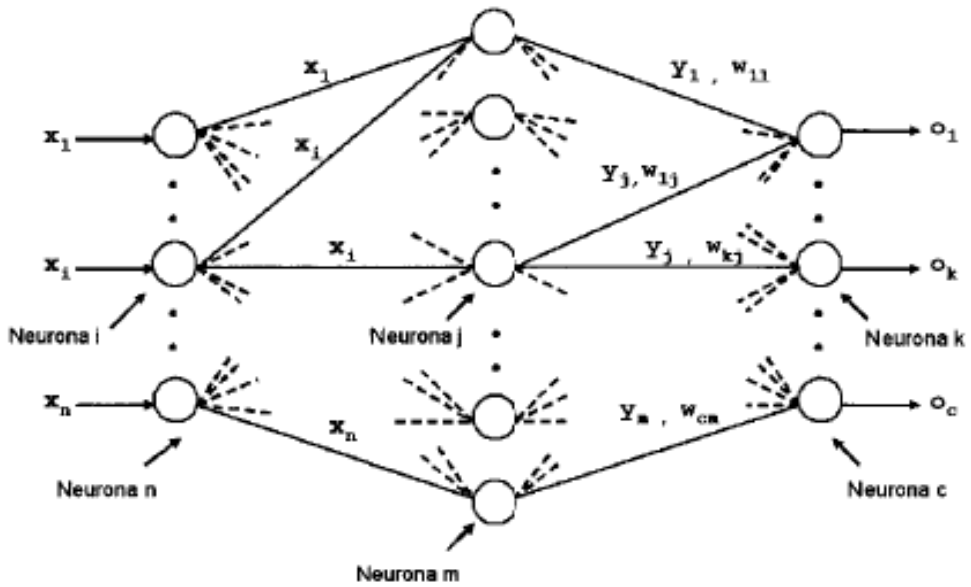


Figura 3.3.3-1: Resumen de notación

Cada una de las neuronas (j) de la capa oculta almacena un vector

$$\mathbf{c_j = (c_{j1}, \ldots, c_{jn})}$$

que van a ser las coordenadas del centroide en el espacio de entradas. Su dimensión coincidirá con la del vector de entradas x, y cada una de estas m neuronas calcula la distancia euclídea r_j que separa el vector de entradas x de su centroide c_j:

$$r_j^2 = \|x - c_j\|^2 = \sum_{i=1}^{n} (x_i - c_{ji})^2$$

Ecuación 3.3.3-1

La salida de la neurona j de la capa oculta, que es el vector y_j, se calcula a partir de la función de activación radial Gaussiana (Ecuación 3.3.2-1)y se puede expresar como:

$$y_j = e^{-r_j^2 / 2\sigma_j^2} = e^{-\sum_{i=1}^{n} (x_i - c_{ij})^2 / 2\sigma_j^2}$$

Ecuación 3.3.3-2

De esta forma, si el vector de entradas coincide con el centroide de la neurona j($x=c_j$). Ésta responde con la máxima salida (la unidad). Es decir, cuando el vector de entradas se sitúa en una región próxima al centroide la neurona, ésta se activa, indicando que reconoce el patrón de entrada; si el patrón de entrada es muy diferente del centroide, la respuesta tiende a cero como se puede ver en la siguiente figura:

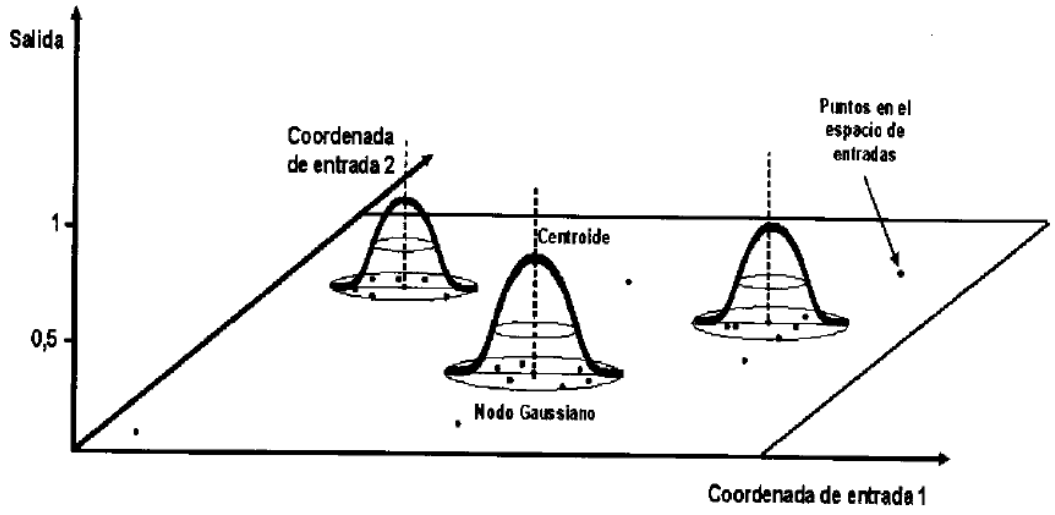


Figura 3.3.3-2: Respuesta local en las neuronas de una RNBR

El factor de escala σ_j, mide la anchura de la función gaussiana, que equivale al grado de influencia de la función implementada en la neurona dentro del espacio de entradas; cuanto mayor sea ese factor, la neurona abarcará un espacio más amplio en torno al centroide.

Las salidas de las neuronas ocultas son a su vez las entradas de las neuronas de salida, las cuales calculan su respuesta como una combinación lineal o_k de las salidas de

las neuronas de la capa oculta y_j, con los pesos de la conexión entre la neurona de la capa oculta j y la neurona k de la capa de salida: w_{kj}. Esta expresión tendrá la forma:

$$o_k = \sum_{j=1}^{m} w_{kj} y_j + U_k = \sum_{j=1}^{m} w_{kj} \phi(r_j) + U_k$$

Ecuación 3.3.3-3

Siendo

- w_{kj} , el peso que conecta la neurona de la capa oculta j con la salida k.
- U_k es el umbral de la neurona k.

Las RNBR basadas en funciones de activación local (como las gaussianas que hemos utilizado), se ha demostrado que constituyen un aproximadores universales, al igual que los perceptrónes multicapas.

3.3.4 – Aprendizaje de RNBR

En una red neuronal de base radial, el aprendizaje es un proceso que involucra la determinación, en primer lugar de la cantidad de centroides de la red y posteriormente de los parámetros libres de la red iterativamente. A continuación se expone la forma de cálculo del número de centroides y del valor de los parámetros:

- Cálculo del número de centroides

  Se trata de calcular el número de centroides de las funciones de activación de la capa oculta. En esta fase se busca que número de neuronas debe tener la red (en la capa oculta) para que pueda realizar un aprendizaje óptimo.

- Cálculo de los parámetros

  Una vez que ha sido calculado el número de centroides, habrá que calcular el valor de los mismos.

  Para cada una de las neuronas j de la capa oculta, tenemos:

  Vectores $c_j = (c_{j1},\ldots,c_{jn})$

  Los parámetros de escala de las funciones de activación para cada neurona de la capa oculta j:

  Números reales σ_j

Los pesos de la capa oculta a la capa de salida:

$$\text{Matriz } w = (w_{11},\ldots,w_{1m},w_{21},\ldots,w_{2m},\ldots w_{kj},\ldots w_{cl},\ldots w_{cm})$$

Para el cálculo de los parámetros disponemos de dos métodos de aprendizaje que veremos mas adelante: método totalmente supervisado y método híbrido.

La elección de uno de estos dos métodos hará que la red neuronal tenga una serie de características y comportamientos distintos.

Antes de entrar en detalles, primero se va a exponer las diferentes formas de calcular el parámetro denominado número de nodos gaussianos que tiene gran importancia y afecta muy significativamente al comportamiento de la RNBR.

3.3.4.1 – Elección del número de neuronas de la capa oculta

Antes de empezar a trabajar con una RNBR es importante elegir bien el número de nodos que van a estar en la capa oculta. Cada nodo gaussiano cubre una parte del espacio de entrada, de manera que para un espacio determinado que esté formado por los puntos de entrada, habrá que elegir un número suficiente de nodos para que el espacio quede significativamente cubierto.

Cuando se trabaja con espacios de entrada de muchas variables surge el problema, ya que el número de nodos necesarios para cubrirlo crece de manera exponencial al crecer la dimensión. Este fenómeno es conocido como *maldición de la dimensionalidad* [Bellman, 61]. Por lo tanto se debe llegar a un compromiso entre el número de nodos radiales seleccionado, el error en el ajuste de los patrones de aprendizaje y la capacidad de generalización.

La elección del número de neuronas se puede llevar a cabo a través de diferentes métodos.

Algoritmo autoorganizado jerarquizado: este algoritmo va eligiendo un número pequeño de neuronas y va comprobando si cada vector de entrada es activado. El algoritmo va aumentando el número de neuronas en cada iteración si fuese necesario [Lee, 92].

Análisis estadístico de clúster: mediante este análisis se obtienen las distintas agrupaciones o clusters de puntos, es decir los conjuntos de puntos que tienen características parecidas en ese espacio de entrada, a los que corresponderá una función radial que los cubra [Anderberg, 73].

Método de prueba y error: este método es el más utilizado y consiste simplemente en ir probando distintos números de nodos gaussianos y se evalúa la capacidad de la red hasta encontrar el más óptimo.

3.3.4.2 – Determinación de los parámetros de la RNBR

Como ya se ha expuesto al principio del capítulo 3.2.4, existen dos métodos de aprendizaje para el cálculo de los parámetros de la red:

Método totalmente supervisado: el uso de este método de aprendizaje supervisado en una red neuronal de base radial provoca que dicha red adquiera propiedades muy semejantes a otros tipos de red multicapas que emplean este grupo de algoritmos supervisados. Este tipo de redes constituye un aproximador universal de funciones ya que solo cambia la forma de afrontar el problema del aprendizaje.

En primer lugar denotaremos la salida que deseamos que proporcione la red (la mejor salida) con el vector:

$$t = (t_1,\ldots,t_c)$$

Sin embargo, para un determinado patrón de test x, la salida que obtenemos de la red será:

$$o = (o_1,\ldots,o_c)$$

Los parámetros de la red se determinan para minimizar el error cuadrático medio:

$$E = \frac{1}{N_p}\sum_{q=1}^{N_p} e_q$$

Ecuación 3.3.4.2-1

Con el error cuadrático medio para el vector de entrenamiento q-ésimo x_q:

$$e_q = \frac{1}{2}\sum_{i=1}^{c}\left(t_{qi} - o_{qi}\right)^2$$

Ecuación 3.3.4.2-2

Como estamos ante un problema de optimización no lineal, podemos utilizar el método del descenso del gradiente, que considera un gradiente o dirección de máximo decaimiento del error entre los distintos parámetros libres de la red, en nuestro caso pesos, centroides y parámetros de escala.

En cada iteración los parámetros se van actualizando y para poder verlo, se va a tomar como ejemplo el gradiente para los pesos (Ecuación 3.3.4.2-3) y después se mostrarán las expresiones finales para los centroides y para los factores de escala, cuya

obtención es análoga. La dirección del descenso del gradiente para los pesos w_{jk} entre la capa oculta y la de salida, viene dada por:

$$\Delta_q w_{kj} = -\frac{\delta e_q}{\delta w_{kj}}$$

Ecuación 3.3.4.2-3

Para obtener la expresión de actualización de los pesos, simplemente hay que derivar, teniendo presente las dependencias funcionales y aplicando adecuadamente la regla de la cadena:

$$w_{kj}^{(q)} = w_{kj}^{(q-1)} + \alpha_1 \Delta_q w_{kj}$$

Ecuación 3.3.4.2-4

siendo α_1 la razón de aprendizaje para los pesos.

Como ya hemos dicho, la forma de cálculo para los centroides c_j y para los factores de escala σ_j es análoga a la de los pesos:

- Para los centroides

$$c_j^{(q)} = c_j^{(q-1)} + \alpha_2 \Delta_q c_j$$

Ecuación 3.3.4.2-5

siendo

$$\Delta_q c_j = -\frac{\delta e_q}{\delta c_j}$$

Ecuación 3.3.4.2-6

α_2 la razón de aprendizaje para los centroides.

- Para los factores de escala

$$\sigma_j^{(q)} = \sigma_j^{(q-1)} + \alpha_3 \Delta_q \sigma_j$$

Ecuación 3.3.4.2-7

siendo

$$\Delta_q \sigma_j = -\frac{\delta e_q}{\delta \sigma_j}$$

Ecuación 3.3.4.2-8

α_3 la razón de aprendizaje para los parámetros de escala.

Las fórmulas para adaptar los parámetros de la red según la dirección del descenso del gradiente son:

Dado un parámetro cualquiera p, tenemos que...

$$\Delta_k p = -\sum_{j=1}^{m} (t_j - o_j) \frac{\partial o_j}{\partial p}$$

entonces:

- Para los pesos:

$$\frac{\delta o_k}{\delta w_{kj}} = f\left(\frac{\|x - c_j\|}{\sigma_j}\right)$$

Ecuación 3.3.4.2-9

- Para los centroides:

$$\frac{\delta o_k}{\delta c_{ji}} = w_{kj} f\left(\frac{\|x - c_j\|}{\sigma_j}\right) \cdot \frac{(x_i - c_i)}{2\sigma_j^2}$$

Ecuación 3.3.4.2-10

- Para los factores de escala:

$$\frac{\delta o_k}{\delta d_j} = w_{kj} f\left(\frac{\|x - c_j\|}{\sigma_j}\right) \cdot \left(\frac{(x_i - c_i)}{2\sigma_j^2}\right)^2$$

Ecuación 3.3.4.2-11

Las redes de base radial cuyo aprendizaje se lleva a cabo utilizando el método supervisado poseen, prácticamente las mismas características que cualquier red multicapas que utiliza aprendizaje supervisado, ya que las funciones de activación pueden llegar a tener carácter global.

Con este tipo de aprendizaje utilizado en la red neuronal de base radial se obtienen las siguientes características:

- Alta precisión en los resultados
- Se pierde el carácter local, pues el método no coloca restricciones: Un nuevo patrón de entrada puede producir actualizaciones a lo largo de toda la red.
- Problema de los mínimos locales, que ocurre con el uso de este tipo de algoritmos de optimización no lineal.
- Convergencia lenta, ya que es un problema de optimización lineal, y además requiere la actualización de muchos parámetros.

Método híbrido: este método consta de dos fases [Moody, 89][Lippmann, 89][Chen, 92],

Fase 1ª: Fase no supervisada

Para empezar con el aprendizaje necesitamos en primer lugar clasificar los patrones de entrada, siguiendo algún método de clasificación o agrupamiento (clustering). Estos métodos, entre otros, pueden ser:

- Método de las k-medias.
- Algoritmo de Kohonen.
- Métodos basados en el análisis cluster.

Ya que cada neurona va a representar a una función de base radial, el número de neuronas de la capa oculta va a ser el número de clases. Los parámetros de cada función radial serán de la siguiente forma:

- El centroide de la función de base radial es un elemento representativo de cada clase, normalmente el centro.
- La desviación o factor de escala de cada neurona se calcula utilizando alguna medida de distancia entre el centroide y los vecinos más próximos que constituyen una clase en sí.

Método de las k-medias

Este algoritmo, [Duda, 73], coloca los centros de las funciones de base radial en aquellas regiones donde existen grupos de datos significativos.

Sea m_1 el número de funciones de base radial o lo que es lo mismo el número de neuronas en la capa oculta, la determinación del valor de m1 debe basarse en la experimentación, mientras que el valor $\{t_k(n)\}^{m1}_{k=1,}$ representa los centros de las funciones de base radial en la iteración n del algoritmo.

Con esta notación, el algoritmo de las k-medias procede a actuar de la siguiente manera:

En primer lugar, está la ***fase de Inicialización*** en la cual se escogen valores aleatorios para los centroides iniciales $t_k(0)$; con la única restricción de que esos valores iniciales deben ser distintos entre sí. Estos centroides serán la disposición inicial de los representantes de las clases. Puede ser deseable que la distancia euclídea inicial entre esos centros sea pequeña, es decir que todos se encuentren en una misma región del espacio inicial.

En segundo lugar, la ***fase de Muestreo*** selecciona un vector de muestras x sobre el espacio de entradas con una cierta probabilidad. El vector x es una entrada en el algoritmo en la iteración n.

A continuación el método pasa a la ***fase de Comparación***, donde se busca el índice de similitud de los centros para el vector de entrada x, o lo que es lo mismo k(x), usando el criterio de mínima distancia euclídea en la iteración n.

$$K(x) = arg\ \min_k \|x(n) - t_k(n)\|$$,con k=1,2.... m_1

Ecuación 3.3.4.2-12

En la ***fase de Actualización***, se ajustan los centros de la función de base radial, usando la siguiente regla de actualización.

$$t_k(n+1) = \begin{cases} t_k(n) + n[x(n) - t_k(n)], & k = k(x) \\ t_k(n), & \end{cases}$$

Ecuación 3.3.4.2-13

Para terminar, el método de las k-medias, realiza una ***iteración*** incrementando n en 1 y volviendo a la fase de muestreo, continuando el proceso hasta que no haya cambios significativos en los centros t_k.

Por lo tanto la condición para que termine de iterar el método consiste en comprobar si el conjunto de centroides se ha desplazado, en caso negativo habrá terminado el método.

Algoritmo de Kohonen

Los mapas de Kohonen [Kohonen, 89] [Kohonen, 90] constituyen un modelo de redes de neuronas de dos capas que utiliza aprendizaje no supervisado autoorganizado y que se basa en redes competitivas.

Mediante este tipo de redes de neuronas se puede realizar una clasificación de los puntos que se encuentren más cercanos en el espacio de entrada.

Métodos basados en el análisis de cluster estadístico

Se denomina análisis de cluster [Anderberg, 73] [Hartigan, 75] a un grupo de técnicas de estadística multivariante cuya principal finalidad es agrupar objetos basándose en las características que tienen.

Mediante el análisis de cluster se puede clasificar datos de entrada de manera que cada uno de los datos sea muy parecido a los que se encuentran dentro de la agrupación.

Dicha agrupación toma su base respecto alguna medida de distancia como por ejemplo, la distancia euclídea o la distancia de Mahalanobis. Las agrupaciones de datos que resultan deben mostrar un grado de homogeneidad muy alto dentro de la misma agrupación, así como un alto grado de heterogeneidad con los que están fuera de la agrupación.

Una vez que se han obtenido los centroides, se debe proceder al cálculo de los parámetros de escala σ_j de cada una de las neuronas (normalmente se suele llevar a cabo a través de criterios heurísticos).

Un procedimiento se basa en calcular de forma aproximada el radio de influencia en el espacio de las entradas de cada neurona en relación a las demás, para lo cual se procede de la siguiente forma: para determinar el valor de σ_j de la neurona j, seleccionaremos los centroides de las n neuronas que estén más próximos al nodo j, y a continuación calcularemos el promedio de las distancias cuadráticas entre ellos.

Otro procedimiento consiste en el cálculo del promedio de la distancia de diversos patrones representativos al centroide [Hush, 93].

También se pueden elegir alguna medida de la cual se haya demostrado su buen comportamiento bajo experimentación, como puede ser la raíz cuadrada del producto de las distancias desde el centroide a sus dos vecinos más cercanos x_1 y x_2:

$$\sigma_j = \sqrt{\prod_{q=1}^{2} r_{qj}} = \sqrt{\prod_{q=1}^{2} \|x_q - c_j\|} = \sqrt{\prod_{q=1}^{2} \left[\sum_{i=1}^{n} (x_i - c_{ij})\right]}$$

Ecuación 3.3.4.2-14

Una vez que se han calculado los centroides y los factores de escala, termina el entrenamiento de las neuronas de la capa oculta, y con ello la fase no supervisada del aprendizaje.

Fase 2ª: Fase supervisada

Para terminar, se procede al entrenamiento de las neuronas de salida, haciendo notar que las cantidades $\varphi(r_j)$son valores numéricos conocidos, ya estimados, puesto que son función de los valores de entrada, centroides y factores de escala de las neuronas de

la capa oculta, que son valores ya calculados. Los pesos Wkj y umbrales Uk se calculan aplicando la regla LMS que se expone a continuación a partir de la expresión de la salida de la capa final:

$$o_k = \sum_{j-1}^{m} w_{kj} \emptyset \left(r_j\right) + U_k$$

Ecuación 3.3.4.2-15

La regla LMS, más conocida como mínimos cuadrados o regla de Widrow-Hoff constituye un algoritmo iterativo de optimización con actualizaciones continuas, siendo la actualización de los pesos proporcional al error que la neurona comete. Esta regla está limitada a la optimización de sistemas lineales, por lo que nuestro problema al ser la salida una combinación lineal se puede aplicar.

El método consistirá en proponer una función de error o coste que mida el rendimiento actual de la red, función que dependerá de los pesos sinápticos wj. Dada esta función de error, introduciremos un procedimiento general de optimización que sea capaz de proporcionar una configuración de pesos que correspondan a un mínimo de la función propuesta. El método aplicado a la función coste proporcionará una regla de actualización de pesos, que en función de los patrones de aprendizaje modifique iterativamente los pesos hasta alcanzar el punto óptimo de la red neuronal.

El método de optimización que es usado de manera más habitual es el que se denomina como método del descenso del gradiente. Para ello, se comienza definiendo una función de coste E(.) que proporcione el error actual E que comete la red neuronal, que será una función del conjunto de los pesos sinápticos E=E(W). En nuestro caso, esta función de coste vendrá dada por:

$$E\left[w_{kj}\right] = \frac{1}{2} \sum_{q=1}^{N_p} \sum_{i=1}^{n} (t_i^q - o_i^q)^2$$

Ecuación 3.3.4.2-16

donde

- t_k es la salida deseada
- o_k es la salida que se obtiene en la salida de la red

de esta manera podemos imaginarnos la representación gráfica de la función como una hipersuperficie con montañas y valles.

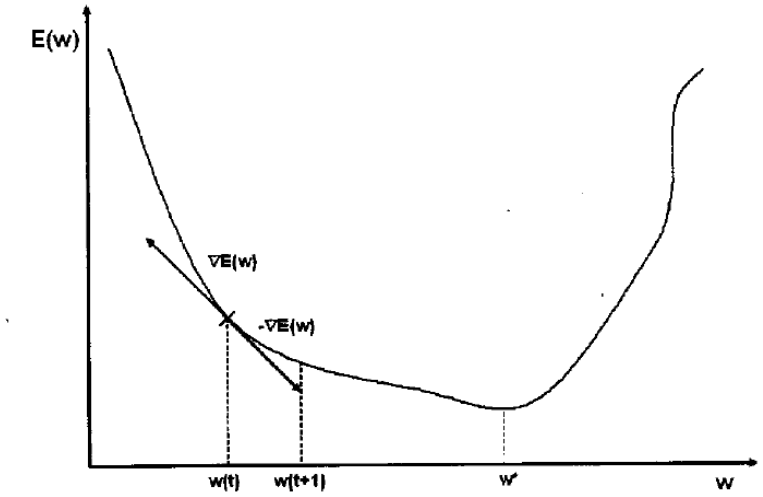


Figura 3.3.4.2-1: Descenso por el gradiente hacia un mínimo

Si observamos el gráfico anterior (3.3.4.2-1), la posición ocupada por un valle se corresponde con la configuración de pesos que corresponde a un mínimo local de la función de error w*. El objetivo del aprendizaje será encontrar siempre el óptimo global de la función de error, aunque dependiendo de los casos, deberemos conformarnos con un mínimo local lo suficientemente bueno.

Para encontrar la configuración de pesos óptima mediante el descenso del gradiente se opera del siguiente modo:

- Se parte de t=0 con una cierta configuración w(0) y se calcula el sentido de la máxima variación de la función E(w) en w(0), que vendrá dado por su gradiente en w(0).
- A continuación se modifican los parámetros w siguiendo el sentido contrario al indicado por el gradiente de la función de error. De este modo se desciende por la superficie de error hacia la zona de menos error.
- Finalmente se itera hasta alcanzar una zona en la que no haya un descenso significativo.

Matemáticamente hablando, si partimos de la expresión de error para el peso wjk se expresa de esta manera:

$$\frac{\delta E[w_{kj}]}{\delta w_{kj}} = \left(\frac{1}{2}\right).2\sum_{q=1}^{N_p}(t_k^q - o_k^q)\frac{do_k^q}{dw_{kj}} = -\sum_{q=1}^{N_p}(t_k^q - o_k^q)y_j^q$$

Ecuación 3.3.4.2-17

Y el incremento de pesos queda…

$$\Delta w_{kj} = -\varepsilon \frac{\delta E\left[w_{kj}\right]}{\delta w_{kj}} = \varepsilon \sum_{q=1}^{N_p} (t_k^q - o_k^q) y_j^q$$

Ecuación 3.3.4.2-18

Según esto, la expresión final de la iteración de los pesos para minimizar el error queda como:

$$w_{kj}(t+1) = w_{kj}(t) + \varepsilon(t_k - o_k)\varphi(y_j)$$

Ecuación 3.3.4.2-19

Esta expresión, como se puede ver, no es más que la regla de mínimos cuadrados.

El aprendizaje de una red neuronal de base radial mediante el método híbrido se caracteriza por lo siguiente:

- Ese generalmente muy rápido debido a que las neuronas poseen carácter local y a que los pesos aparecen de forma lineal con respecto a la salida pudiéndose utilizar el método de los mínimos cuadrados.
- Uno de los inconvenientes de las RNBR es la localización de los centros de las funciones de activación, sobre todo cuando la dimensión del espacio de entrada es muy alta.

3.4 – Utilización de distancias de Mahalanobis

Hasta este momento se ha visto la arquitectura y el método de aprendizaje mas generalizado que es empleado en las redes de neuronas de base radial (RNBR).

Se va a introducir ahora una variante importante, no se va a restringir a la utilización de un espacio euclídeo convencional y se va a utilizar un espacio euclídeo generalizado.

3.4.1 – Tipos de ponderación de la distancia Euclídea

En una RNBR, el aprendizaje depende de forma crítica de la función de distancias que se utilice.

Normalmente se utiliza la distancia euclídea para todos los atributos y patrones, pero existen múltiples formas de definir la función de distancias.

La función euclídea normal se puede extender para tener en cuenta factores de escala o ponderaciones, es decir, podemos dar más importancia a unos atributos sobre otros.

- **Distancia euclídea no ponderada**

La siguiente ecuación muestra el cálculo de la ***distancia euclídea no ponderada***, que es el tipo de distancia más usual y más conocida.

$$d_E(x,q) = \sqrt{\sum_{j=1}^{n}(x_j - q_j)^2} = \sqrt{(x-q)^T(x-q)}$$

Ecuación 3.4.1 -1: Distancia euclídea no ponderada

Uno de los inconvenientes que más se asocia a este tipo de cálculo de distancias es que depende en gran parte de las unidades de medida de las variables, problema que se evita en las redes de neuronas normalizando variables, por lo que es bastante adecuado utilizar este tipo de distancia.

- **Distancia euclídea diagonalmente ponderada**

El cálculo de la distancia euclídea diagonalmente ponderada se suele utilizar para dividir cada variable por un término que elimine el efecto de la escala. Para ello los valores en la diagonal m_j serán las desviaciones típicas de cada una de las variables.

$$d_m(x,q) = \sqrt{\sum_{j=1}^{n} \left(m_j(x_j - q_j)\right)^2} = \sqrt{(x-q)^T M^T M (x-q)} = d_E(Mx, Mq)$$

Ecuación 3.4.1-2: Distancia euclídea diagonalmente ponderada

donde m_j es el factor de escala para la dimensión número j y M es una matriz diagonal con $M_{jj} = m_j$

- **Distancia euclídea totalmente ponderada**

El cálculo de la distancia euclídea totalmente ponderada es conocida como distancia de Mahalanobis. En ella, M no va a ser una matriz diagonal como hemos visto en las distancias anteriores, ya que puede tomar valores arbitrarios.

$$d_M(x,p) = \sqrt{(x-q)^T M^T M (x-q)} = d_E(Mx, Mq)$$

Ecuación 3.4.1-3: Distancia euclídea totalmente ponderada

Función de ponderación euclídea no ponderada:

$$M = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Función de ponderación euclídea diagonalmente ponderada:

$$M = \begin{pmatrix} 1 & 0 \\ 0 & 3/4 \end{pmatrix}$$

Función de ponderación euclídea totalmente ponderada:

$$M = \begin{pmatrix} 1 & 3 \\ 1/2 & 4 \end{pmatrix}$$

En este proyecto, se tienen en cuenta las siguientes consideraciones:

- Cuando se trate de una distancia diagonalmente ponderada, la matriz M tendrá cualquier valor positivo o negativo en las posiciones de la diagonal de la matriz. Al realizar M^T M los elementos de la matriz solución serán necesariamente positivos. A este tipo de matriz haremos referencia a partir de ahora como el término matriz diagonal.

- Si es una distancia totalmente ponderada, la matriz M tendrá también cualquier valor ya sea positivo o negativo tanto en la diagonal como en el resto de la matriz. En cualquier caso la matriz deberá ser simétrica. A este tipo de matriz se hará referencia en adelante como matriz general o simétrica.

3.4.2 – Interpretación geométrica

Una matriz diagonal de función de distancias M (con un coeficiente por cada dimensión) puede construir una función simétrica de escala con una elipse paralela en el eje de ordenadas.

Si utilizamos la matriz $\begin{pmatrix} 1 & 0 \\ 0 & 1/2 \end{pmatrix}$ obtendremos esta elipse,

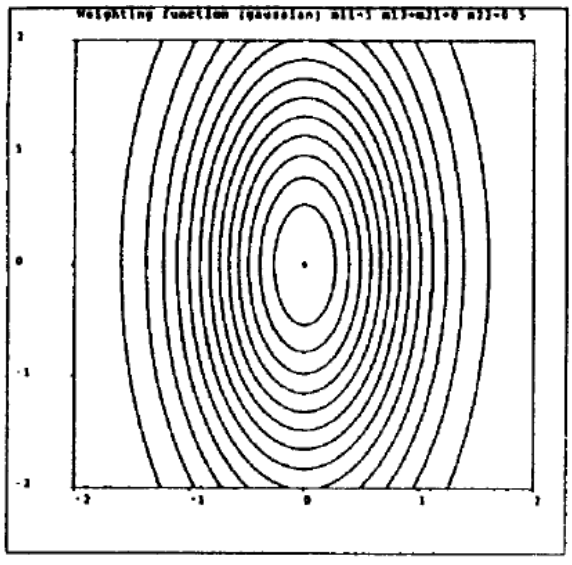


Figura 3.4.2-1: Mapa de nivel de una distancia constante desde un centro utilizando una matriz diagonal M

Mientras que si usamos una matriz de distancias con términos arbitrarios, se orientará la elipse [Ruppert, 94] [Wand, 93] como se ve en el siguiente gráfico. La matriz usada para este ejemplo es $\begin{pmatrix} 1 & 3/10 \\ 3/10 & 1/2 \end{pmatrix}$:

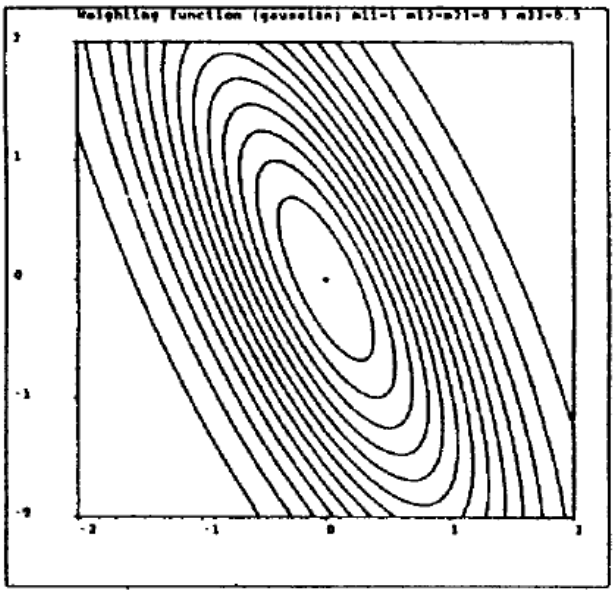


Figura 3.4.2-2: Mapa de nivel de una distancia constante desde un centro utilizando una matriz no diagonal M

En las redes de neuronas de base radial (RNBR) las neuronas tienen una función de activación normal radial o esférica, en la que se utiliza la distancia euclídea, sin embargo al cambiar de función, la activación de las neuronas se ve modificada siendo elíptica en vez de radial o simétrica.

El sistema del que disponemos en el proyecto está implementad para encontrar una matriz M adaptable a las particularidades de los atributos de entrada, mediante ponderación haciendo que la red consiga un mejor ajuste.

4. DESCRIPCIÓN DEL SISTEMA EMPLEADO

En este capítulo se explicará el funcionamiento del sistema que hemos empleado para realizar la experimentación.

Estudiaremos los requisitos funcionales del sistema, así como su estructura para terminar viendo de forma mas detallada los distintos ficheros que utiliza el sistema.

4.1 – Descripción general del sistema utilizado.

Antes de meternos de lleno en los requisitos funcionales del sistema, pondremos en antecedentes al lector explicando, de forma breve, como fue desarrollado el sistema que vamos a emplear.

El sistema sobre el que nos basamos para llevar a cabo la experimentación fue implementado por Oscar Fernández García en el año 2004. [García, 55]

Para aquellos lectores que quieran profundizar en el tema, en el anexo A de este proyecto encontrarán todo lo relacionado con la estructura del sistema empleado (arquitectura, diagrama de clases y funcionamiento).

El sistema que se utiliza fue desarrollado en C++ y en el cual se implementó el funcionamiento de una Red Neuronal de Base Radial (RNBR) con el método de aprendizaje híbrido estándar y un algoritmo genético (AG).

El sistema que se desarrolló cumple los siguientes requisitos:

Sistema batch que emplea el Algoritmo genético para buscar la matriz de Mahalonobis que se utiliza en una red neuronal de base radial con la función modificada de activación para poder usar este tipo de distancias.

El sistema es capaz de realizar el cálculo de la fitnees del Algoritmo genético a partir del error obtenido por la red neuronal de base radial y puede realizar el cálculo de la fitnees tanto a partir del error de entrenamiento como a partir del error de test de la red.

Proporciona la matriz de distancias a la RNBR que puede utilizar para su aprendizaje mediante un algoritmo genético.

Tiene la capacidad de leer ficheros de entrada desde el directorio principal donde se encuentra la aplicación e incluso con los nombres de ficheros que se le indiquen en la configuración, como veremos posteriormente al estudiar el fichero de configuración ***config.ini***.

Es capaz de generar los ficheros de salida con los que son posibles analizar el resultado obtenido.

Es valido tanto para la experimentación con problemas de aproximación como de clasificación.

Se puede configurar el tipo de matriz con el que se desea trabajar: Mahalanobis o diagonal.

Tiene la posibilidad de realizar experimentos empleando validación cruzada o no; y en el caso de emplearla se puede seleccionar el número de particiones. Además es configurable la proporción de patrones de entrenamiento y test que se le puede proporcionar a la RNBR cuando se realiza validación cruzada.

El Algoritmo genético implementado permite utilizar el método de selección del torneo, puede utilizar el operador genético cruce uno por uno o por dos puntos y es capaz de realizar elitismo con el mejor elemento de la población anterior.

4.2 – Ficheros del sistema

En este apartado daremos una descripción general de los ficheros que utiliza el sistema a la hora de llevar a cabo un experimento.

4.2.1 – Ficheros de entrada

4.2.1.1 - Fichero de configuración

El sistema tiene un solo fichero de configuración y al que se le ha llamado "Config.ini". Es un fichero de texto en el que se almacenan los parámetros de configuración del sistema. El requisito principal es el de que el fichero se encuentre en el mismo directorio que el ejecutable.

Vamos a tomar como ejemplo para explicar de forma más detallada los parámetros de configuración del fichero config.ini:

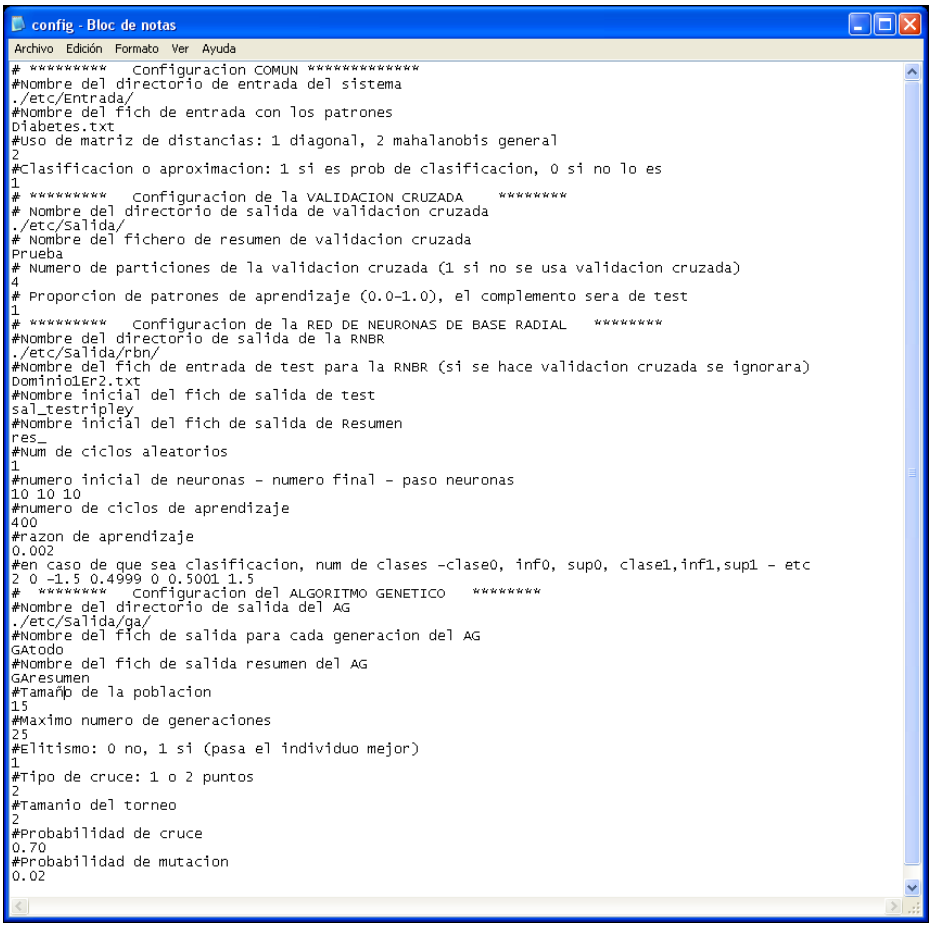


Figura 4.2.1.1 – 1: Fichero de configuración "config.ini"

Como podemos observar en la imagen anterior, el fichero de configuración está compuesto por cuatro secciones bien diferenciadas: parámetros de configuración común, parámetros de configuración de validación cruzada, parámetros de configuración de la RNBR y los parámetros de configuración del Algoritmo genético.

Parámetros de configuración común:

Nombre del directorio de entrada: Es el nombre del directorio donde se encuentran los ficheros de entrada como los dominios, además en este directorio se crearán los ficheros temporales. En nuestro ejemplo, el nombre del directorio de entrada es **./etc/entrada**

Nombre del fichero de entrada con los patrones: En este fichero se encuentran los patrones con los que se realizará el entrenamiento y la validación. En el caso de que se vaya a utilizar validación cruzada, el contenido de este fichero será distribuido en varios dependiendo de la utilización que se vaya a dar a cada uno de los patrones. En este caso el nombre del fichero de entrada es **diabetes.txt.** (Figuras 4.2.1.1-1,2 y 3)

Tipo de matriz de distancias: Dependiendo de si el valor es 1 ó 2, en la experimentación se utilizará una matriz diagonal o simétrica respectivamente. En el ejemplo se va a utilizar una matriz de Mahalonobis o lo que es lo mismo simétrica.

Clasificación o aproximación: En caso de que el valor de este parámetro sea 0 estaremos ante un problema de aproximación, sin embargo si el valor es 1 estaremos ante un problema de clasificación. En nuestro caso, estamos ante un problema de clasificación.

```
# **********   Configuracion COMUN ***************
#Nombre del directorio de entrada del sistema
./etc/Entrada/
#Nombre del fich de entrada con los patrones
Diabetes.txt
#Uso de matriz de distancias: 1 diagonal, 2 mahalanobis general
2
#Clasificacion o aproximacion: 1 si es prob de clasificacion, 0 si no lo es
1
```

Figura 4.2.1.1 – 2: Fragmento inicial del fichero de configuración para un problema de tipo clasificación con matrices de Mahalanobis.

```
# **********   Configuracion COMUN ***************
#Nombre del directorio de entrada del sistema
./etc/Entrada/
#Nombre del fich de entrada con los patrones
Diabetes.txt
#Uso de matriz de distancias: 1 diagonal, 2 mahalanobis general
2
#Clasificacion o aproximacion: 1 si es prob de clasificacion, 0 si no lo es
0
```

Figura 4.2.1.1 – 3: Fragmento inicial del fichero de configuración para un problema de tipo aproximación con matrices de Mahalanobis.

Parámetros de configuración de validación cruzada:

Nombre del directorio de salida para la validación cruzada: En nuestro caso el nombre del directorio es **./etc/salida.**

Nombre del fichero resumen para la validación cruzada: En el ejemplo el nombre del fichero es **Prueba**

Número de particiones: Dependiendo del número de particiones se dividirá el fichero de los patrones de entrada. En nuestro caso se van a utilizar cuatro particiones.

Proporción de patrones de aprendizaje: Esta proporción se utiliza a la hora de calcular la fitness con la RNBR para dividir los patrones de aprendizaje entre patrones de entrenamiento y patrones de validación de la RNBR. En nuestro ejemplo la proporción es 1, siendo sus posibles valores de 0 a 1.

```
# *********   Configuracion de la VALIDACION CRUZADA    *********
# Nombre del directorio de salida de validacion cruzada
./etc/Salida/
# Nombre del fichero de resumen de validacion cruzada
Prueba
# Numero de particiones de la validacion cruzada (1 si no se usa validacion cruzada)
4
# Proporcion de patrones de aprendizaje (0.0-1.0), el complemento sera de test
1
```

Figura 4.2.1.1 – 4: Fragmento del fichero de configuración que contiene los parámetros de configuración de la validación cruzada.

Parámetros de configuración de la RNBR:

Nombre del directorio de salida para la RNBR: En nuestro ejemplo, el nombre del directorio de salida para la red neuronal de base radial es ***.\etc\salida\rbn***

Nombre del fichero con los patrones de test para la RNBR: Este parámetro solo se tendrá en cuenta cuando no se emplee validación cruzada. En el caso de que se aplique validación cruzada, estos ficheros se generarán de manera automática.

Nombre inicial del fichero de salida de test para la RNBR: Como su nombre indica, este parámetro determina el nombre del fichero de salida de test para la red neuronal de base radial.

Nombre inicial del fichero de salida resumen para la RNBR: Al igual que el parámetro anterior, en este caso, indica el nombre del fichero de salida resumen para la red neuronal de base radial.

Número de ciclos aleatorios: En este proyecto siempre se ha utilizado para toda la experimentación un único ciclo aleatorio. El parámetro hace referencia al número de ciclos con diferentes inicializaciones aleatorias que se van a utilizar en el aprendizaje de la red neuronal de base radial.

Número inicial, final, y paso de neuronas: En el proyecto se ha utilizado el mismo número de neuronas iniciales como finales por lo que este parámetro es irrelevante en el mismo

Número de ciclos de aprendizaje: Indica el número de ciclos de aprendizaje que se van a utilizar para entrenar la red neuronal.

Razón de aprendizaje: Este parámetro informa de la razón de aprendizaje que se va a utilizar para actualizar los pesos en el entrenamiento de la red neuronal.

Número de clases: En caso de que el problema no sea de clasificación, éste parámetro se ignora. Cuando el problema es de clasificación indica el número de clases, a continuación se pone la etiqueta de cada clase y los límites inferior y superior de la activación de la red para la RNBR clasifique un patrón de test en cada clase a la hora de hacer la validación.

```
# **********   Configuracion de la RED DE NEURONAS DE BASE RADIAL   *********
#Nombre del directorio de salida de la RNBR
./etc/Salida/rbn/
#Nombre del fich de entrada de test para la RNBR (si se hace validacion cruzada se ignorara)
Dominio1Er2.txt
#Nombre inicial del fich de salida de test
sal_testripley
#Nombre inicial del fich de salida de Resumen
res_
#Num de ciclos aleatorios
1
#numero inicial de neuronas - numero final - paso neuronas
10 10 10
#numero de ciclos de aprendizaje
400
#razon de aprendizaje
0.002
#en caso de que sea clasificacion, num de clases -clase0, inf0, sup0, clase1,inf1,sup1 - etc
2 0 -1.5 0.4999 0 0.5001 1.5
```

Figura 4.2.1.1 – 5: Fragmento del fichero de configuración que contiene los parámetros de configuración de la RNBR.

Parámetros de configuración del algoritmo genético:

Nombre del directorio de salida del AG: Este parámetro indica el nombre del directorio de salida del Algoritmo genético. En el ejemplo el directorio tiene la ruta ***./etc/salida/ga***

Nombre del fichero de salida para cada generación del AG: Indica el nombre del fichero de salida para cada una de las generaciones del algoritmo genético. En este ejemplo el nombre del fichero es ***GAtodo***.

Nombre del fichero de salida de resumen del AG: Este parámetro indica el nombre del fichero de salida resumen del algoritmo genético, en el ejemplo que se muestra mas adelante el nombre del fichero resumen es ***GAresumen***.

Tamaño de la población: Indica el número de soluciones candidatas que serán generadas en cada una de las generaciones.

Número de generaciones: Número generaciones (poblaciones) que generará el algoritmo genético, por defecto, el sistema comienza en la generación 0.

Elitismo: En caso de que no haya elitismo su valor será 0 y en caso de que el mejor individuo de cada población pase automáticamente a la siguiente generación su valor será 1 ya que existirá elitismo.

Tipo de cruce: Este parámetro tomará el valor 1 en el caso de que se desee realizar un cruce unitario y un valor 2 cuando se quiera utilizar cruce por dos puntos.

Tamaño del torneo: Indica el número de soluciones candidatas entre las que se elegirá para aplicar selección genética.

Probabilidad de cruce: Indica la probabilidad de que se produzca el operador genético cruce.

Probabilidad de mutación: Este parámetro indica la probabilidad por bit de aplicar el operador genético mutación.

```
#  *********   Configuracion del ALGORITMO GENETICO   *********
#Nombre del directorio de salida del AG
./etc/Salida/ga/
#Nombre del fich de salida para cada generacion del AG
GAtodo
#Nombre del fich de salida resumen del AG
GAresumen
#Tamaño de la poblacion
15
#Maximo numero de generaciones
25
#Elitismo: 0 no, 1 si (pasa el individuo mejor)
1
#Tipo de cruce: 1 o 2 puntos
2
#Tamanio del torneo
2
#Probabilidad de cruce
0.70
#Probabilidad de mutacion
0.02
```

Figura 4.2.1.1 – 5: Fragmento del fichero de configuración que contiene los parámetros de configuración del algoritmo genético.

4.2.1.2. – Fichero con los patrones de entrenamiento

Este fichero contiene los patrones de entrenamiento de la red neuronal de base radial (RNBR). En el caso de que en la experimentación se use validación cruzada, a partir del fichero con los patrones de entrenamiento se generarán el resto de los ficheros temporales con los patrones para cada una de las particiones.

Para cada partición se van a crear los siguientes ficheros:

- **Fichero de aprendizaje**

  El fichero de aprendizaje contiene los datos con los que se entrenará la red, incluye tanto los patrones de test como los de entrenamiento.

- **Fichero de validación**

  El fichero de validación es la porción de patrones que han reservado en la partición para llevar a cabo la validación cruzada.

- **Fichero de entrenamiento**

  Se obtiene a partir del fichero de aprendizaje y contiene los patrones de entrenamiento de la red.

- **Fichero de test**

  Al igual que el fichero de entrenamiento, el fichero de test se obtiene a partir del fichero de aprendizaje y contiene los patrones de test de la red.

Todos los ficheros temporales son eliminados al finalizar la ejecución del sistema.

El nombre del fichero que contiene los parámetros de entrenamiento puede ser modificado en el fichero config.ini como se explicó en el apartado anterior.

4.2.1.3 – Fichero con los patrones de test

Este fichero contiene los patrones de test de la red neuronal de base radial (RNBR), por lo que en el caso de que se utilice validación cruzada, el fichero será ignorado, ya que en este caso los ficheros con los patrones de test de la red se generan de manera automática para cada una de las particiones de la validación cruzada a partir del fichero de entrenamiento.

Como en el caso anterior, el nombre del fichero de test puede ser modificado en el fichero de configuración.

4.2.2 – Ficheros salida

Durante la ejecución del sistema se manejan una gran cantidad de datos que son necesarios volcarlos a un fichero para posteriormente poder estudiar los resultados.

Al igual que en los ficheros de entrenamiento y de test, los nombres de los ficheros de salida pueden ser modificados en el fichero de configuración config.ini.

4.2.2.1 – Fichero resumen de validación cruzada

Este fichero no se genera en el caso de no utilizar en la experimentación validación cruzada (se indica en el fichero de configuración). Para cada partición, este fichero almacena los resultados que se han obtenido para cada una de ellas:

- Fitness obtenida utilizando la distancia euclídea.
- Fitness obtenida utilizando la mejor solución encontrada por el algoritmo genético.
- Los valores reales de la matriz que representan la mejor solución encontrada.

En el caso de que el problema sea de aproximación para calcular la fitness se utilizará el error del conjunto de patrones que se han reservado para llevar a cabo la validación cruzada; en caso de que el problema sea de clasificación se emplea la tasa de aciertos.

4.2.2.2 – Fichero resumen de cada generación

Independientemente de que se use validación cruzada o no, para cada una de las generaciones se creará un fichero que mostrará las estadísticas obtenidas en una generación determinada. Para cada una de las generaciones el fichero muestra:

- Fitness obtenida utilizando la distancia euclídea. (la misma para todas las generaciones).
- Fitness mínima de la generación actual.
- Fitness máxima de la generación actual.
- Fitness media de la generación actual.
- Sumatorio de todas las fitness de la generación actual.
- Número de cruces realizados.
- Número de mutaciones efectuadas.

Tanto el número de cruces como el de mutaciones realizadas se contabilizan como un acumulado de todas las generaciones que se llevan hasta el momento.

4.2.2.3 – Fichero con las soluciones candidatas de cada generación.

Para cada generación, se va a generar también un fichero de salida, en el que se muestran los datos de cada una de las soluciones generadas:

- Fitness obtenida por la RNBR con cada solución
- Valores reales de la matriz que representa cada solución candidata.

Una vez estudiado y analizado el sistema que se va a emplear, se procede a probarlo de manera empírica. Para ello, se han utilizado 5 dominios de experimentación sobre los cuales se realizarán una serie de experimentos para comparar su funcionamiento respecto a una red de neuronas de base radial (RNBR) clásica y para posteriormente, en base a los datos, comprobar que es lo que está haciendo el sistema al realizar el ajuste.

5.1.- DOMINIO DE CLASIFICACIÓN DE LA DIABETES

5.1.1.- Introducción

En el primer dominio de experimentación vamos a emplear datos procedentes del Instituto Nacional de Enfermedades Digestivas, Renales y Diabetes (Nacional Insititute of Diabetes and Kidney Diseases).

Este dominio ha sido utilizado para probar el funcionamiento de diversos sistemas de aprendizaje automático.

Basándonos en distintos parámetros biométricos de un paciente se diagnosticará si éste presenta signos de diabetes (diagnóstico positivo) o carece de ellos (diagnóstico negativo) de acuerdo con el criterio de la Organización Mundial de la Salud (OMS).

En nuestro caso la determinación del resultado de un diagnóstico va a venir dado por una variable binaria: 0 – diagnóstico negativo; 1 – diagnóstico positivo.

Nuestro dominio se compone de 768 instancias, de las cuales 500 se corresponden con pacientes que no presentan signos de diabetes y de 268 cuyo diagnóstico es positivo para la diabetes.

Para seleccionar dichas instancias, que forman parte de una base de datos más extensa, se han tenido en cuenta diversas restricciones: todos los pacientes son mujeres con una edad inferior a los 22 años y que viven cerca de Phoenix, Arizona, USA.

Las instancias están compuestas por 8 atributos numéricos que atienden a las siguientes parámetros biométricos:

- Número de embarazos de la paciente.
- Concentración de glucosa en plasma por cada 2 horas en un test oral de tolerancia a la glucosa.
- Presión sanguínea diastólica (mm. Hg.).
- Grosor de la piel que rodea el tríceps (mm.).
- Insulina en suero en 2 horas (mu. U/ml.).
- Índice de masa corporal (peso en Kg./ (altura en m.)^2).
- Función de linaje de diabetes.
- Edad (años).

El valor de estos atributos ha sido normalizado en el intervalo [0,1].

A estos atributos, en el dominio, se le añade el *atributo de clasificación* cuyo valor determina si el test es positivo para la diabetes (1) o negativo (0).

En la figura 5.1-1 podemos observar un fragmento del fichero que contiene las instancias del dominio de la diabetes y en el que podemos ver los valores que toman los diferentes atributos en las primeras ocho columnas y a continuación el valor de clasificación que determina el diagnóstico.

Figura 5.1.1-1: Fragmento del fichero de entrada del dominio de la Diabetes.

Cada una de las instancias del dominio de la diabetes tiene 8 dimensiones, una por cada uno de sus atributos (Figura 5.1.1-1). [*Recordemos que la última columna que aparece en la imagen corresponde al valor de clasificación (0 ó 1)]*.

5.1.2.- Experimentación previa

Antes de pasar a la experimentación utilizando el sistema a evaluar deberemos realizar una serie de pruebas previas para determinar cuáles son los parámetros que serán empleados para la RNBR.

Éstas pruebas se van a realizar con 576 patrones de aprendizaje y 192 de test, que se corresponden con el 75% y el 25% respectivamente del número de instancias del dominio que como vimos en el apartado anterior era de 768.

Para el dominio de la diabetes se han fijado los siguientes parámetros en función de los resultados de esta experimentación previa:

- Número de ciclos de aprendizaje : 400
- Razón de aprendizaje : 0.002

La elección del número de neuronas vendrá determinada en función del error de entrenamiento y el error de test logrado por la red con distinto número de neuronas.

En la siguiente figura *(5.1.2-1)* se muestra el fichero de configuración *config.net* que contiene los parámetros correctos previos a la ejecución de las pruebas con RNBR y a continuación se muestra una tabla *(Tabla 5.1.2-1)* con los resultados numéricos de las pruebas:

```
#Fich de configuración
#num de dimensiones
8
#Nombre del fich de entrada de entrenamiento
traindiabetes.txt
#Nombre del fich de entrada de test
testdiabetes.txt
#Nombre inicial del fich de salida de test
test
#Nombre inicial del fich de salida de Resumen
Resumen
#Nombre del directorio de entrada
./entrada/
#Nombre del directorio de salida
./salida/
#Num de ciclos aleatorios
1
#numero inicial de neuronas - numero final- paso neuron
5 100 5
#numero de ciclos de aprendizaje
400
#razón de aprendizaje
0.02
#Clasificacion o aproximacion: 1 si es prob de clasificacion, 0 si no lo es
1
#en caso de que sea clasificacion, num de clases -clase0, inf0, sup0, clase1,inf1,sup1 - etc
2 0 -0.5 0.45 1 0.55 1.5
```

Figura 5.1.2-1: Fichero config.net para el dominio de diabetes.

Número de neuronas	Error de entrenamiento	Error de test	Tasa de aciertos
5	0,091269	0,350949	0,65285
10	0,082571	0,336297	0,68912
15	0,082516	0,324685	0,6943
20	0,079922	0,320240	0,70466
25	0,077595	0,308577	0,74093
30	0,075261	0,299173	0,73575
35	0,074888	0,301851	0,73057
40	0,075919	0,308414	0,74611
45	0,076177	0,301883	0,74611
50	0,075382	0,298113	0,74611
55	0,073142	0,298620	0,7513
60	0,071575	0,296846	0,73057
65	0,073404	0,297390	0,74093
70	0,073689	0,298702	0,7513
75	0,076493	0,303755	0,76166
80	0,074924	0,294738	0,74611
85	0,077967	0,306402	0,74093
90	0,076410	0,301333	0,7513
95	0,083293	0,317498	0,72021
100	0,077423	0,302708	0,74093

Tabla 5.1.2-1: Evolución del error de entrenamiento y de la tasa de aciertos obtenida con RNBR en función del número de neuronas.

En las gráficas que se muestran a continuación *(ver figura 5.1.2-2 y figura 5.1.2-3)* se encuentran representados el error de entrenamiento y la tasa de aciertos en función del número de neuronas.

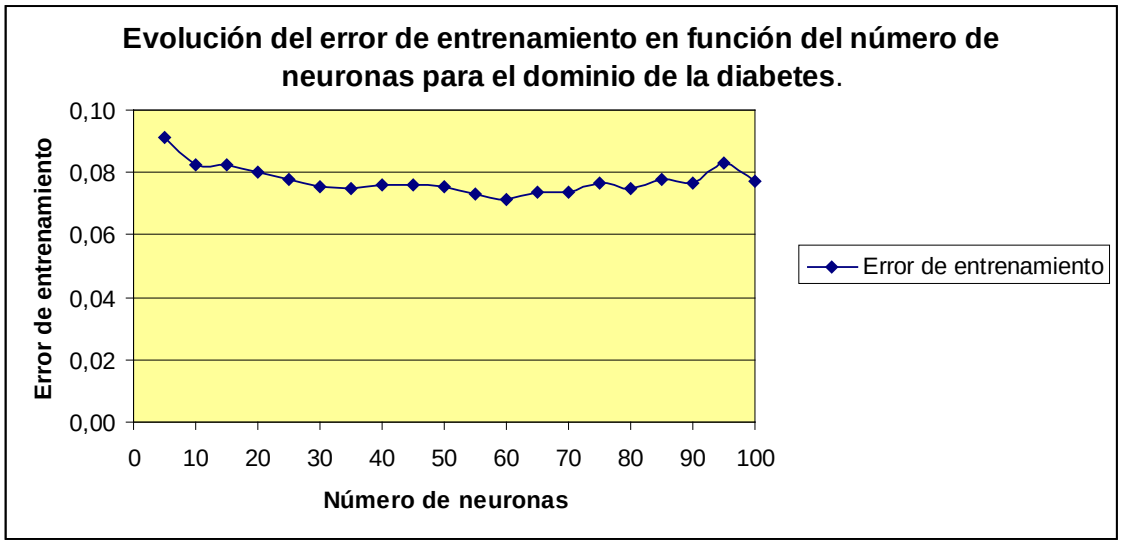


Figura 5.1.2-2

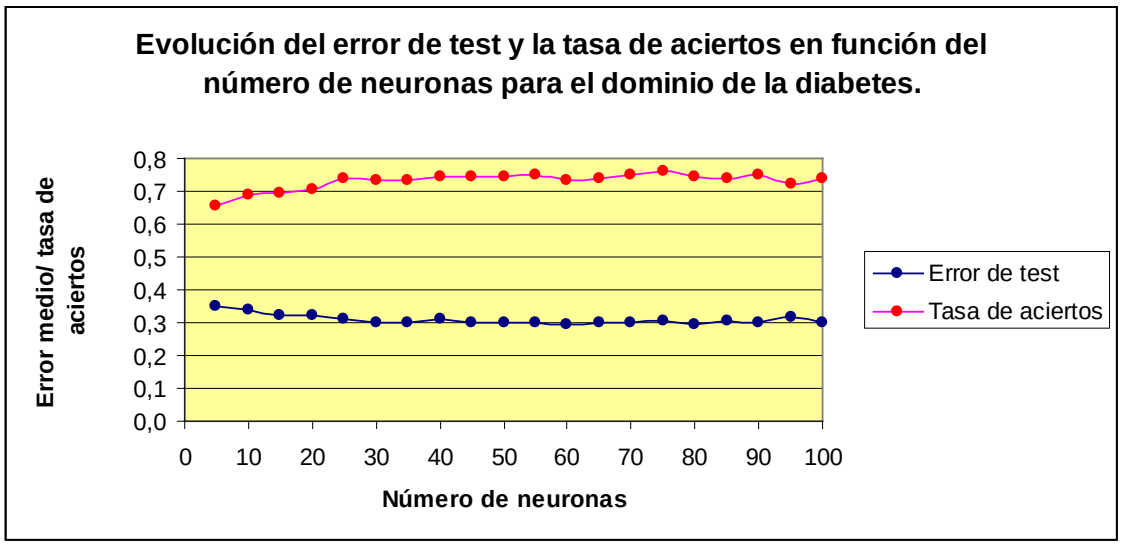


Figura 5.1.2-3

Si estudiamos ambos gráficos podemos determinar que la tasa de aciertos en test queda estabilizada a partir de 25 neuronas, por lo que utilizaremos dicho número para realizar la experimentación con este dominio.

5.1.3.- Experimentación con matrices de distancias diagonales.

Para la primera prueba de experimentación sobre el dominio de la diabetes sólo vamos a permitir que se obtenga como solución una matriz de distancias diagonal.

En posteriores capítulos realizaremos pruebas experimentales pudiendo obtener como solución una matriz simétrica.

Tanto en soluciones con matrices diagonales como simétricas, realizaremos experimentos calculando la fitness a partir del error de entrenamiento de la red.

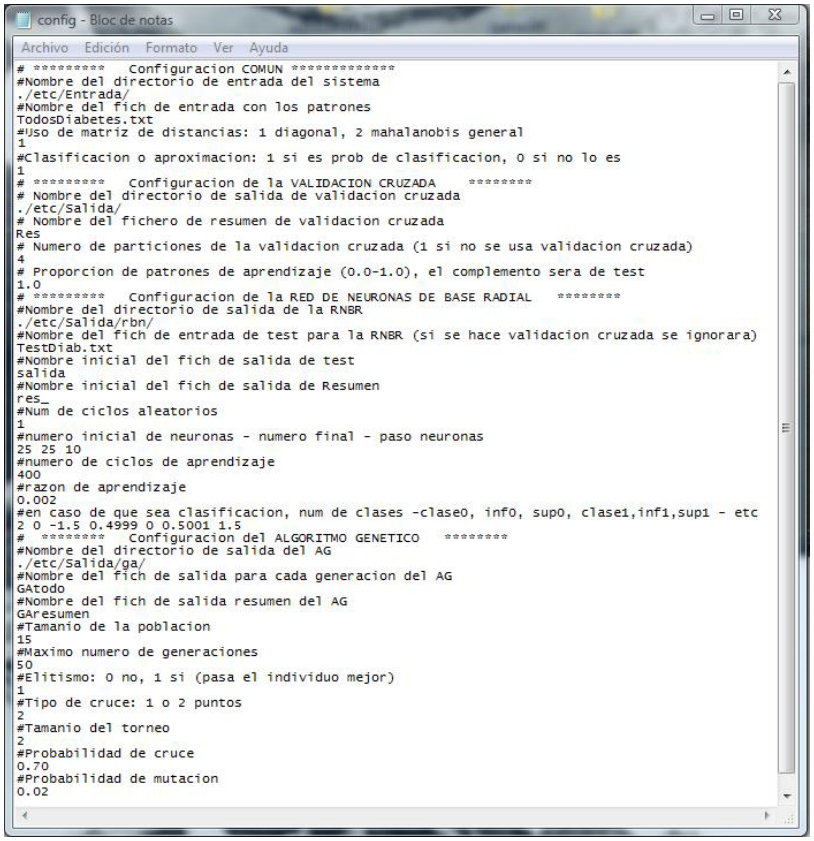
Con la matriz de distancias obtenida se calculará la tasa de aciertos en test. Este proceso lo repetiremos cuatro veces puesto que utilizaremos un esquema de validación cruzada con cuatro particiones.

La fitness que se va a emplear para hacer evolucionar el algoritmo genético (AG) se obtiene a partir del error de entrenamiento de la red. Una vez que termina de entrenarse se aplica la transformación logarítmica de ese error para obtener la fitness.

Recordemos que disponemos de 768 patrones, los cuales debido al uso de validación cruzada vamos a dividir en 4 particiones. Por lo tanto vamos a tener 192 patrones en cada partición y 576 patrones en la parte de aprendizaje para proceder a la validación de cada una de las particiones.

Los parámetros del sistema son los siguientes:

- La solución es una matriz de distancias diagonal
- Se emplea validación cruzada: 4 particiones.
- Se utilizan el 100% de los patrones del fichero de aprendizaje de cada partición para el entrenamiento de la red, a partir de los cuales se calcula la fitness.
- Número de generaciones: 50
- Tamaño de la población: 15
- Tamaño del torneo: 2
- Elitismo de 1 elemento
- Cruce de dos puntos
- Probabilidad de cruce: 0,7
- Probabilidad de mutación: 0,01
- Conversión binario a real con una precisión de 2 dígitos en la parte entera binaria y 3 en la parte fraccionaria(LongVal=6; Lpunto=3).



config - Bloc de notas

Archivo Edición Formato Ver Ayuda

```
# *********   Configuracion COMUN *************
#Nombre del directorio de entrada del sistema
./etc/Entrada/
#Nombre del fich de entrada con los patrones
TodosDiabetes.txt
#Uso de matriz de distancias: 1 diagonal, 2 mahalanobis general
1
#Clasificacion o aproximacion: 1 si es prob de clasificacion, 0 si no lo es
1
# *********   Configuracion de la VALIDACION CRUZADA    ********
# Nombre del directorio de salida de validacion cruzada
./etc/Salida/
# Nombre del fichero de resumen de validacion cruzada
Res
# Numero de particiones de la validacion cruzada (1 si no se usa validacion cruzada)
4
# Proporcion de patrones de aprendizaje (0.0-1.0), el complemento sera de test
1.0
# *********   Configuracion de la RED DE NEURONAS DE BASE RADIAL   ********
#Nombre del directorio de salida de la RNBR
./etc/Salida/rbn/
#Nombre del fich de entrada de test para la RNBR (si se hace validacion cruzada se ignorara)
TestDiab.txt
#Nombre inicial del fich de salida de test
salida
#Nombre inicial del fich de salida de Resumen
res_
#Num de ciclos aleatorios
1
#numero inicial de neuronas - numero final - paso neuronas
25 25 10
#numero de ciclos de aprendizaje
400
#razon de aprendizaje
0.002
#en caso de que sea clasificacion, num de clases -clase0, inf0, sup0, clase1,inf1,sup1 - etc
2 0 -1.5 0.4999 0 0.5001 1.5
#  ********   Configuracion del ALGORITMO GENETICO   ********
#Nombre del directorio de salida del AG
./etc/Salida/ga/
#Nombre del fich de salida para cada generacion del AG
GAtodo
#Nombre del fich de salida resumen del AG
GAresumen
#Tamanio de la poblacion
15
#Maximo numero de generaciones
50
#Elitismo: 0 no, 1 si (pasa el individuo mejor)
1
#Tipo de cruce: 1 o 2 puntos
2
#Tamanio del torneo
2
#Probabilidad de cruce
0.70
#Probabilidad de mutacion
0.02
```

Figura 5.1.3-1: Fichero de configuración utilizado para la experimentación con matrices diagonales en el dominio de clasificación de la diabetes.

Una vez introducidos los parámetros en la aplicación *(figura 5.1.3-1)* y ejecutarla, procedemos a estudiar los resultados obtenidos:

Partición	Fitness con RNBR	Fitness con el sistema	Matriz solución obtenida
1	0,7561	0,7479	0,938 0 0 0 0 0 0 0 0 1,875 0 0 0 0 0 0 0 0 -0,875 0 0 0 0 0 0 0 0 0,438 0 0 0 0 0 0 0 0 1,062 0 0 0 0 0 0 0 0 2,125 0 0 0 0 0 0 0 0 1,562 0 0 0 0 0 0 0 0 -3,750
2	0,7357	0,7195	-1,062 0 0 0 0 0 0 0 0 3,867 0 0 0 0 0 0 0 0 -0,187 0 0 0 0 0 0 0 0 -0,562 0 0 0 0 0 0 0 0 0,312 0 0 0 0 0 0 0 0 -1,875 0 0 0 0 0 0 0 0 -1,125 0 0 0 0 0 0 0 0 -1,375
3	0,8008	0,8008	3,812 0 0 0 0 0 0 0 0 0,812 0 0 0 0 0 0 0 0 -0,875 0 0 0 0 0 0 0 0 -1,375 0 0 0 0 0 0 0 0 -0,187 0 0 0 0 0 0 0 0 1,375 0 0 0 0 0 0 0 0 1,437 0 0 0 0 0 0 0 0 1,875
4	0,9796	0,9756	-0,937 0 0 0 0 0 0 0 0 -2,250 0 0 0 0 0 0 0 0 -0,562 0 0 0 0 0 0 0 0 -0,562 0 0 0 0 0 0 0 0 -1,250 0 0 0 0 0 0 0 0 1,437 0 0 0 0 0 0 0 0 -1,062 0 0 0 0 0 0 0 0 1,437

Tabla 5.1.3-1: Resultados obtenidos para cada una de las particiones de validación cruzada restringiendo el resultado a matriz diagonal calculada a partir de la tasa de aciertos.

En el gráfico *(figura 5.1.3-2)* podemos observar representada la tasa de acierto para cada una de las particiones. Las últimas columnas de la gráfica se corresponden con la tasa de acierto media:

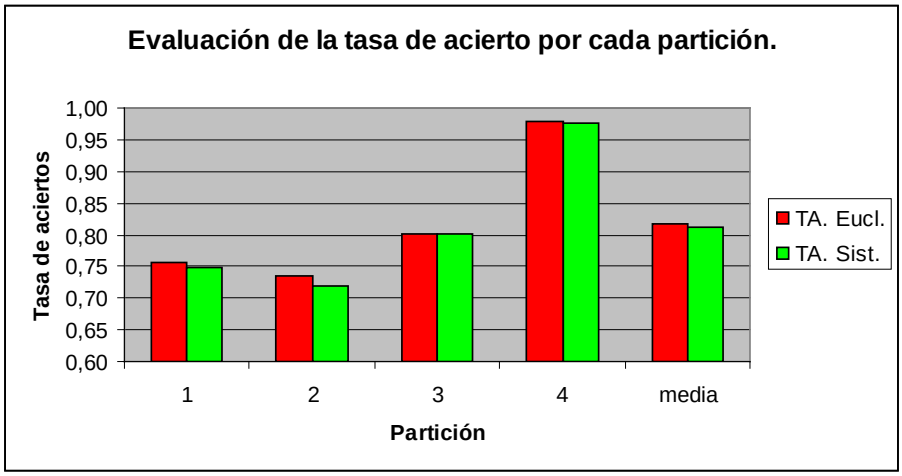


Figura 5.1.3-2: Evaluación de las particiones. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

Dada la tasa de aciertos media obtenida en el sistema *(0,8181)* y en la RNBR *(0,8110)* se puede observar como la tasa de aciertos obtenida es ligeramente inferior en nuestro sistema. A pesar de que en la tercera partición, ambas tasas se igualan *(0,8008)*, en el resto de particiones el número de aciertos de la red neuronal es ligeramente superior a los obtenidos por el sistema. (0,7561 de la red por 0,7479 del sistema en la primera partición, 0,7357 por 0,7195 en la segunda partición y 0,9796 por 0,9756 en la cuarta partición).

Como conclusión se puede decir que para el dominio de la diabetes, utilizando matrices diagonales como solución y calculada la fitness a partir del error de entrenamiento, a pesar de que el sistema no obtiene mejores resultados que la Red Neuronal de Base Radial, prácticamente los iguala, superando el 80% de aciertos.

En las siguientes gráficas podemos observar la evolución de la fitness máxima, mínima y media de la población para cada una de las particiones:

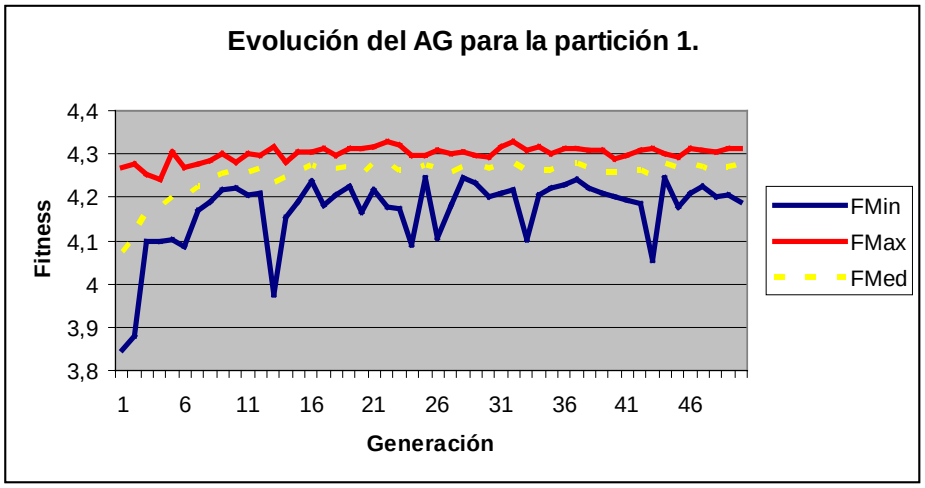


Figura 5.1.3-3: Evolución del AG para la partición 1. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

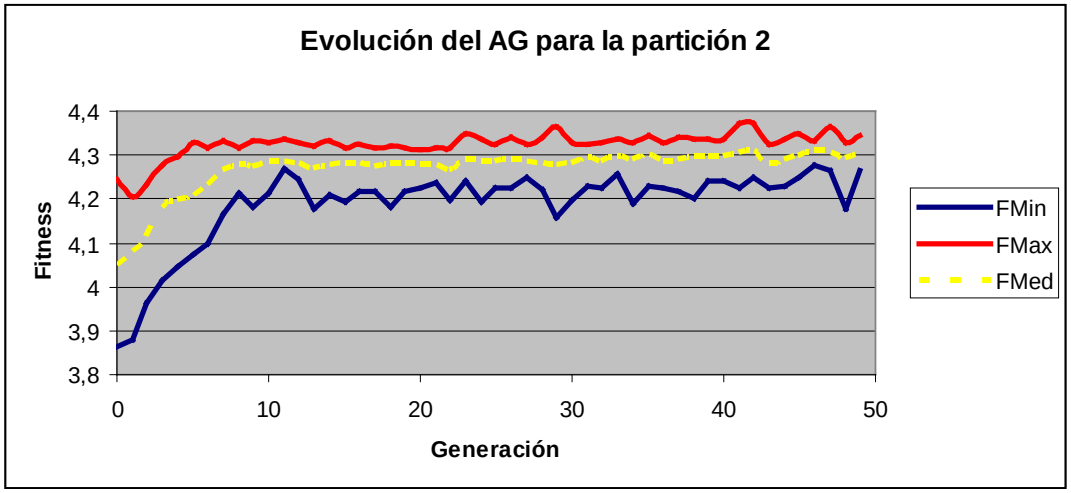


Figura 5.1.3-4: Evolución del AG para la partición 2. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

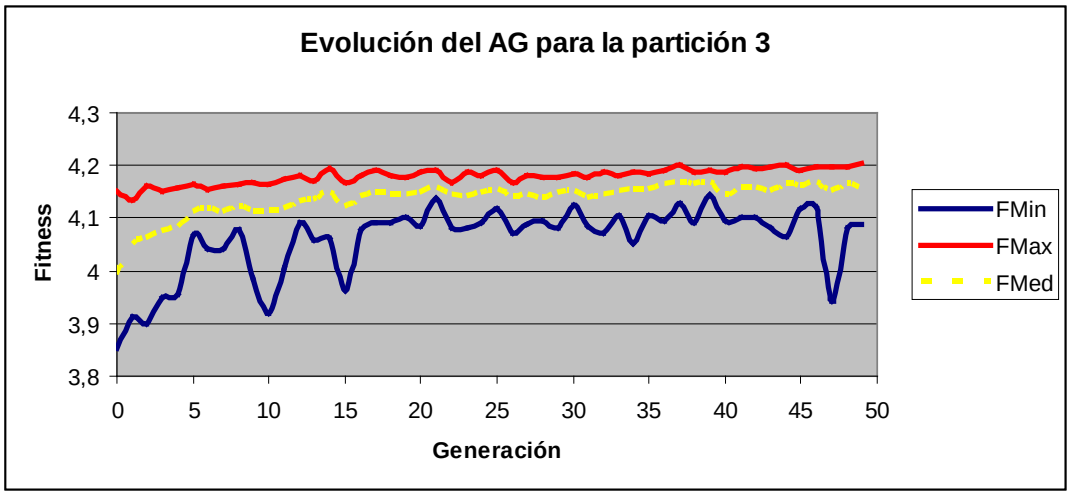


Figura 5.1.3-5: Evolución del AG para la partición 3. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

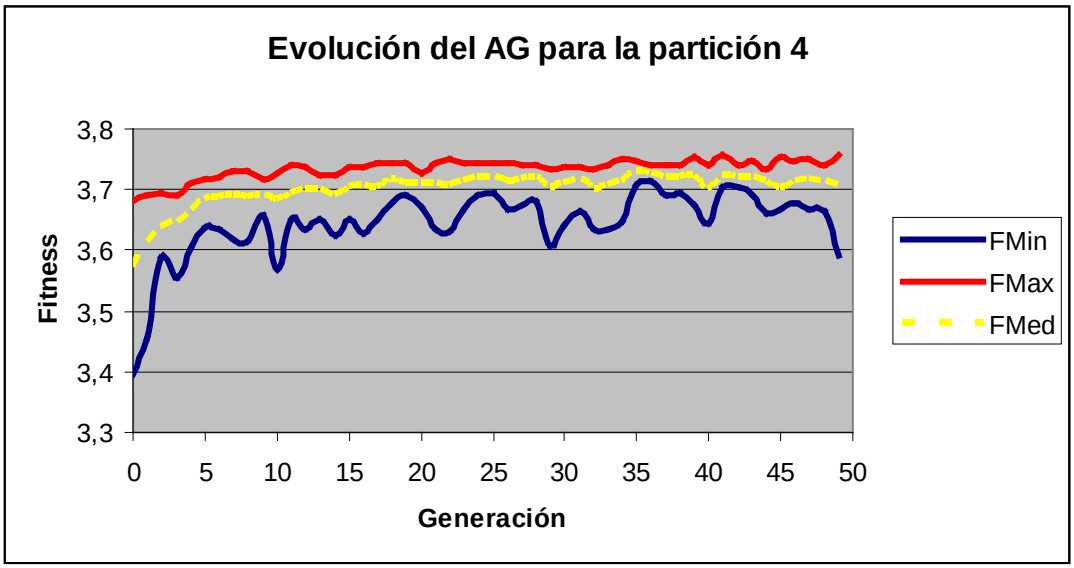


Figura 5.1.3-6: Evolución del AG para la partición 4. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

5.1.4 - Experimentación con matrices de distancias simétricas.

Para la siguiente prueba de experimentación sobre el dominio de la diabetes sólo se va a permitir que se obtenga como solución una matriz de distancias simétrica (Mahalanobbis).

Al igual que en la experimentación anterior (5.1.3), con matrices diagonales, se calculará la a partir del error de entrenamiento de la red y se va a utilizar validación cruzada.

La fitness que se va a emplear para hacer evolucionar el algoritmo genético (AG) se obtiene a partir del error de entrenamiento de la red. Una vez que termina de entrenarse se aplica la transformación logarítmica de ese error para obtener la fitness.

Recordemos que disponemos de 768 patrones, los cuales debido al uso de validación cruzada vamos a dividir en 4 particiones. Por lo tanto vamos a tener 192 patrones en cada partición y 576 patrones en la parte de aprendizaje para proceder a la validación de cada una de las particiones.

Los parámetros del sistema son los siguientes:

- La solución es una matriz de distancias simétrica
- Se emplea validación cruzada: 4 particiones.
- Se utilizan el 100% de los patrones del fichero de aprendizaje de cada partición para el entrenamiento de la red, a partir de los cuales se calcula la fitness.
- Número de generaciones: 50
- Tamaño de la población: 15
- Tamaño del torneo: 2
- Elitismo de 1 elemento
- Cruce de dos puntos
- Probabilidad de cruce: 0,7
- Probabilidad de mutación: 0,01
- Conversión binario a real con una precisión de 2 dígitos en la parte entera binaria y 3 en la parte fraccionaria (LongVal=6; Lpunto=3).

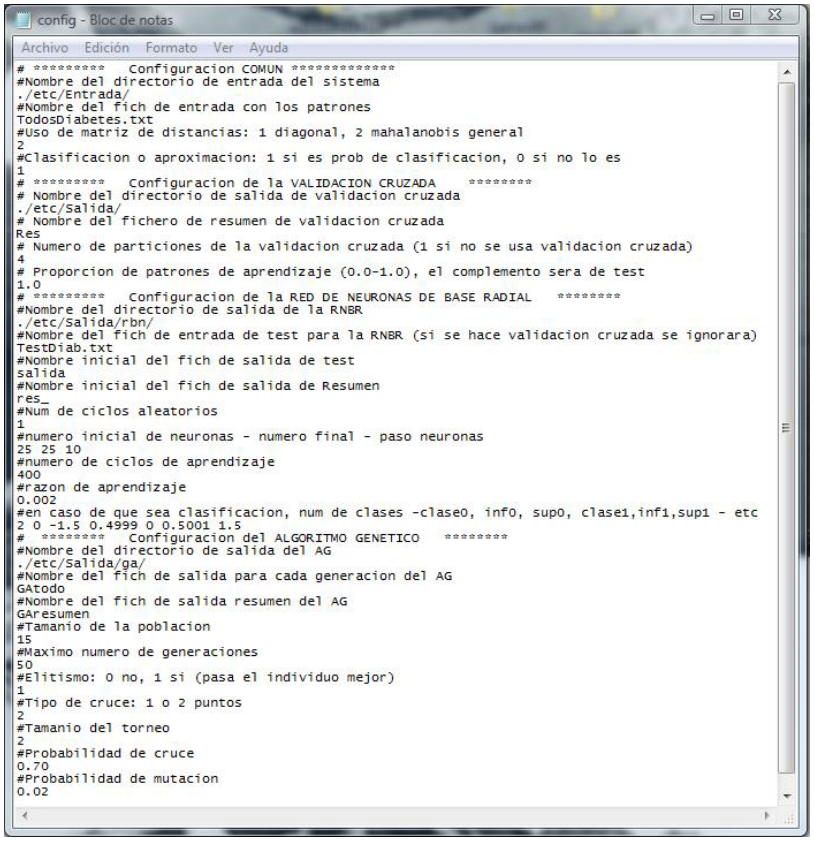


Figura 5.1.4-1: Fichero de configuración para la experimentación con el dominio de la diabetes con matrices simétricas.

Una vez introducidos los parámetros en la aplicación *(ver figura 5.1.4-1)* procedamos a estudiar los resultados obtenidos:

Partición	Fitness con RNBR	Fitness con el sistema	Matriz solución obtenida							
1	0,76016	0,65447	1,5625	-1,375	-0,125	0,1875	-1,625	0,625	-0,6875	-0,75
			-1,375	-0,875	1,875	0,75	0,313	0,313	0,75	1,5625
			-0,125	1,875	2,75	-1,063	3,313	-1,06	0,1875	-2,25
			0,1875	0,75	-1,0625	-0,625	-0,938	0,313	0,3125	0,375
			-1,625	0,3125	3,3125	-0,938	-2,25	-2,25	0,1875	1,3125
			0,625	0,3125	-1,0625	0,3125	-2,25	-0,31	1,5625	1,0625
			-0,6875	0,75	0,1875	0,3125	0,188	1,563	-1,6875	0,75
			-0,75	1,5625	-2,25	0,375	1,313	1,063	0,75	0,75
2	0,73984	0,65447	-0,5	0	-0,3125	-3,5	0,313	-1,25	1,5	0,3125
			0	1	-2,4375	-0,125	0,313	-1	1,25	-1,125
			-0,3125	-2,4375	1,8125	-3,438	0,938	-2,31	-0,1875	-0,875
			-3,5	-0,125	-3,4375	-2,5	0,125	-1,38	1,375	-2,1875
			0,3125	0,3125	0,9375	0,125	0,5	0,813	-1,75	1,125
			-1,25	-1	-2,3125	-1,375	0,813	-0,31	0,9375	-0,625
			1,5	1,25	-0,1875	1,375	-1,75	0,938	0,625	2,375
			0,3125	-1,125	-0,875	-2,188	1,125	-0,63	2,375	-2,875
3	0,80081	0,76423	-1,0625	1,1875	0,75	-0,75	-0,375	2,063	0,6875	0,5625
			1,1875	-1,8125	-1,9375	0,375	0	0,75	-0,3125	-1,375
			0,75	-1,9375	1,5	-0,375	0,313	-0,56	1,5625	-1,6875
			-0,75	0,375	-0,375	2,6875	-2,25	1,25	2,0625	0,8125
			-0,375	0	0,3125	-2,25	2,313	-1,06	0,25	-0,9375
			2,0625	0,75	-0,5625	1,25	-1,063	2	1,4375	1,1875
			0,6875	-0,3125	1,5625	2,0625	0,25	1,438	0,8125	0
			0,5625	-1,375	-1,6875	0,8125	-0,938	1,188	0	-0,8125
4	0,97967	0,95935	1,0625	-0,75	3,375	-2,125	-1,625	1,25	0,75	0,8125
			-0,75	1,9375	0,8125	0,25	0	0,875	0	0,4375
			3,375	0,8125	0,8125	-2,375	-0,688	2,813	2,0625	-0,125
			-2,125	0,25	-2,375	0,5	0,5	-0,88	0,25	3,25
			-1,625	0	-0,6875	0,5	-0,563	-0,13	-1,5	1,375
			1,25	0,875	2,8125	-0,875	-0,125	1	1,75	-0,4375
			0,75	0	2,0625	0,25	-1,5	1,75	1,875	-0,75
			0,8125	0,4375	-0,125	3,25	1,375	-0,44	-0,75	-0,625

Tabla 5.1.4-1: Resultados obtenidos para cada una de las particiones de validación cruzada restringiendo el resultado a matriz simétrica con fitness calculada a partir del error de entrenamiento.

En el gráfico *(ver figura 5.1.4-2)* podemos observar representada la tasa de acierto para cada una de las particiones. Las últimas columnas de la gráfica se corresponden con la tasa de acierto media:

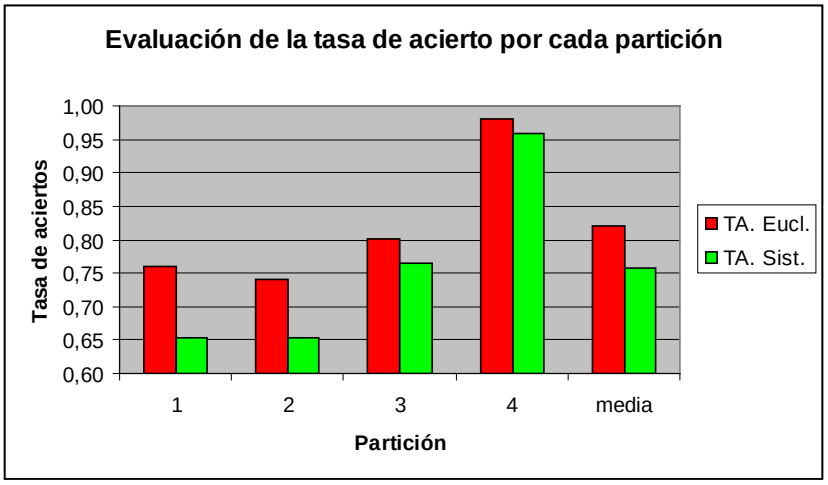


Figura 5.1.4-2: Evaluación de las particiones. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

Dada la tasa de aciertos media obtenida en el sistema *(0,7581)* y en la RNBR *(0,8201)* se puede observar como los resultados empeoran en relación a la utilización de matrices diagonales, siendo el número de aciertos del sistema muy inferior a la utilización de una red neuronal de base radial clásica. La red neuronal en este caso obtiene un 7% más de aciertos respecto a nuestro sistema, alcanzando un 82% de aciertos de media.

Esto queda de manifiesto sobre todo en las dos primeras particiones donde la tasa de aciertos del sistema es muy inferior a la de una RNBR, habiendo una diferencia en el número de aciertos de la red respecto al sistema de más de un 10% en la primera partición (0,7601 de la red por 0, 6544 del sistema) y de casi un 9% en la segunda (0,7398 por 0,6544).

En la tercera partición el sistema mejora los resultados de las dos particiones anteriores llegando a una tasa de aciertos de 0, 7642, pero se ve igualmente superada por el número de aciertos de la red (0,8001).

Es en la cuarta partición en la cual el sistema obtiene su mayor número de aciertos superando el 95% (0,9593), pero de nuevo los resultados de la red son mejores llegando casi a alcanzar un 98% de aciertos (0,9796).

En las siguientes gráficas podemos observar la evolución de la fitness para cada una de las particiones:

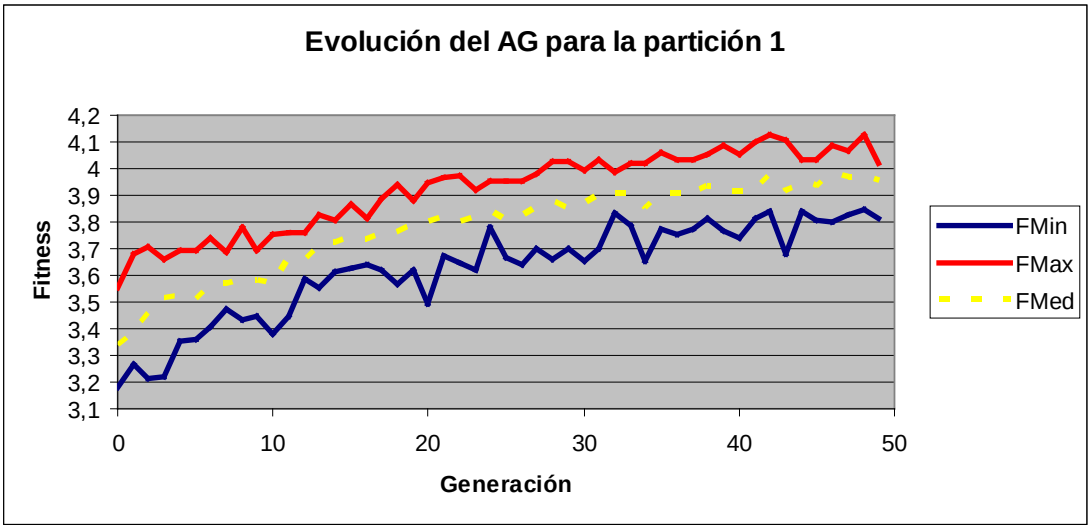


Figura 5.1.4-3: Evolución del AG para la partición 1. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

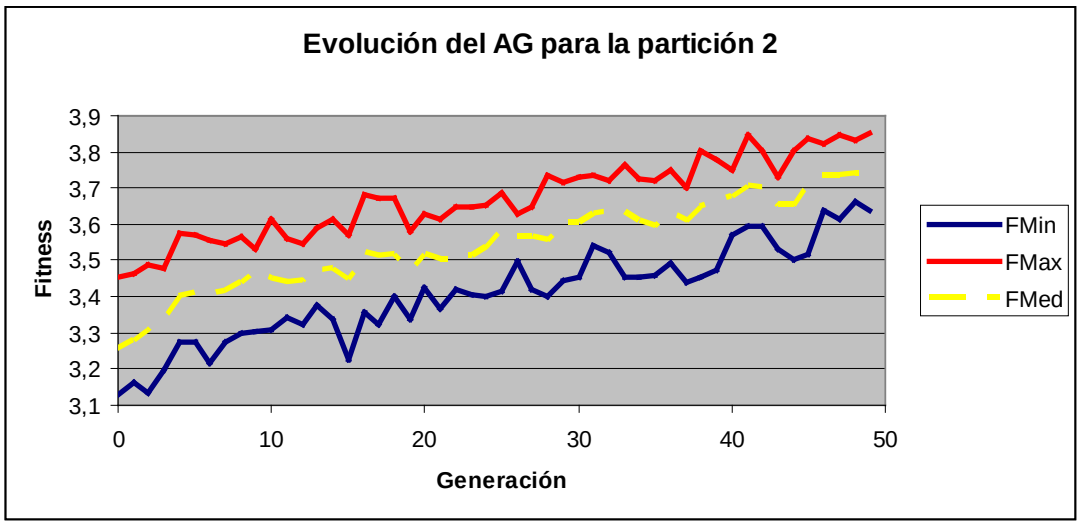


Figura 5.1.4-4: Evolución del AG para la partición 2. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

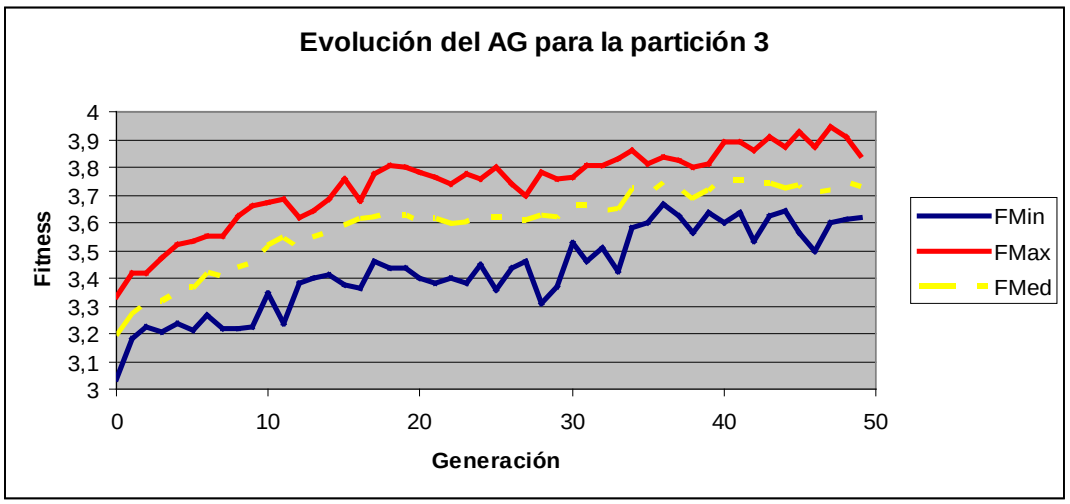


Figura 5.1.4-5: Evolución del AG para la partición 3. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

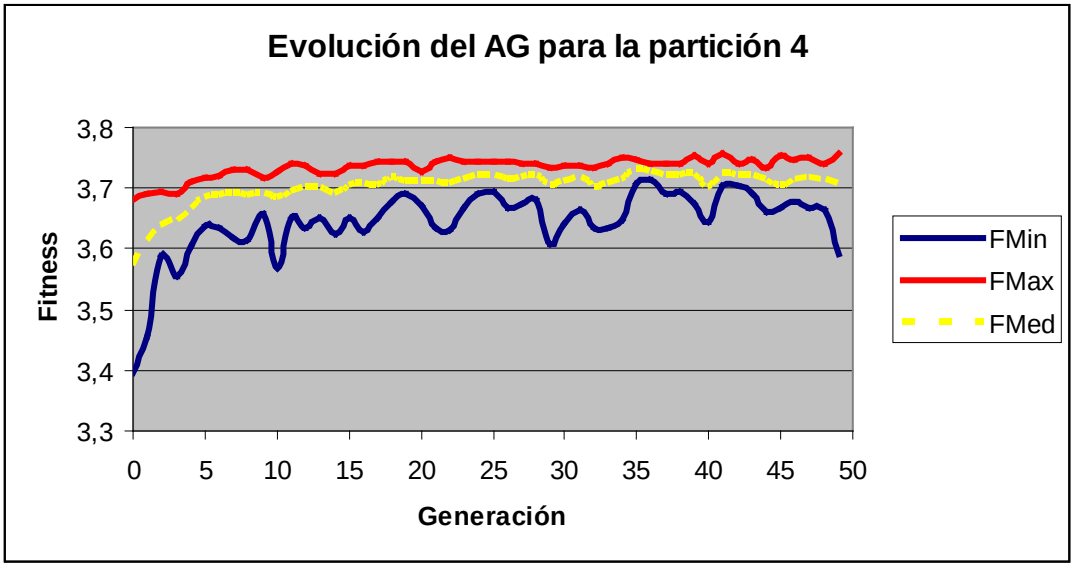


Figura 5.1.4-6: Evolución del AG para la partición 4. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

5.2 - DOMINIO DE CLASIFICACIÓN DE RIPLEY

5.2.1.- Introducción

Este conjunto de datos generados artificialmente ha sido utilizado en [Ripley, 25].Cada modelo tiene dos coordenadas de valor real y una clase que puede ser 0 ó 1. Cada clase corresponde a una distribución bimodal que es una composición equilibrada de dos composiciones normales.

Las matrices de covarianza son idénticas para todas las distribuciones y los centros son diferentes.

Una de las cuestiones que hacen a este campo interesante es el gran solapamiento que existe entre ambas clases. Debido a este fuerte solapamiento, por lo general la RNBR obtiene resultados mediocres con este campo.

Aquí, estamos interesados en la mejora del funcionamiento de la RNBR clásica utilizando distancias generalizadas euclídeas.

Después de realizar la experimentación previa con el dominio de Ripley, de manera similar a como lo hicimos con el dominio de la diabetes (5.1.2), se ha determinado que el número de neuronas que vamos a utilizar es de 25.

El fichero de entrada del dominio de clasificación de Ripley, tiene dos dimensiones o atributos. (*Figura 5.2.1-1).*

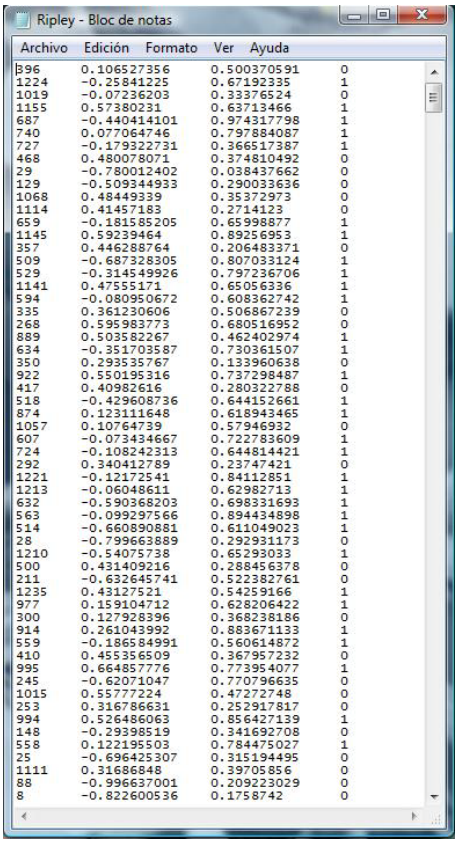


Figura 5.2.1-1: Fragmento del fichero de entrada del dominio de clasificación de Ripley.

5.2.2 - Experimentación con matrices de distancias diagonales.

A continuación vamos a realizar la primera prueba de experimentación sobre el dominio de Ripley utilizando matrices diagonales.

Recordemos que como en el dominio anterior, tanto en soluciones con matrices diagonales como con matrices simétricas, se realizará la experimentación a partir del error de entrenamiento de la red y a partir de la tasa de aciertos al aplicar los patrones de test con el objetivo de poder compararlos.

En ambos casos vamos a realizar el estudio de lo óptima o no que es la matriz obtenida como solución empleando validación cruzada…

La fitness que se va a emplear para hacer evolucionar el algoritmo genético (AG) se obtiene a partir del error de entrenamiento de la red. Una vez que termina de entrenarse se aplica la transformación logarítmica de ese error para obtener la fitness.

En este dominio, disponemos de 1250 patrones, los cuales debido al uso de validación cruzada vamos a dividir en 4 particiones. Por lo tanto en el conjunto de test de cada partición dispondremos de unos 312 patrones de test y 938 patrones de entrenamiento.

Los parámetros del sistema son los siguientes:

- La solución es una matriz de distancias diagonal
- Se emplea validación cruzada: 4 particiones.
- Se utilizan el 100% de los patrones del fichero de aprendizaje de cada partición para el entrenamiento de la red, a partir de los cuales se calcula la fitness.
- Número de generaciones: 50
- Tamaño de la población: 15
- Tamaño del torneo: 2
- Elitismo de 1 elemento
- Cruce de dos puntos
- Probabilidad de cruce: 0,7
- Probabilidad de mutación: 0,01
- Conversión binario a real con una precisión de 2 dígitos en la parte entera binaria y 3 en la parte fraccionaria (LongVal=6; Lpunto=3).

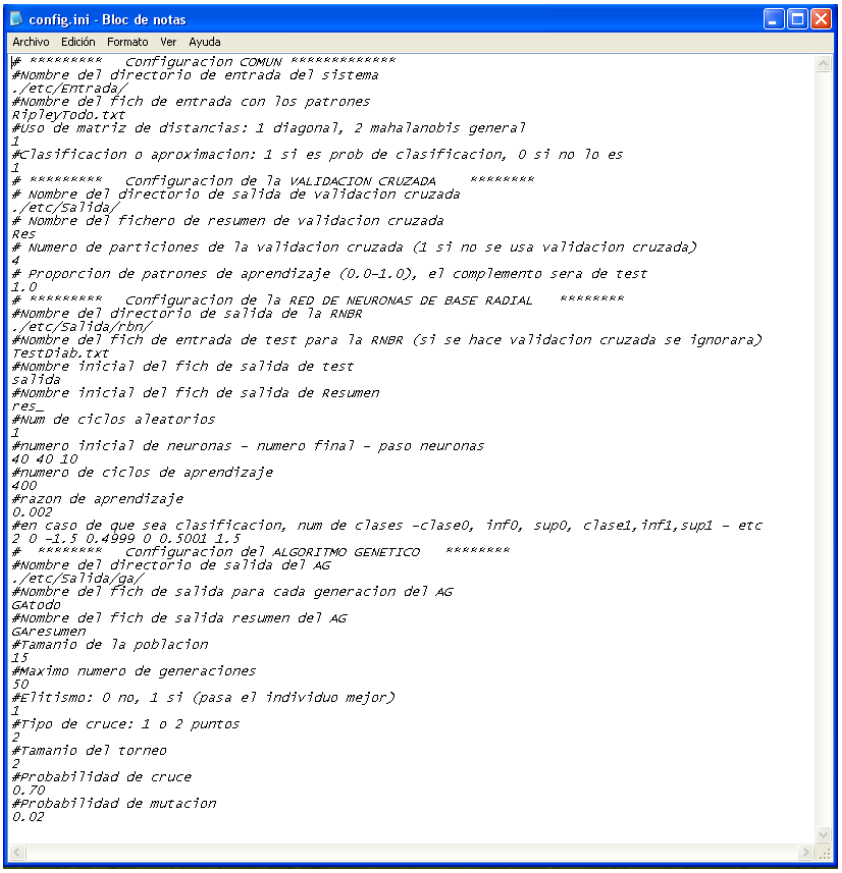


Figura 5.2.2-1: Fichero de configuración utilizado para la experimentación con matrices diagonales en el dominio de clasificación de Ripley.

Una vez introducidos los parámetros en la aplicación *(figura 5.2.2-1)* procedamos a estudiar los resultados obtenidos:

Partición	Fitness con RNBR	Fitness con el sistema	Matriz solución obtenida
1	0,91374	0,89776	-2 0 0 2,437
2	0,88179	0,86581	2,625 0 0 2,375
3	0,89457	0,91054	1,937 0 0 2,437
4	0.91054	0,92013	2,250 0 0 -2,312

Tabla 5.2.2-1: Resultados obtenidos para cada una de las particiones de validación cruzada restringiendo el resultado a matriz diagonal con fitness calculada a partir del error de entrenamiento.

En el gráfico *(Figura 5.2.2-2)* podemos observar representada la tasa de acierto para cada una de las particiones. Las últimas columnas de la gráfica se corresponden con la tasa de acierto media:

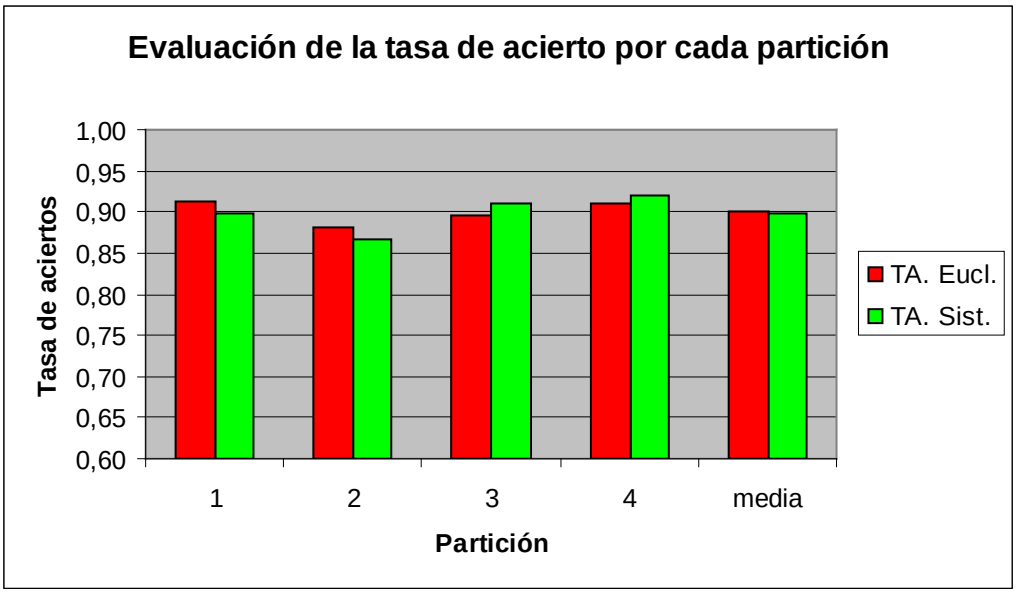


Figura 5.2.2-2: Evaluación de las particiones. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

Si estudiamos la tasa de aciertos media obtenida por el sistema *(0,8956)* y por la RNBR *(0,9001)*, se puede observar como el número de aciertos obtenidos son prácticamente iguales, rozando en ambos casos el 90% de aciertos.

La tasa de aciertos del sistema es ligeramente inferior en la primera y segunda partición (0,9137 de la red por 0,8977 del sistema, en la primera y 0,8817 por 0,8658 en la segunda), pero supera el número de aciertos de la red en la tercera y cuarta partición (0,9105 del sistema por 0,8945, de la red en la tercera y 0,9201 por 0,9105 en la cuarta), lo que hace que en media se pueda decir que tanto usando una red neuronal de base radial clásica como el sistema desarrollado, se obtiene una tasa de acierto similar.

En las siguientes gráficas podemos observar la evolución de la fitness para cada una de las particiones:

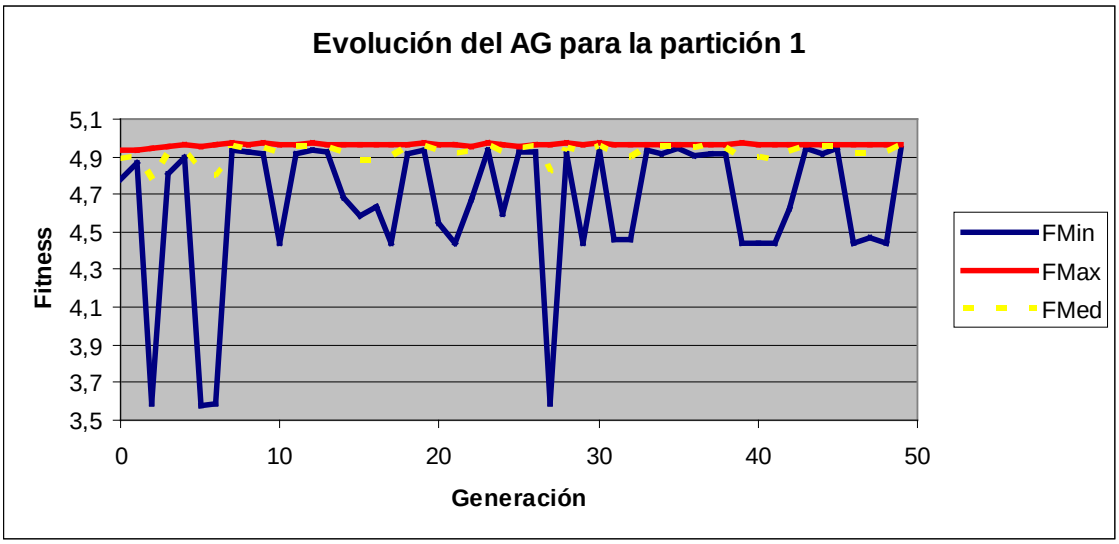


Figura 5.2.2-3: Evolución del AG para la partición 1. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

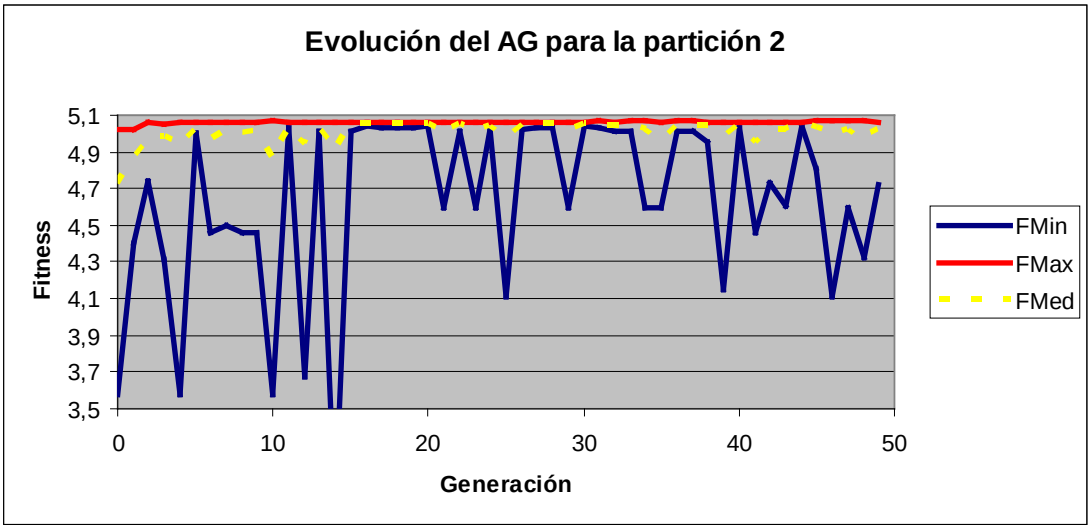


Figura 5.2.2-4: Evolución del AG para la partición 2. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

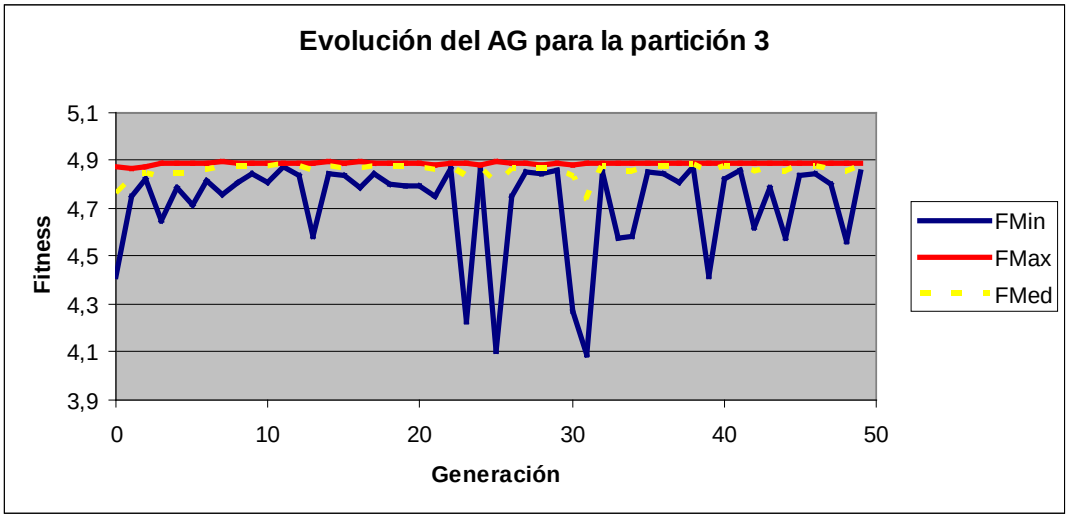


Figura 5.2.2-5: Evolución del AG para la partición 3. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

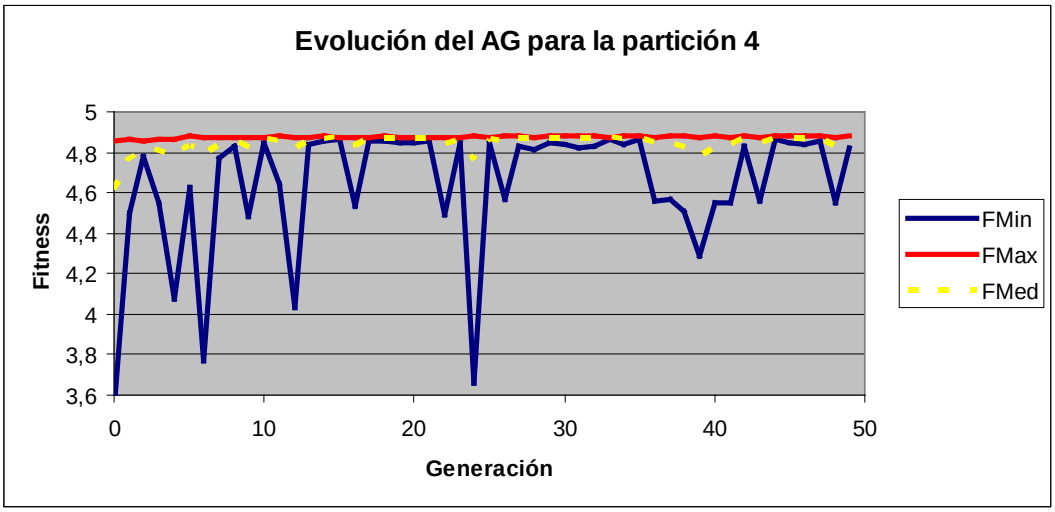


Figura 5.2.2-4: Evolución del AG para la partición 4. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

5.2.3 - Experimentación con matrices de distancias simétricas.

A continuación vamos a realizar la segunda prueba de experimentación sobre el dominio de Ripley, en esta ocasión empleando matrices simétricas.

Al igual que en la experimentación anterior con este dominio (5.2.2), con matrices diagonales, se calculará la a partir del error de entrenamiento de la red y se va a utilizar validación cruzada, para realizar el estudio de lo óptimo n no que es la matriz obtenida como solución.

La fitness que se va a emplear para hacer evolucionar el algoritmo genético (AG) se obtiene a partir del error de entrenamiento de la red. Una vez que termina de entrenarse se aplica la transformación logarítmica de ese error para obtener la fitness.

En el dominio de Ripley, como ya hemos expuesto anteriormente, disponemos de 1250 patrones, los cuales debido al uso de validación cruzada vamos a dividir en 4 particiones. Por lo tanto en cada partición dispondremos de unos 312 patrones de test y 936 patrones de entrenamiento.

Los parámetros del sistema para este experimento son los siguientes:

- La solución es una matriz de distancias simétrica
- Se emplea validación cruzada: 4 particiones.
- Se utilizan el 100% de los patrones del fichero de aprendizaje de cada partición para el entrenamiento de la red, a partir de los cuales se calcula la fitness.
- Número de generaciones: 50
- Tamaño de la población: 15
- Tamaño del torneo: 2
- Elitismo de 1 elemento
- Cruce de dos puntos
- Probabilidad de cruce: 0,7
- Probabilidad de mutación: 0,01
- Conversión binario a real con una precisión de 2 dígitos en la parte entera binaria y 3 en la parte fraccionaria (LongVal=6; Lpunto=3).

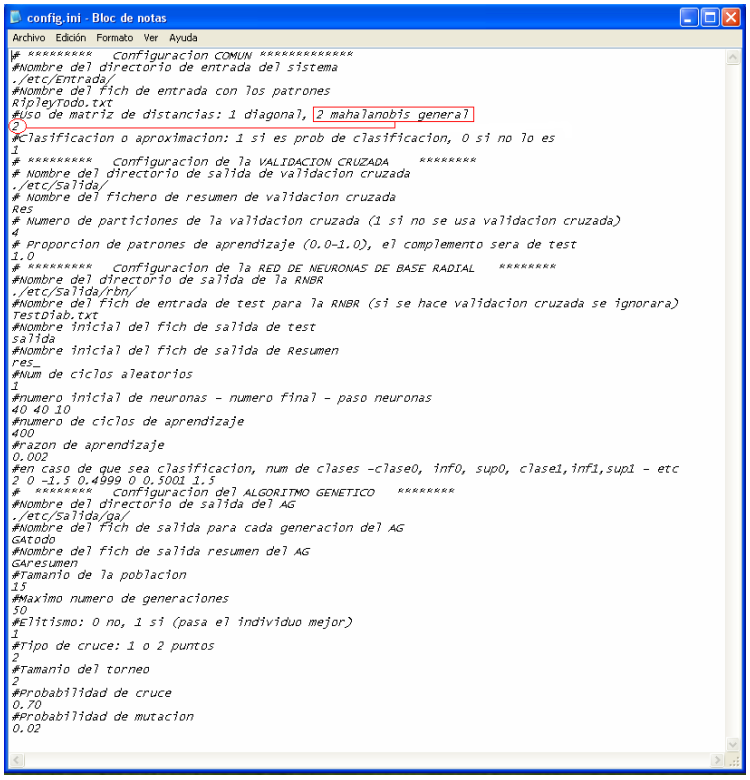


Figura 5.2.3-1: Fichero "config.ini" utilizado para la experimentación con matrices simétricas en el dominio de clasificación de Ripley.

Una vez introducidos los parámetros en la aplicación *(figura 5.2.3-1)* procedamos a estudiar los resultados obtenidos:

Partición	Tasa de aciertos RNBR	Tasa de aciertos del sistema	Matriz solución obtenida
1	0,90415	0,90415	[2,0000 0,1875 ; 0,1875 -2,2500]
2	0,87859	0,86851	[2,3125 -0,9375 ; -0,9375 -2,3750]
3	0,88818	0,90096	[2,4375 0,5625 ; 0,5625 -1,3125]
4	0,91374	0,92013	[0,1875 -2,1250 ; -2,1250 0,2500]

Tabla 5.2.3-1: Resultados obtenidos para cada una de las particiones de validación cruzada restringiendo el resultado a matriz simétrica con fitness calculada a partir del error de entrenamiento.

En el gráfico *(Figura 5.2.3-2)* podemos observar representada la tasa de acierto para cada una de las particiones. Las últimas columnas de la gráfica se corresponden con la tasa de acierto media:

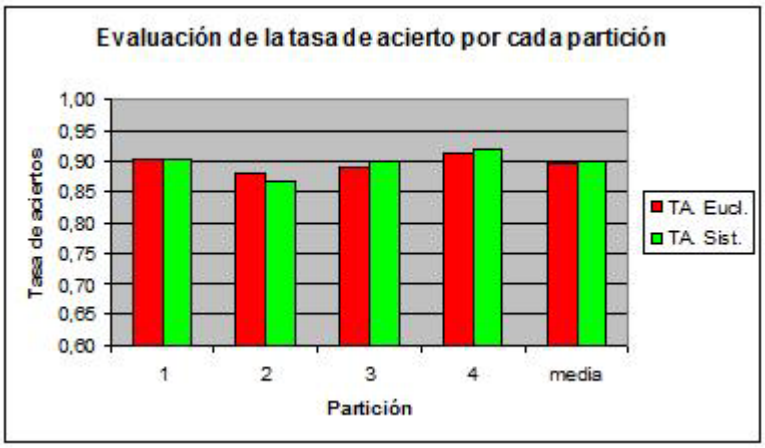


Figura 5.2.3-2: Evaluación de las particiones. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

Dada la tasa de aciertos media obtenida en el sistema *(0,8977)* y en la RNBR *(0,8961)* se puede observar como las tasas de aciertos obtenidas, al igual que en el experimento anterior con el dominio Ripley y matriz diagonal (*Figura 5.2.2-2*) son prácticamente idénticas

La tasa de aciertos del sistema y de la RNBR en las cuatro particiones es prácticamente idéntica, igualándose en la primera partición (0,9041), siendo ligeramente inferior la tasa de aciertos del sistema en la segunda partición (0,8785 la tasa de aciertos por parte de la red neuronal por 0,8658 del sistema) y siendo algo superior en la tercera (0,8881 por 0,9009) y cuarta partición (0,9137 por 0,9201).

A nivel general, se puede concluir que para el dominio de Ripley, utilizando matrices simétricas, los resultados obtenidos son, por muy poco, mejores que si se utilizara una RNBR normal, llegando a alcanzar un 90% de aciertos.

En las siguientes gráficas podemos observar la evolución de la fitness para cada una de las particiones:

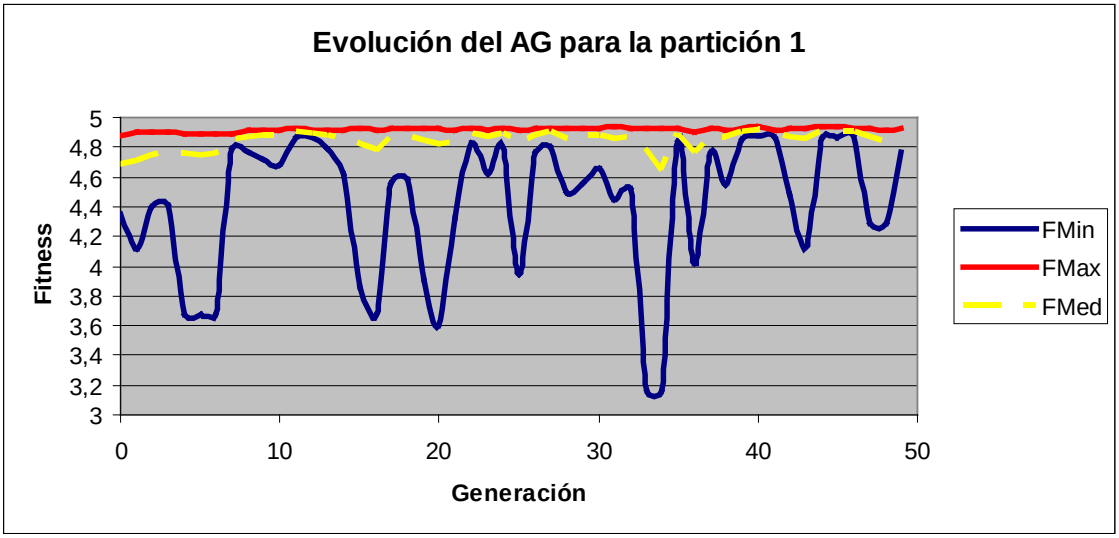


Figura 5.2.3-3: Evolución del AG para la partición 1. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

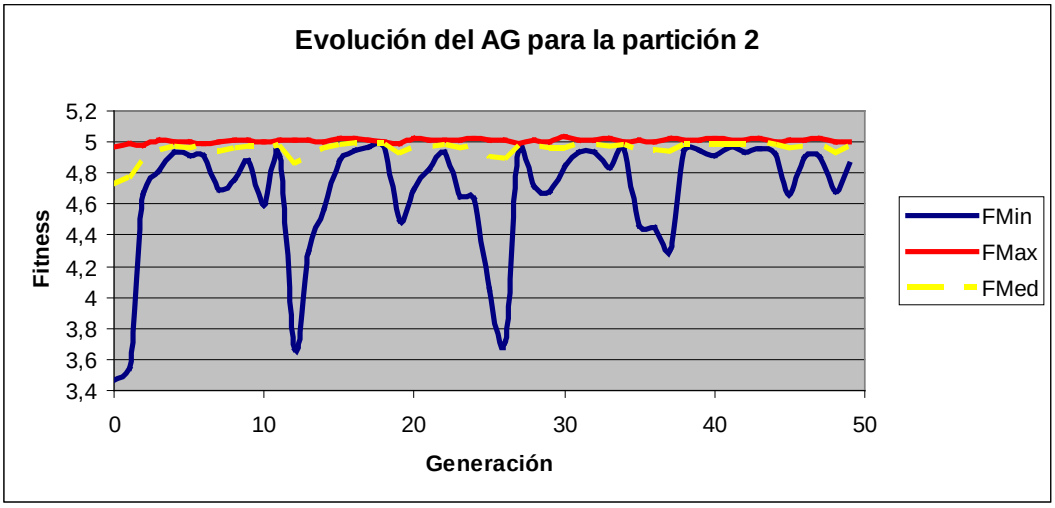


Figura 5.2.3-4: Evolución del AG para la partición 2. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

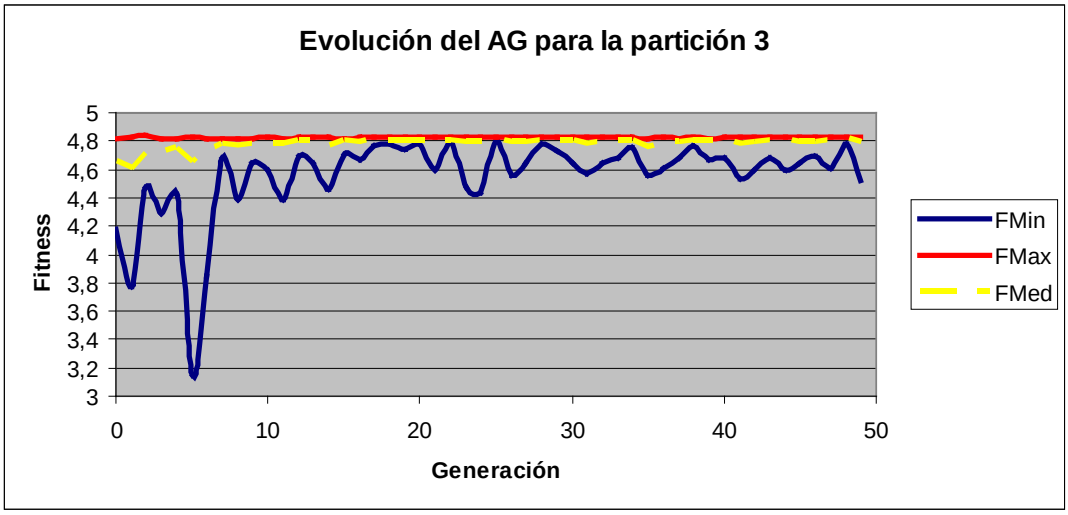


Figura 5.2.3-5: Evolución del AG para la partición 3. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

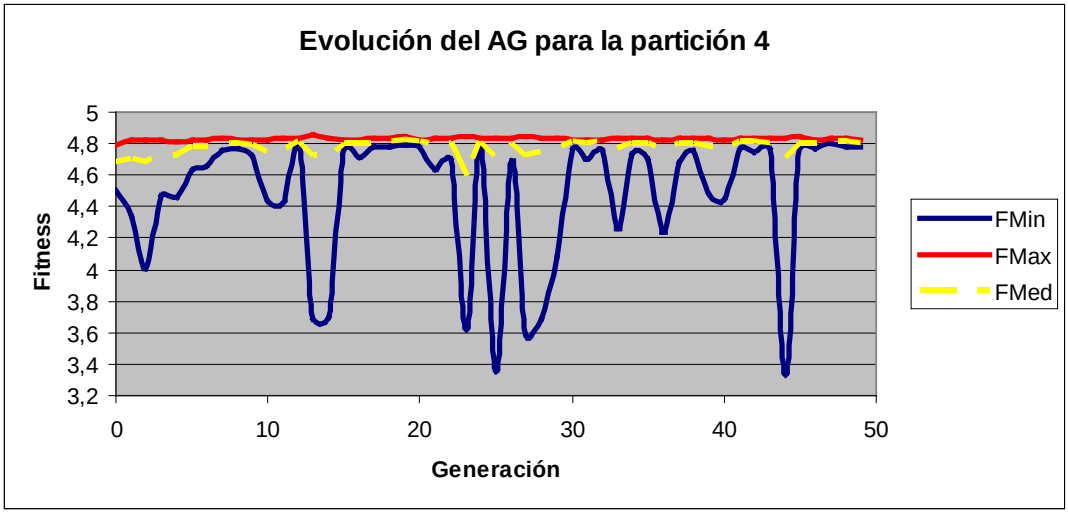


Figura 5.2.3-6: Evolución del AG para la partición 4. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

5.3.- DOMINIO DE CLASIFICACIÓN DE RECTAS 45

5.3.1 – Introducción

El dominio de clasificación rectas 45, ha sido generado artificialmente de manera aleatoria.

Los ejemplos, a partir de ahora "instancias", han sido generados de la siguiente manera:

- 100 instancias con valor de clasificación 1, que están localizados en intervalos regulares a lo largo de una recta que pasa por el origen (0,0) con un ángulo de 45º respecto de la horizontal. La distancia entre dos puntos consecutivos es de 1.
- 100 instancias con valor de clasificación 0, que están generados de la misma manera que los anteriores, localizándolos en una línea paralela que pasa por el punto (0,1), de tal manera que el punto mas cercano a uno dado siempre pertenece a la clase (valor de clasificación) opuesta.

A continuación, todas las instancias anteriores, en este caso 200, son alterados añadiéndoles a cada uno de manera aleatoria un número entre -0,5 y +0,5 de manera uniforme.

La intención que lleva a crear este dominio, no es mas que observar como el algoritmo vecino mas cercano alcanzará un resultado pésimo, porque en la mayoría de los casos el vecino mas cercano será de la clase opuesta, es decir, su valor de clasificación será el contrario. Pero ciertas transformaciones de los datos que implican rotaciones y el escalamiento de coordenadas van a permitir una tasa de aciertos bastante buena.

La siguiente figura (*5.5.1-1*) muestra un subconjunto de datos de este dominio.

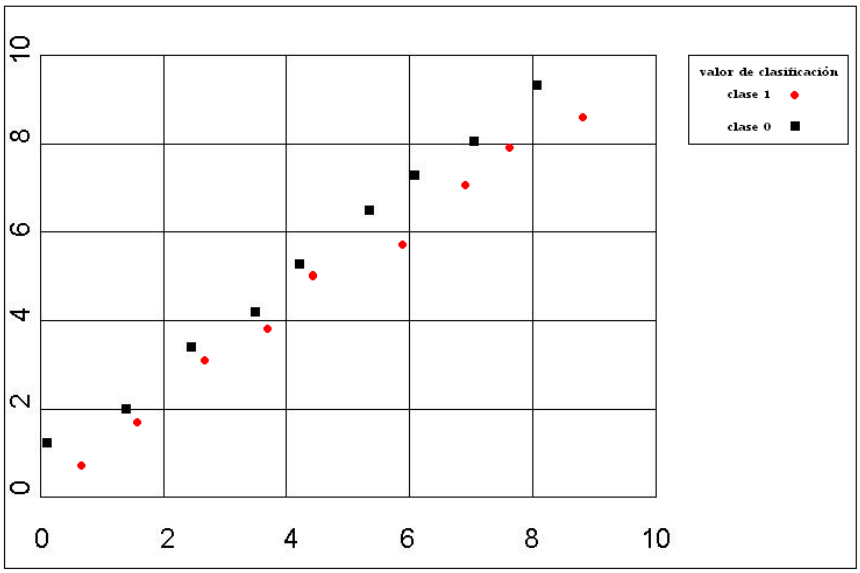


Figura 5.5.1.1-1: Subconjunto de datos del dominio rectas 45

El fichero de entrada del dominio de clasificación de rectas ruido, tiene dos instancias o atributos (*Figura 5.3.1-1*), por lo que todas las matrices solución que se obtengan durante la experimentación con el mismo, serán de 2 dimensiones.

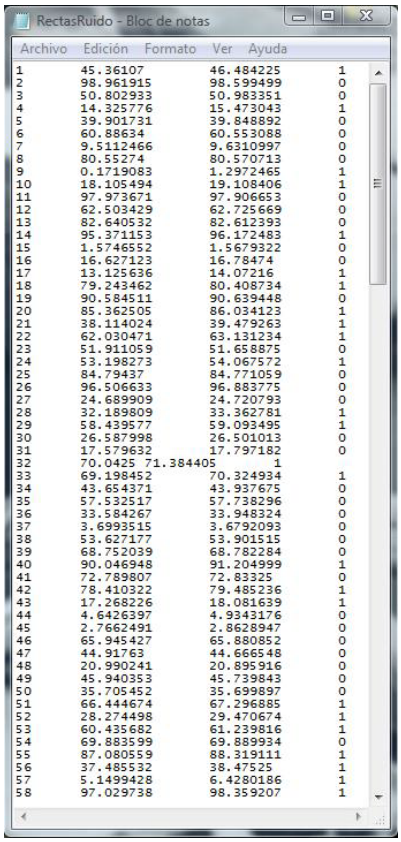


Figura 5.3.1-1: Fragmento del fichero de entrada del dominio rectas 45.

5.3.2 - Experimentación con matrices de distancias diagonales.

Antes de comenzar con la primera prueba de experimentación sobre el dominio Rectas 45, que al igual que experimentos con dominios anteriores va a consistir en la obtención como solución de una matriz de distancias diagonal, tenemos que tener presente que este dominio es sintético, puesto que está generado artificialmente.

Como ya se ha visto en anteriores pruebas de experimentación (dominio de diabetes y Ripley) en este proyecto, tanto en soluciones con matrices diagonales como simétricas, realizamos los experimentos calculando la fitness a partir del error de entrenamiento de la red.

En cualquiera de los casos vamos a realizar el estudio de lo óptima o no que es la matriz obtenida utilizando validación cruzada.

La fitness que se va a emplear para hacer evolucionar el algoritmo genético (AG) se obtiene a partir del error de entrenamiento de la red. Una vez que termina de entrenarse se aplica la transformación logarítmica de ese error para obtener la fitness.

En el caso del dominio rectas 45 disponemos de 200 patrones, los cuales debido al uso de validación cruzada vamos a dividir en 4 particiones. Por lo tanto vamos a tener en cada partición 50 patrones de test y 150 patrones de entrenamiento para proceder a la validación de cada una de las particiones.

Los parámetros del sistema, para realizar el experimento sobre este dominio con matrices diagonales, son los siguientes:

- La solución es una matriz de distancias diagonal
- Se emplea validación cruzada: 4 particiones.
- Se utilizan el 100% de los patrones del fichero de aprendizaje de cada partición para el entrenamiento de la red, a partir de los cuales se calcula la fitness.
- Número de generaciones: 50
- Tamaño de la población: 15
- Tamaño del torneo: 2
- Elitismo de 1 elemento
- Cruce de dos puntos
- Probabilidad de cruce: 0,7
- Probabilidad de mutación: 0,01
- Conversión binario a real con una precisión de 2 dígitos en la parte entera binaria y 3 en la parte fraccionaria (LongVal=6; Lpunto=3).

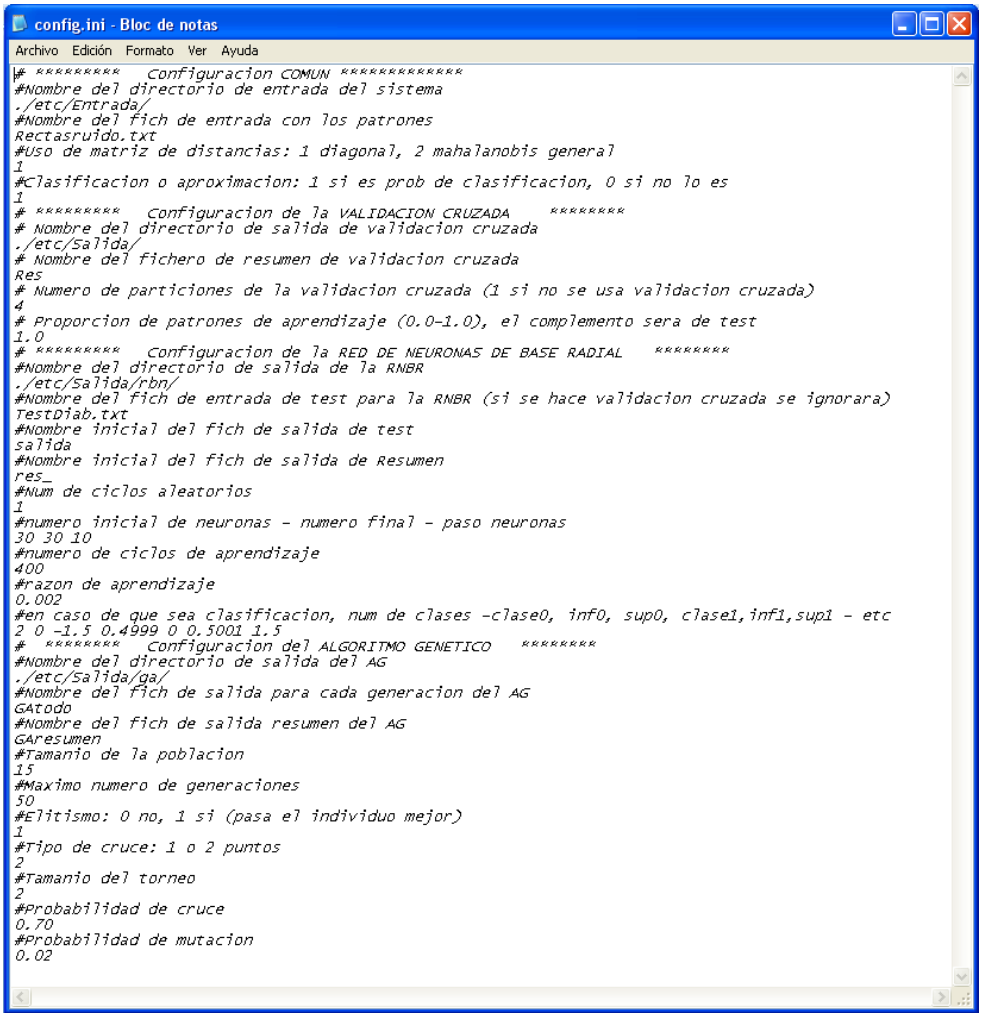


Figura 5.3.2-1: Fichero "config.ini" utilizado para la experimentación con matrices diagonales en el dominio de clasificación de Rectas 45.

Una vez introducidos los parámetros en la aplicación *(Figura 5.3.2-1)* procedamos a estudiar los resultados obtenidos:

Partición	Fitness con RNBR	Fitness con el sistema	Matriz solución obtenida
1	0,5098	0,45098	0,0625 0 0 0,5
2	0,43137	0,33333	0,75 0 0 0
3	0,39216	0,31373	0,9375 0 0 0,0625
4	0,41176	0,29412	0,5625 0 0 0,5625

Tabla 5.3.2-1: Resultados obtenidos para cada una de las particiones de validación cruzada restringiendo el resultado a matriz diagonal con fitness calculada a partir del error de entrenamiento.

En el gráfico *(Figura 5.3.2-2)* podemos observar representada la tasa de acierto para cada una de las particiones. La última columna de la gráfica se corresponde con la tasa de acierto media:

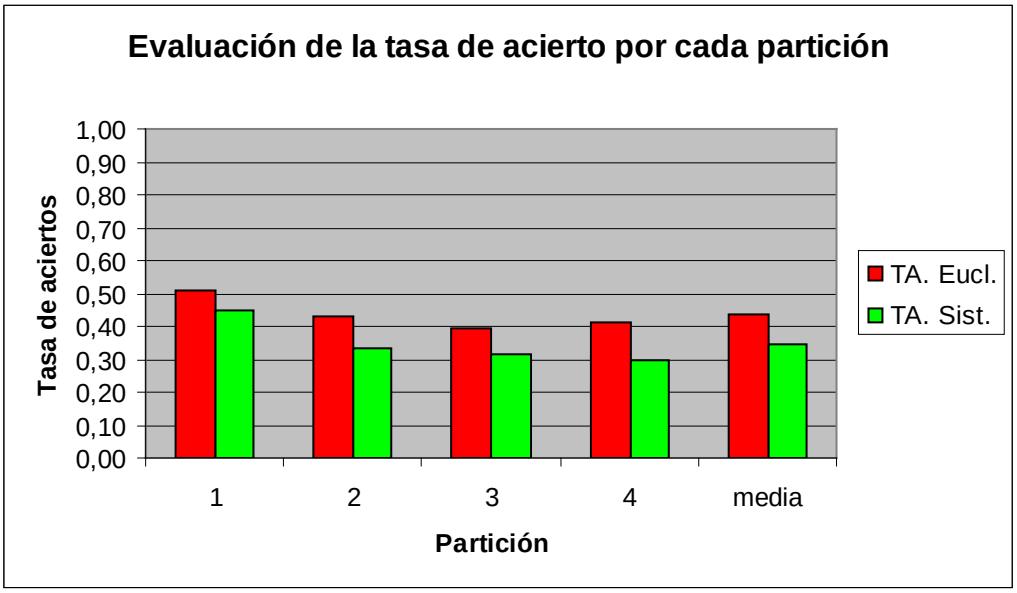


Figura 5.3.2-2: Evaluación de las particiones. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

Dada la tasa de aciertos media obtenida en el sistema *(0,3480)* y en la RNBR *(0,4342)* se puede observar como la tasa de aciertos obtenida es bastante inferior (un 11%) en nuestro sistema. En cada una de las particiones la tasa de acierto siempre es superior utilizando una Red Neuronal de Base Radial que empleando el sistema. Se puede observar que a cada partición, el número de aciertos del sistema, en comparación con los obtenidos por la RNBR, se va reduciendo, siendo la diferencia entre ambas de un 5% en la primera partición (0,5098 de la red por 0,4509 del sistema) y llegando a un 10% en la cuarta partición (0,4117 por 0,2941).

Por lo tanto se puede decir que en el caso del dominio rectas 45, utilizando matrices diagonales como solución, no se obtienen mejores resultados con el sistema.

A pesar de ello la RNBR no obtiene una tasa de aciertos muy óptima, puesto que su número de aciertos está por debajo del 40%.

En las siguientes gráficas podemos observar la evolución de la fitness para cada una de las particiones:

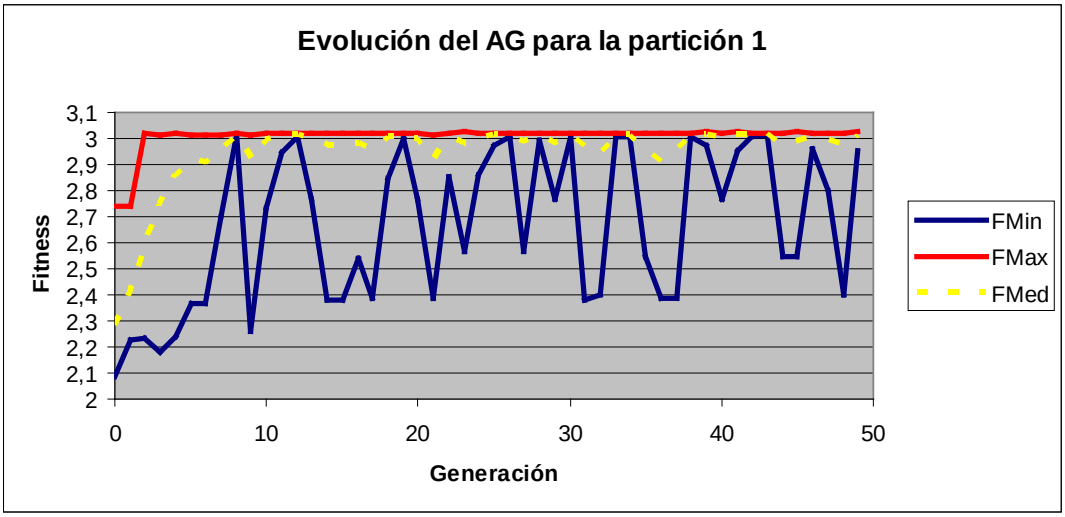


Figura 5.3.2-3: Evolución del AG para la partición 1. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

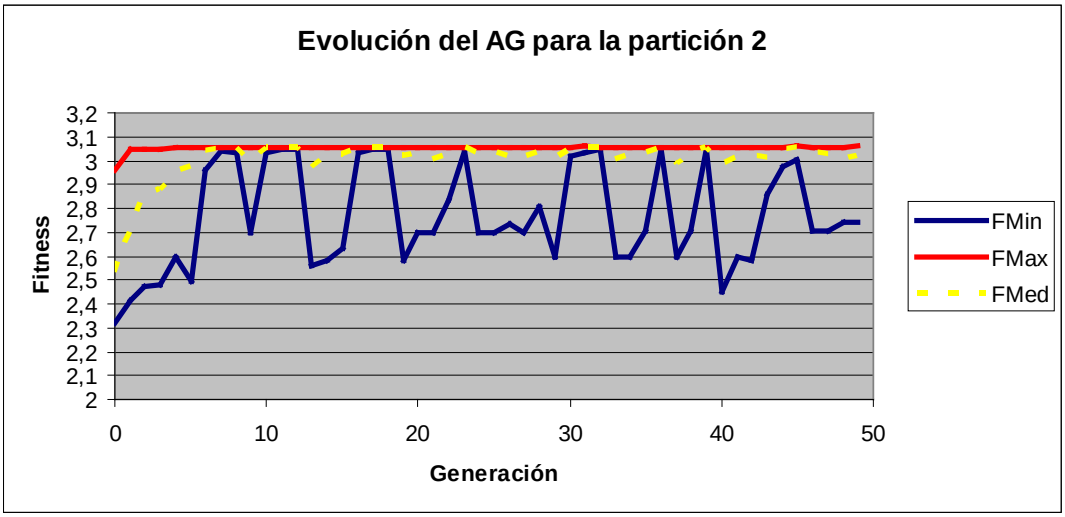


Figura 5.3.2-4: Evolución del AG para la partición 2. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

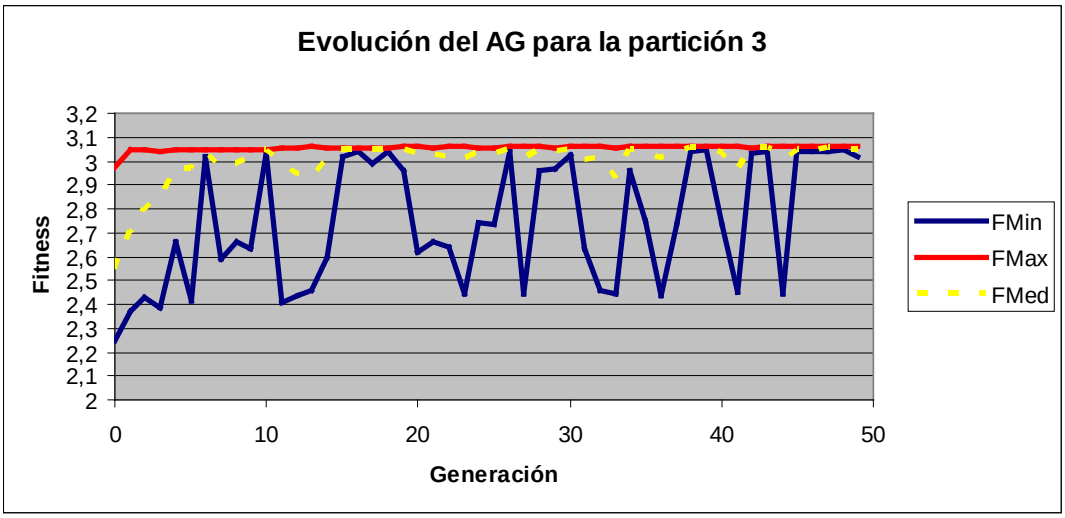


Figura 5.3.2-5: Evolución del AG para la partición 3. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

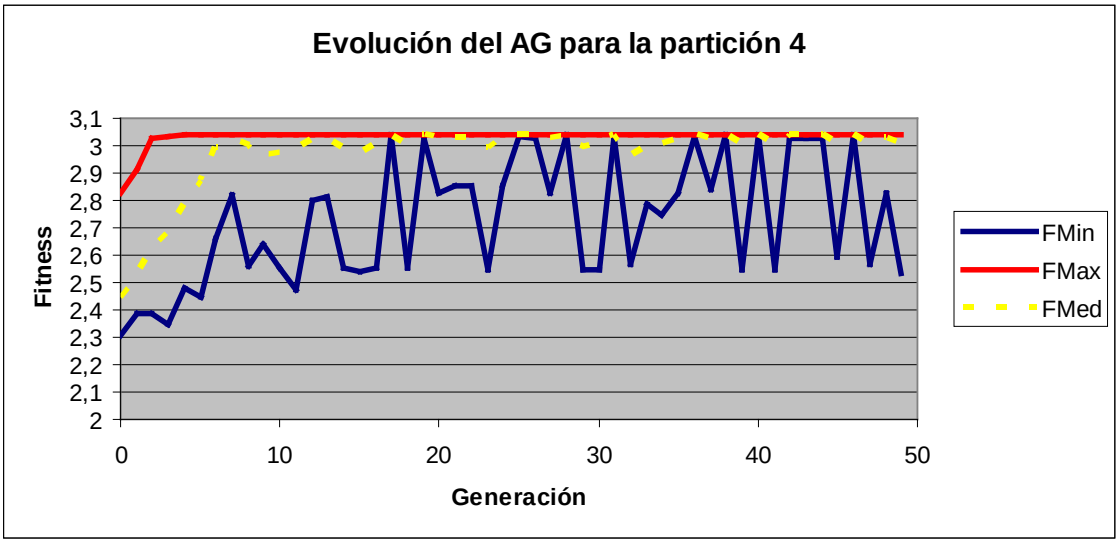


Figura 5.3.2-6: Evolución del AG para la partición 4. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

5.3.3 - Experimentación con matrices de distancias simétricas.

Para la siguiente prueba de experimentación sobre el dominio rectas 45 se van a utilizar matrices simétricas.

Como en el caso anterior de experimentación, para este dominio, se realizará el experimento calculando la fitness a partir del error de entrenamiento de la red y se va a utilizar validación cruzada.

Recordemos que para el dominio de rectas 45, disponemos de 200 patrones, los cuales debido al uso de validación cruzada vamos a dividir en 4 particiones. Por lo tanto tendremos 50 patrones en cada partición y 150 patrones en la parte de aprendizaje para proceder a la validación de cada una de las particiones.

Los parámetros del sistema son los siguientes:

- La solución es una matriz de distancias simétrica
- Se emplea validación cruzada: 4 particiones.
- Se utilizan el 100% de los patrones del fichero de aprendizaje de cada partición para el entrenamiento de la red, a partir de los cuales se calcula la fitness.
- Número de generaciones: 50
- Tamaño de la población: 15
- Tamaño del torneo: 2
- Elitismo de 1 elemento
- Cruce de dos puntos
- Probabilidad de cruce: 0,7
- Probabilidad de mutación: 0,01
- Conversión binario a real con una precisión de 2 dígitos en la parte entera binaria y 3 en la parte fraccionaria (LongVal=6; Lpunto=3).

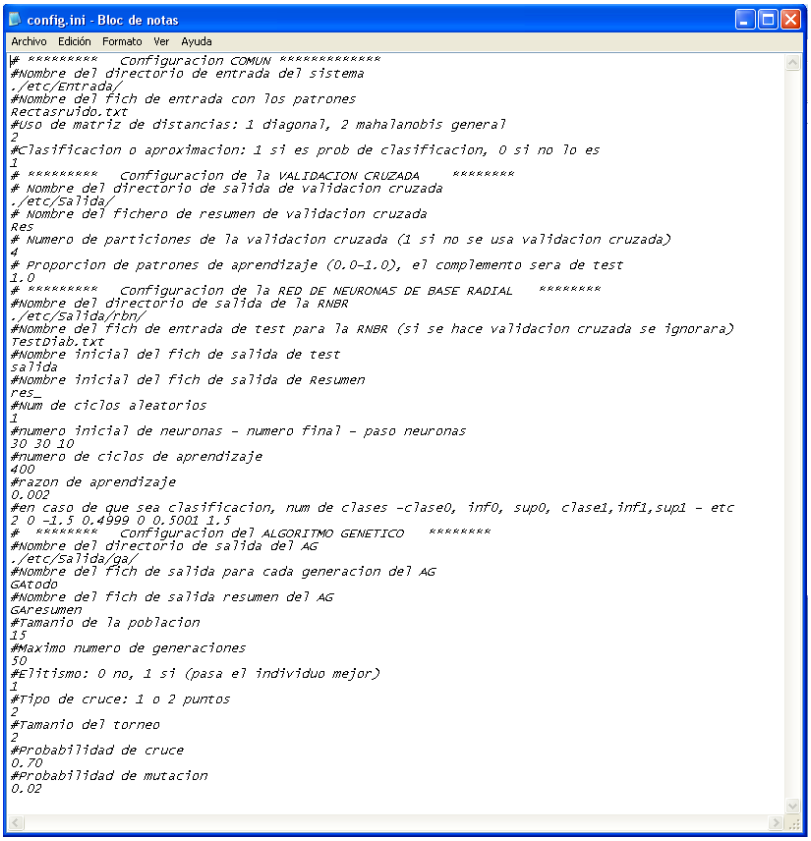


Figura 5.3.3-1: Fichero de configuración para el dominio rectas 45, utilizando una matriz simétrica como solución.

Una vez introducidos los parámetros en la aplicación *(ver figura 5.3.3.-1)* procedamos a estudiar los resultados obtenidos:

Partición	Fitness con RNBR	Fitness con el sistema	Matriz solución obtenida
1	0,5098	1,0000	1,3125 -1,3125 -1,3125 1,3125
2	0,43137	0,78431	1,9375 -2,125 -2,125 2,0625
3	0,39216	0,92157	2 -1,9375 -1,9375 2
4	0,41176	0,98039	-2,5 2,4375 2,4375 -2,5

Tabla 5.3.3-1: Resultados obtenidos para cada una de las particiones de validación cruzada restringiendo el resultado a matriz simétrica con fitness calculada a partir del error de entrenamiento.

En el gráfico *(Figura 5.3.3-2)* podemos observar representada la tasa de acierto para cada una de las particiones. Las últimas columnas de la gráfica se corresponden con la tasa de acierto media:

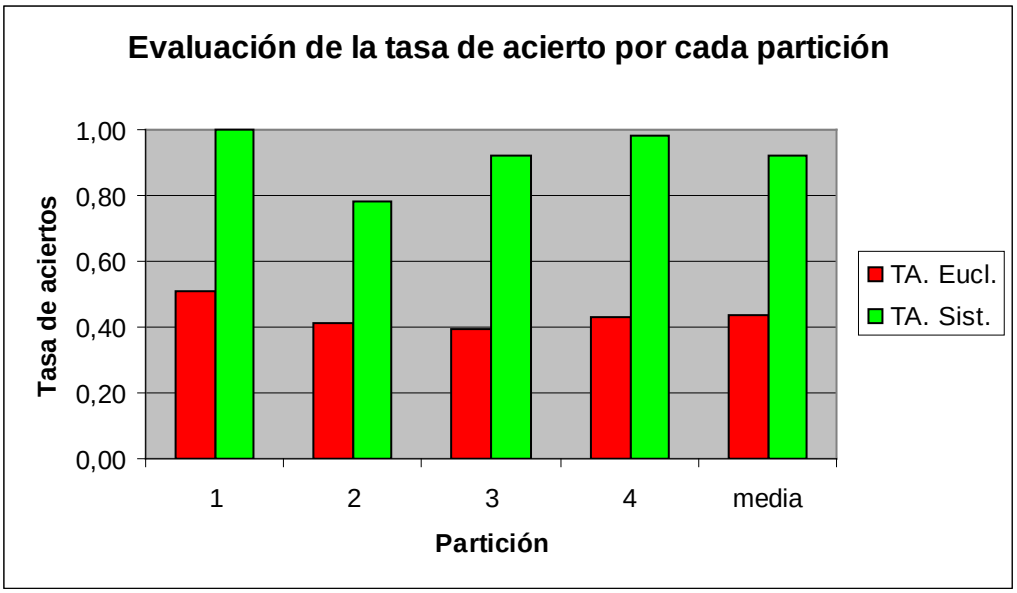


Figura 5.3.3-2: Evaluación de las particiones. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

Dada la tasa de aciertos media obtenida en el sistema *(0,9215)* y en la RNBR *(0,4362)* se puede observar que la precisión que obtiene el sistema, mejora sustancialmente la obtenida por la RNBR, llegándose a acercar en media al 100% de aciertos.

Esto queda de manifiesto en las cuatro particiones, llamando la atención, sobre el resto, la partición 1, en la cual nuestro sistema obtiene un 100% de aciertos.

En el resto de particiones, la tasa de aciertos de nuestro sistema llega a superar el 90% de aciertos, a excepción de la partición 2 en la cual hay un descenso en la misma, que a pesar de ello, llega a rozar el 80% de aciertos.

Por tanto, se puede concluir que para el dominio Rectas 45 restringiendo la matriz solución a ser simétrica, nuestro sistema obtiene resultados más óptimos que una RNBR (un 49% más de aciertos), alcanzado un 92% de media.

En las siguientes gráficas podemos observar la evolución de la fitness para cada una de las particiones:

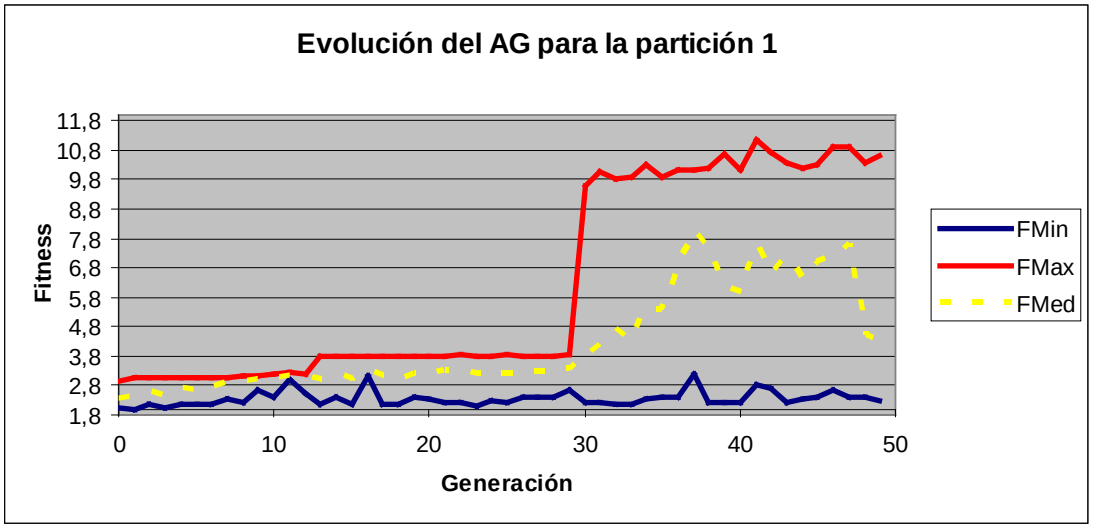


Figura 5.3.3-3: Evolución del AG para la partición 1. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

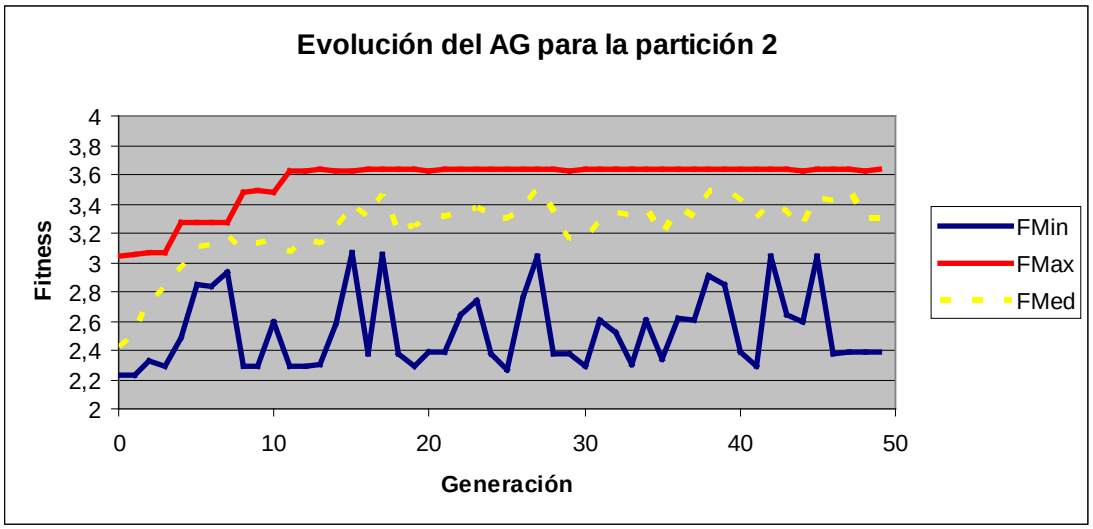


Figura 5.3.3-4: Evolución del AG para la partición 2. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

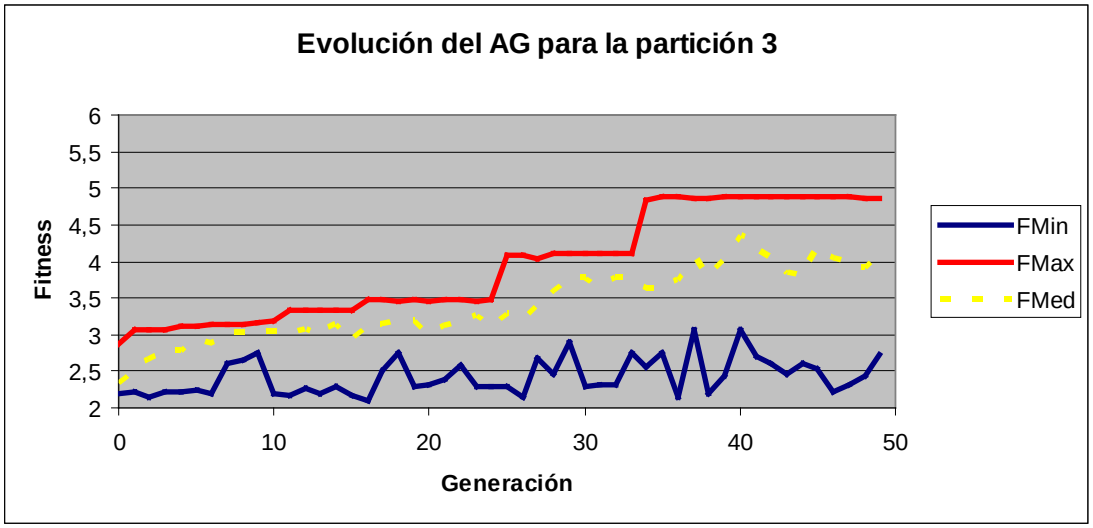


Figura 5.3.3-5: Evolución del AG para la partición 3. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

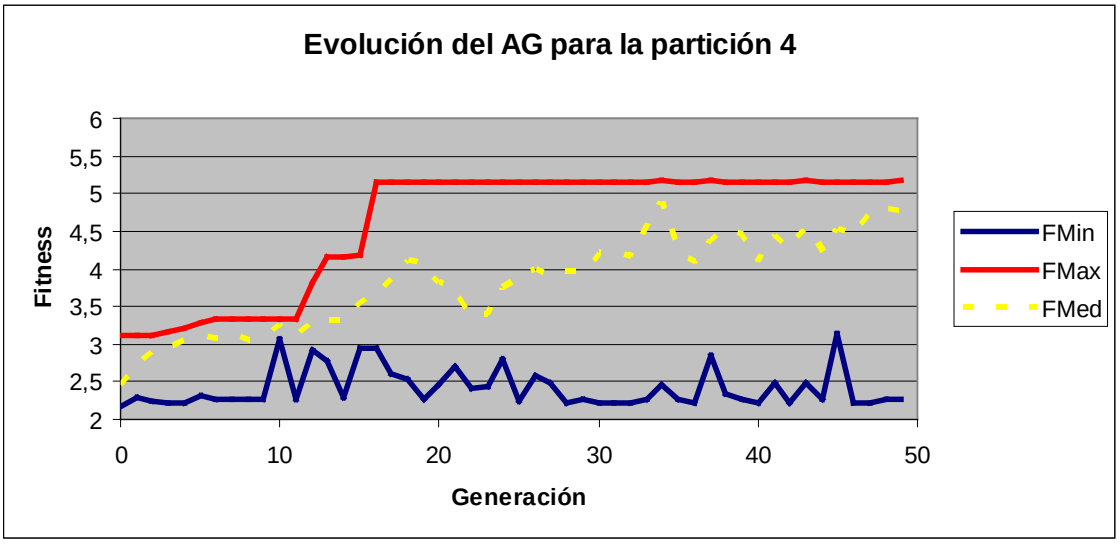


Figura 5.3.3-6: Evolución del AG para la partición 4. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

5.4.- DOMINIO DE CLASIFICACIÓN RECTAS 0.

5.4.1 – Introducción

El dominio de clasificación sintético Rectas 0, es igual que el dominio Rectas 45, con el que hemos experimentado anteriormente (5.3), con la única salvedad de que todos los puntos han sido girados 45º en sentido contrario a las agujas del reloj.

El lector, puede repasar en el capítulo 5.3.1, la manera en la que fue generado el dominio sintético Rectas 45, sobre el que posteriormente sus puntos han sido girados 45º, generando el dominio Rectas 0, sobre el que se va a experimentar en los capítulos posteriores.

La motivación, por la que surge este dominio es la posibilidad de que con una transformación simple, al no ser necesaria la rotación, los datos pueden ser clasificados de forma más correcta.

En la siguiente figura (*figura 5.4.1-1*), se representa un subconjunto de datos de este dominio:

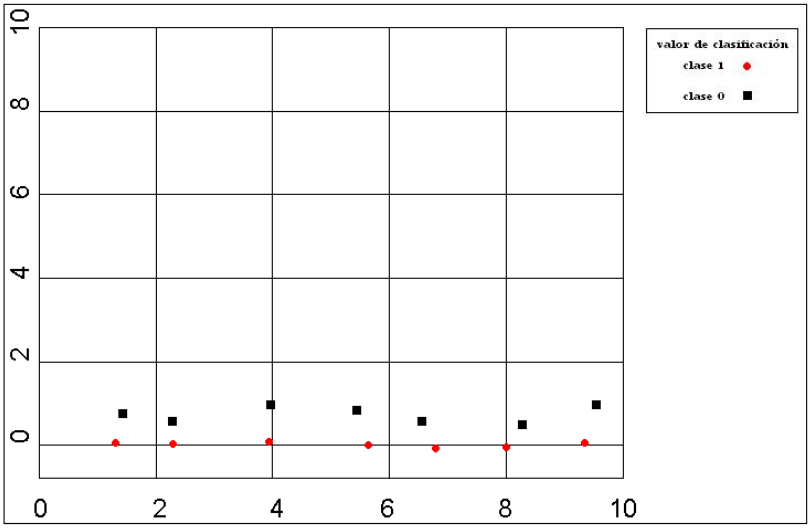


Figura 5.4.1-1: Subconjunto de datos del dominio sintético rectas 0.

Si comparamos este subconjunto de datos con el del dominio rectas 45 (*figura 5.3.1-1*), se puede ver que el dominio rectas 0 requiere solo de escalamiento (matriz diagonal), mientras que el rectas 45 requiere tanto escalamiento como rotación (matriz simétrica)..

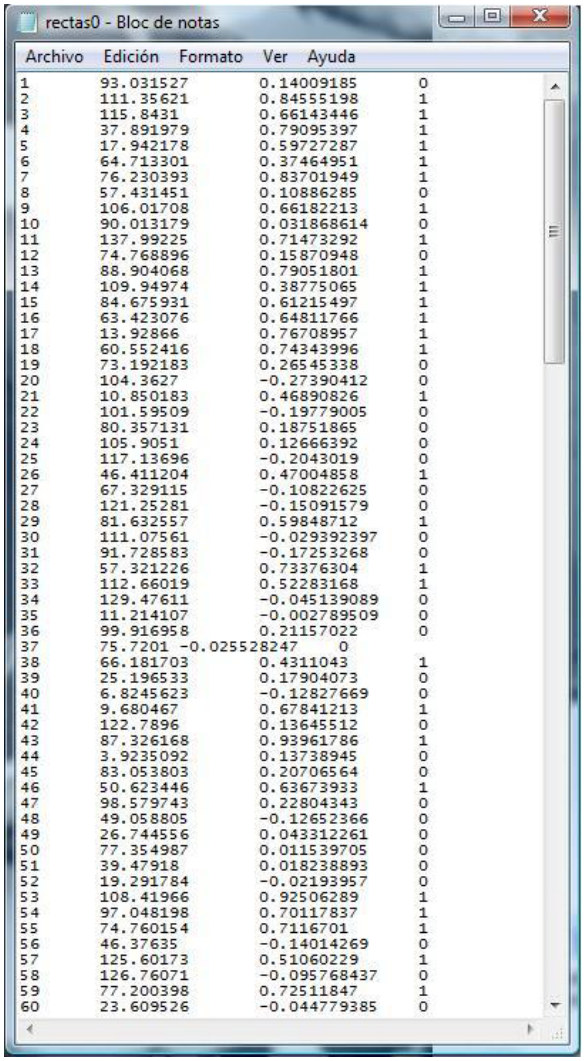


Figura 5.4.1-1: Fragmento del fichero de entrada del dominio Rectas 0.

5.4.2 - Experimentación con matrices de distancias diagonales.

Antes de comenzar con la primera prueba de experimentación sobre el dominio Rectas cero, que al igual que experimentos con dominios anteriores va a consistir en la obtención como solución de una matriz de distancias diagonal, tenemos que tener presente que este dominio es sintético, puesto que está generado artificialmente.

Como ya se ha visto en anteriores pruebas de experimentación (dominio de diabetes) en este proyecto, tanto en soluciones con matrices diagonales como simétricas, realizamos los experimentos calculando la fitness a partir del error de entrenamiento de la red.

En cualquiera de los casos vamos a realizar el estudio de lo óptima o no que es la matriz obtenida utilizando validación cruzada.

La fitness que se va a emplear para hacer evolucionar el algoritmo genético (AG) se obtiene a partir del error de entrenamiento de la red. Una vez que termina de entrenarse se aplica la transformación logarítmica de ese error para obtener la fitness.

En el caso del dominio rectas cero disponemos de 200 patrones, los cuales debido al uso de validación cruzada vamos a dividir en 4 particiones. Por lo tanto vamos a tener en cada partición 50 patrones de test y 150 patrones de entrenamiento para proceder a la validación de cada una de las particiones.

Los parámetros del sistema, para realizar el experimento sobre este dominio con matrices diagonales, son los siguientes:

- La solución es una matriz de distancias diagonal
- Se emplea validación cruzada: 4 particiones.
- Se utilizan el 100% de los patrones del fichero de aprendizaje de cada partición para el entrenamiento de la red, a partir de los cuales se calcula la fitness.
- Número de generaciones: 50
- Tamaño de la población: 15
- Tamaño del torneo: 2
- Elitismo de 1 elemento
- Cruce de dos puntos
- Probabilidad de cruce: 0,7
- Probabilidad de mutación: 0,01
- Conversión binario a real con una precisión de 2 dígitos en la parte entera binaria y 3 en la parte fraccionaria (LongVal=6; Lpunto=3).

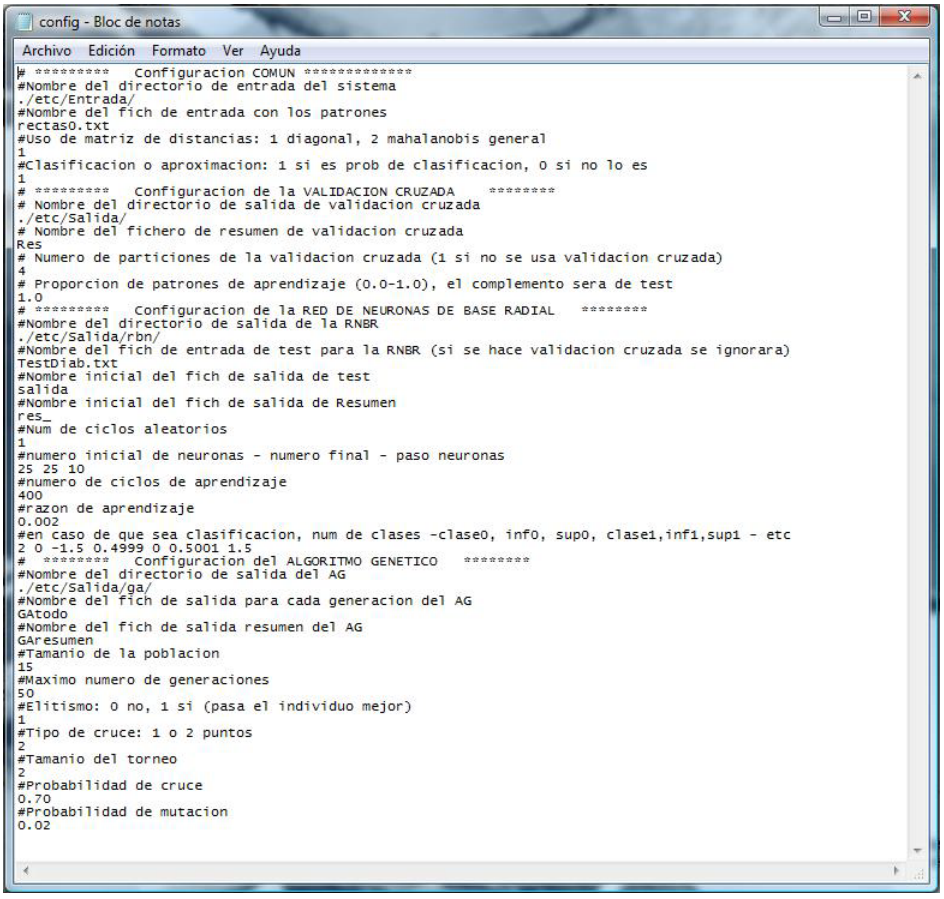


Figura 5.4.2-1: Fichero "config.ini" utilizado para la experimentación con matrices diagonales en el dominio de clasificación de Rectas cero.

Una vez introducidos los parámetros en la aplicación *(Figura 5.4.2-1)* procedamos a estudiar los resultados obtenidos:

Partición	Tasa de aciertos RNBR	Tasa de aciertos del sistema	Matriz solución obtenida
1	0,43137	0,98039	{ 0 0 / 0 -2 }
2	0,37255	1,00000	{ 0 0 / 0 2,625 }
3	0,47059	1,00000	{ 0 0 / 0 2,375 }
4	0,43137	1,00000	{ 0 0 / 0 -1,875 }

Tabla 5.4.2-1: Resultados obtenidos para cada una de las particiones de validación cruzada restringiendo el resultado a matriz diagonal con fitness calculada a partir del error de entrenamiento.

En el gráfico *(Figura 5.4.2-2)* podemos observar representada la tasa de acierto para cada una de las particiones. La última columna de la gráfica se corresponde con la tasa de acierto media:

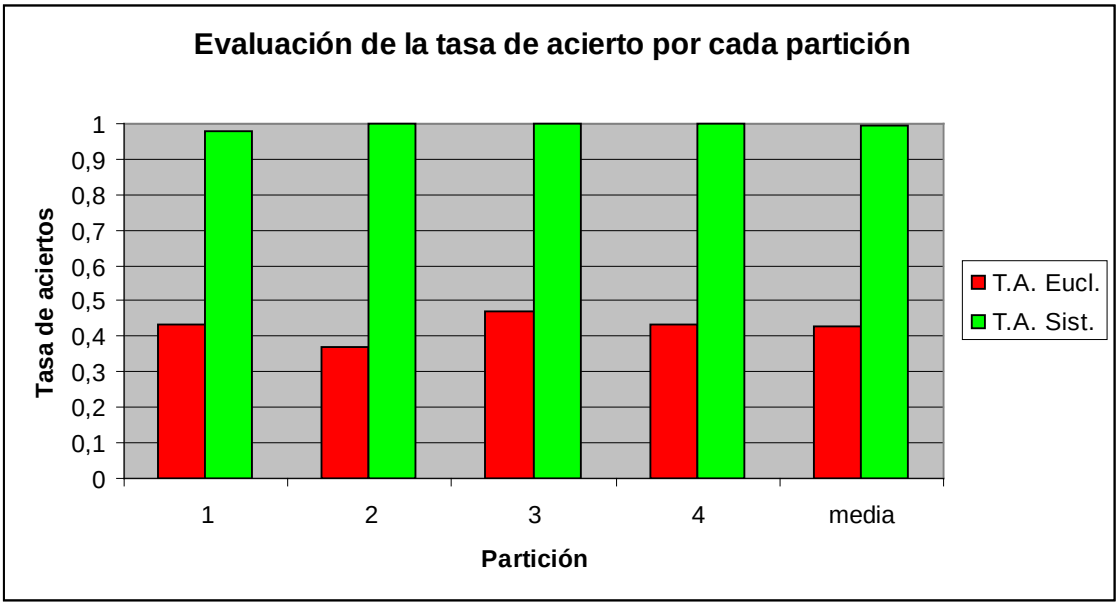


Figura 5.4.2-2: Evaluación de las particiones. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

Dada la tasa de aciertos media obtenida en el sistema (0,9950) y en la RNBR (0,4264) se puede observar como la tasa de aciertos obtenida por el sistema es muy superior a la obtenida por la RNBR.

Por lo tanto se puede decir que en el caso del dominio rectas 0, utilizando matrices diagonales como solución, se obtienen mejores resultados con el sistema, puesto que en las particiones 2, 3 y 4 llega a conseguir el 100% de aciertos.

Atendiendo a la figura de la evaluación de las particiones (figura 5.4.2-2), se puede observar como en la primera partición el sistema la tasa de aciertos supera el 98%, lo que contrasta con los pésimos resultados que obtiene la RNBR que no llega a superar el 45% de aciertos. Como ya se ha dicho, esta situación se repite en las otras tres particiones restantes, obteniendo su mejor resultado para la red en la tercera partición alcanzando un pésimo 47% de aciertos.

En las siguientes gráficas podemos observar la evolución de la fitness para cada una de las particiones:

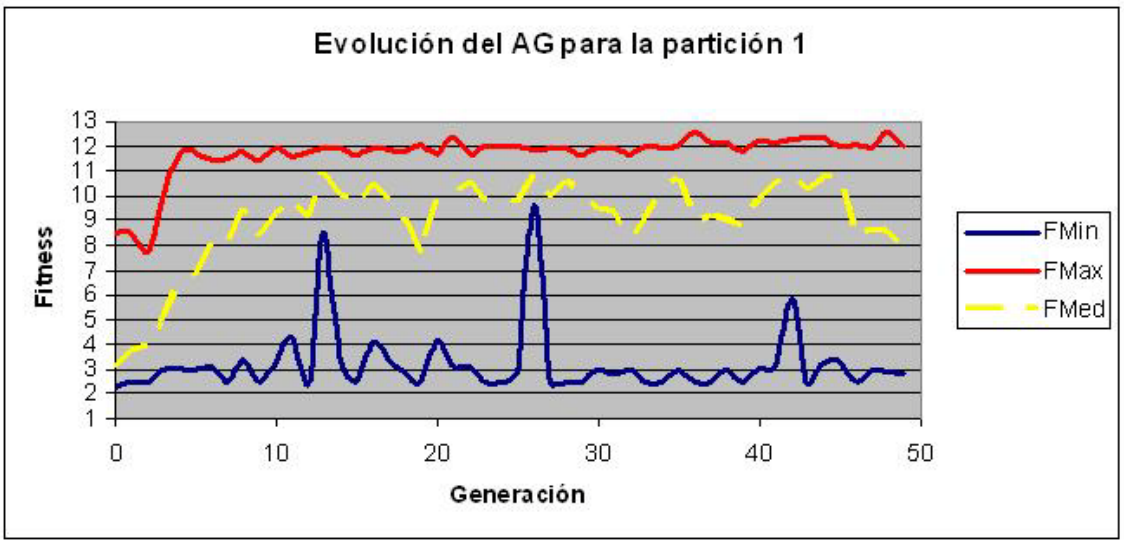


Figura 5.4.2-3: Evolución del AG para la partición 1. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

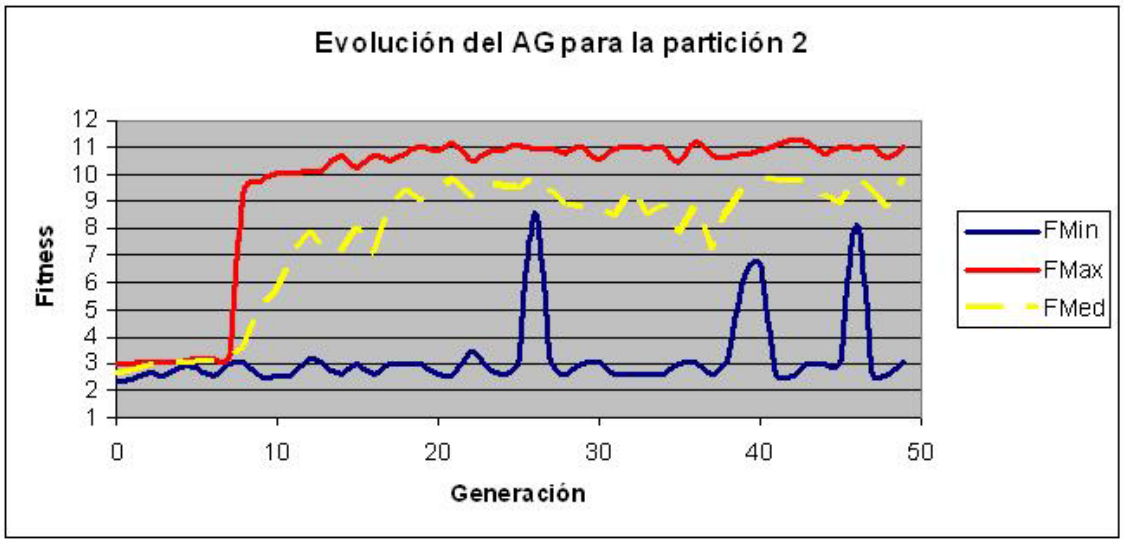


Figura 5.4.2-4: Evolución del AG para la partición 2. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

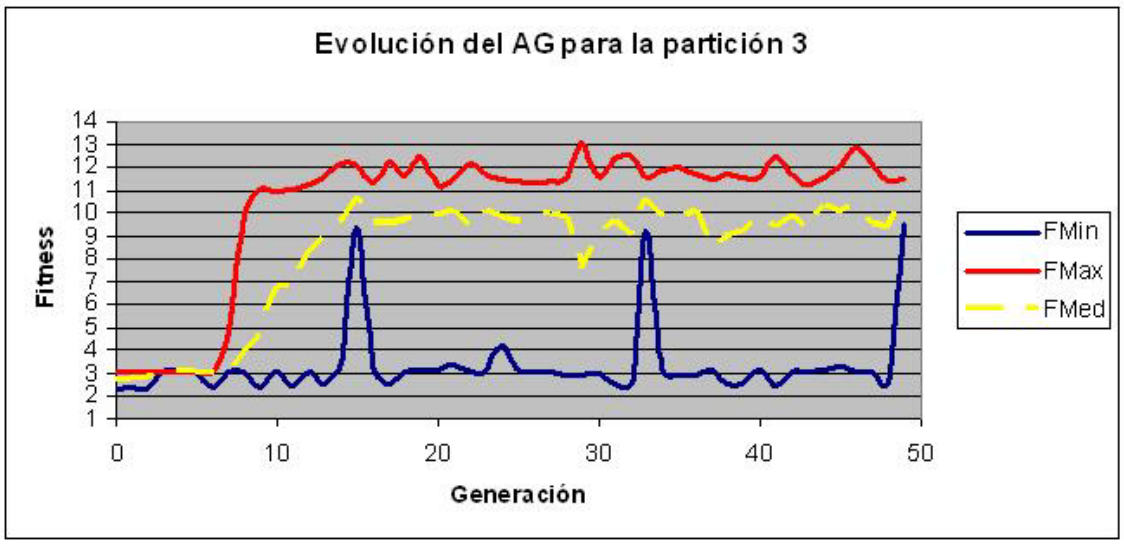


Figura 5.4.2-5: Evolución del AG para la partición 3. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

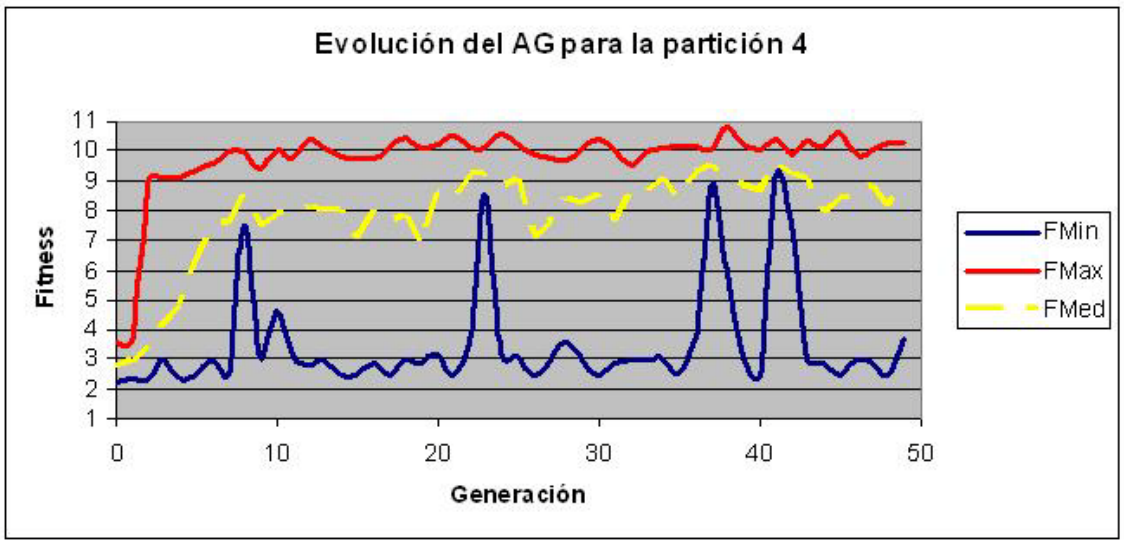


Figura 5.4.2-6: Evolución del AG para la partición 4. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

5.4.3 - Experimentación con matrices de distancias simétricas.

Para la siguiente prueba de experimentación sobre el dominio rectas ruido se van a utilizar matrices simétricas.

Como en el caso anterior de experimentación, para este dominio, se realizará el experimento calculando la fitness a partir del error de entrenamiento de la red y se va a utilizar validación cruzada.

Recordemos que para el dominio de rectas ruido, disponemos de 200 patrones, los cuales debido al uso de validación cruzada vamos a dividir en 4 particiones. Por lo tanto tendremos 50 patrones en cada partición y 150 patrones en la parte de aprendizaje para proceder a la validación de cada una de las particiones.

Los parámetros del sistema son los siguientes:

- La solución es una matriz de distancias simétrica
- Se emplea validación cruzada: 4 particiones.
- Se utilizan el 100% de los patrones del fichero de aprendizaje de cada partición para el entrenamiento de la red, a partir de los cuales se calcula la fitness.
- Número de generaciones: 50
- Tamaño de la población: 15
- Tamaño del torneo: 2
- Elitismo de 1 elemento
- Cruce de dos puntos
- Probabilidad de cruce: 0,7
- Probabilidad de mutación: 0,01
- Conversión binario a real con una precisión de 2 dígitos en la parte entera binaria y 3 en la parte fraccionaria (LongVal=6; Lpunto=3).

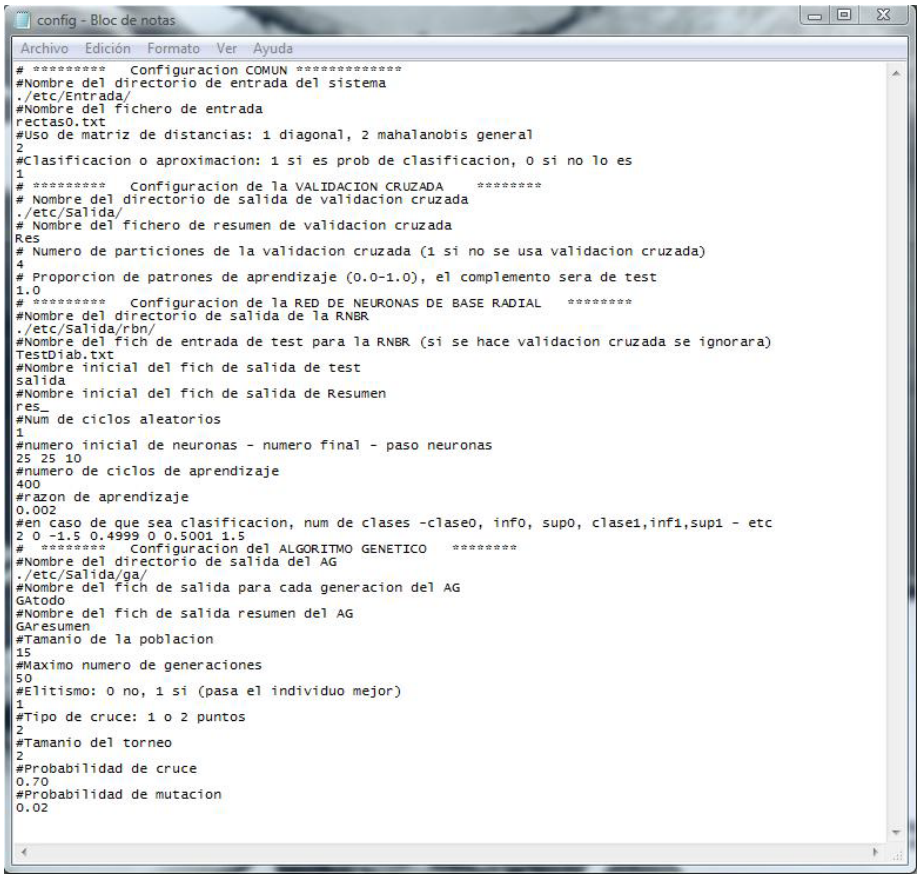


Figura 5.4.3-1: Fichero de configuración para el dominio rectas ruido, utilizando una matriz simétrica como solución.

Una vez introducidos los parámetros en la aplicación *(ver figura 5.4.3.-1)* procedamos a estudiar los resultados obtenidos:

Partición	Tasa de aciertos RNBR	Tasa de aciertos del sistema	Matriz solución obtenida
1	0,43137	0,96078	0 0 0 3,9375
2	0,39216	1,00000	0 0 0 -3,8125
3	0,47059	1,00000	0 0 0 2,75
4	0,45098	1,00000	0 0 0 -3,9375

Tabla 5.4.3-1: Resultados obtenidos para cada una de las particiones de validación cruzada restringiendo el resultado a matriz simétrica con fitness calculada a partir del error de entrenamiento.

En el gráfico *(Figura 5.4.3-2)* podemos observar representada la tasa de acierto para cada una de las particiones. Las últimas columnas de la gráfica se corresponden con la tasa de acierto media:

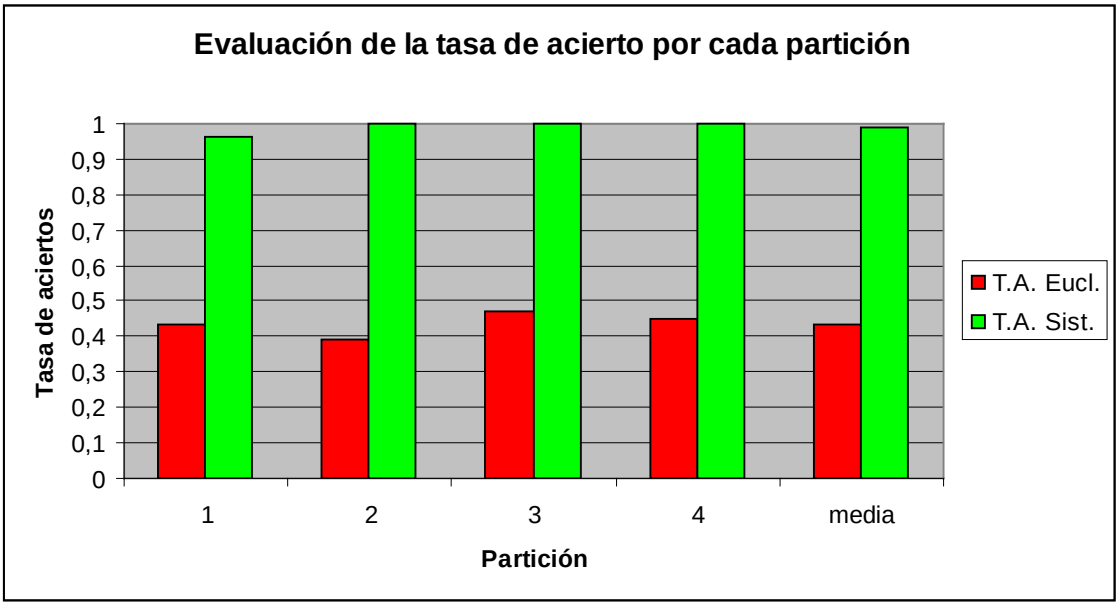


Figura 5.4.3-2: Evaluación de las particiones. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

Dada la tasa de aciertos media obtenida en el sistema (0,9901) y en la RNBR (0,4264), se observa que el sistema obtiene una mayor tasa de aciertos que la red, llegando a doblar su porcentaje de aciertos.

Los resultados obtenidos empleando matriz simétrica como solución, se asemejan a los obtenidos en la experimentación anterior con el dominio rectas 0 empleando una matriz diagonal.

Al igual que en el experimento con matriz diagonal, la segunda, tercera y cuarta partición alcanza el 100% de aciertos, siendo los porcentajes para estas mismas particiones obtenidos por la RNBR del 39%,47% y 45% respectivamente.

Por tanto, se puede concluir que para el dominio Rectas 0, restringiendo la matriz solución a ser simétrica, nuestro sistema obtiene resultados más óptimos que una RNBR (un 42% más de aciertos), alcanzado un 99% de media.

En las siguientes gráficas podemos observar la evolución de la fitness para cada una de las particiones:

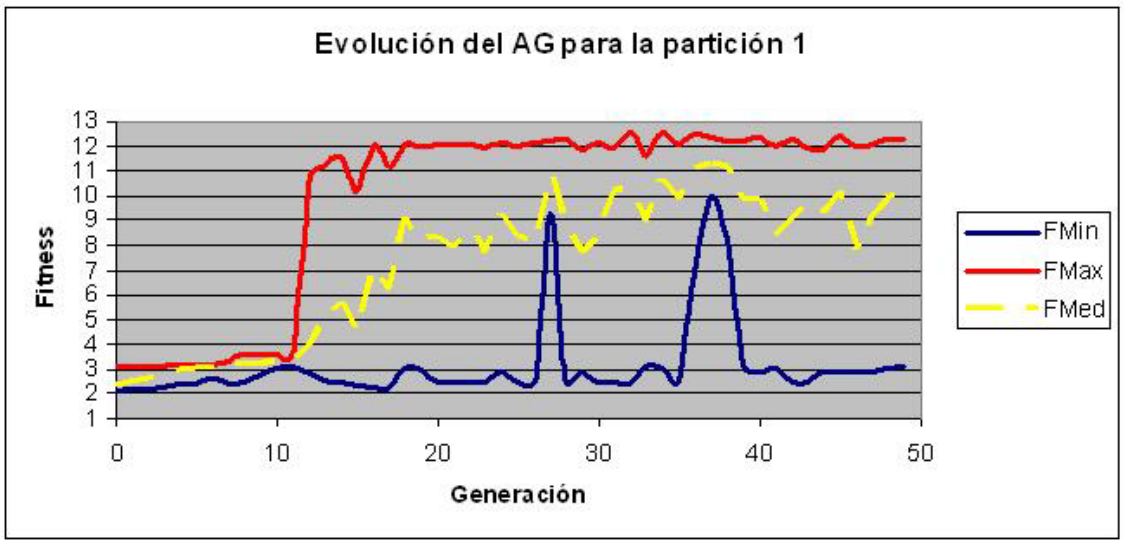


Figura 5.4.3-3: Evolución del AG para la partición 1. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

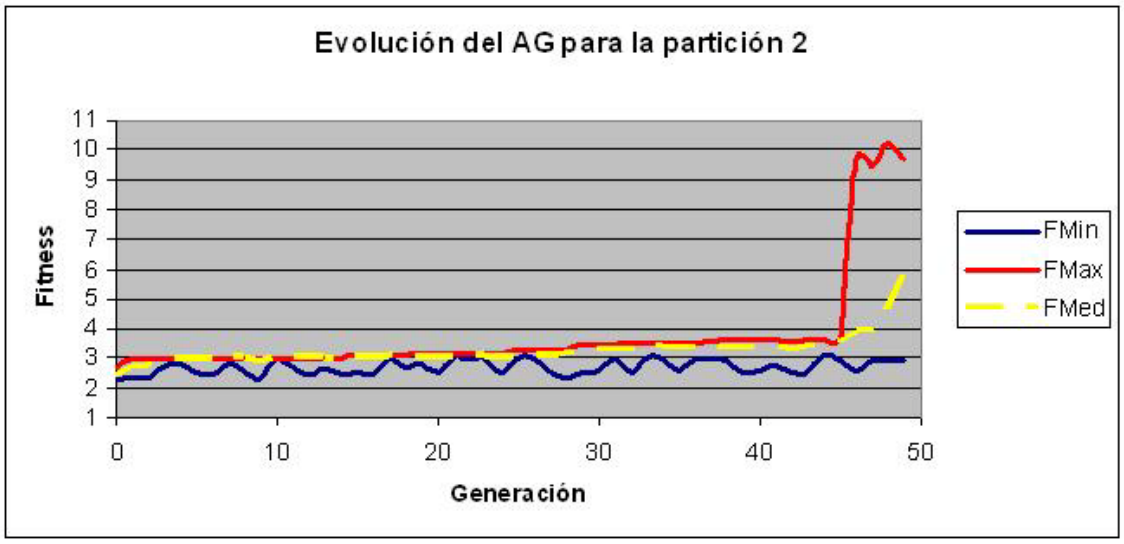


Figura 5.4.3-4: Evolución del AG para la partición 2. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

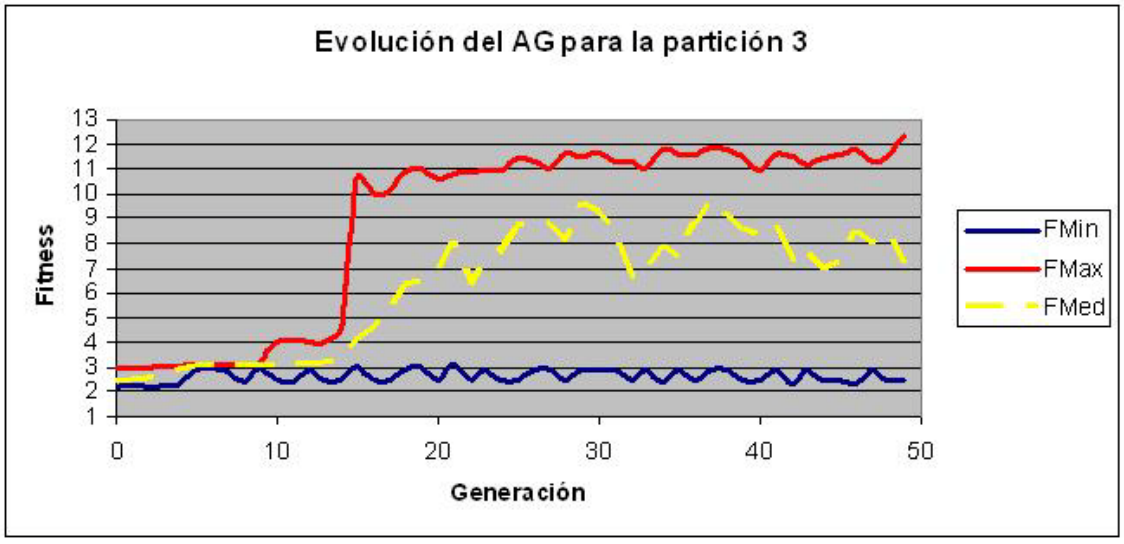


Figura 5.4.3-5: Evolución del AG para la partición 3. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

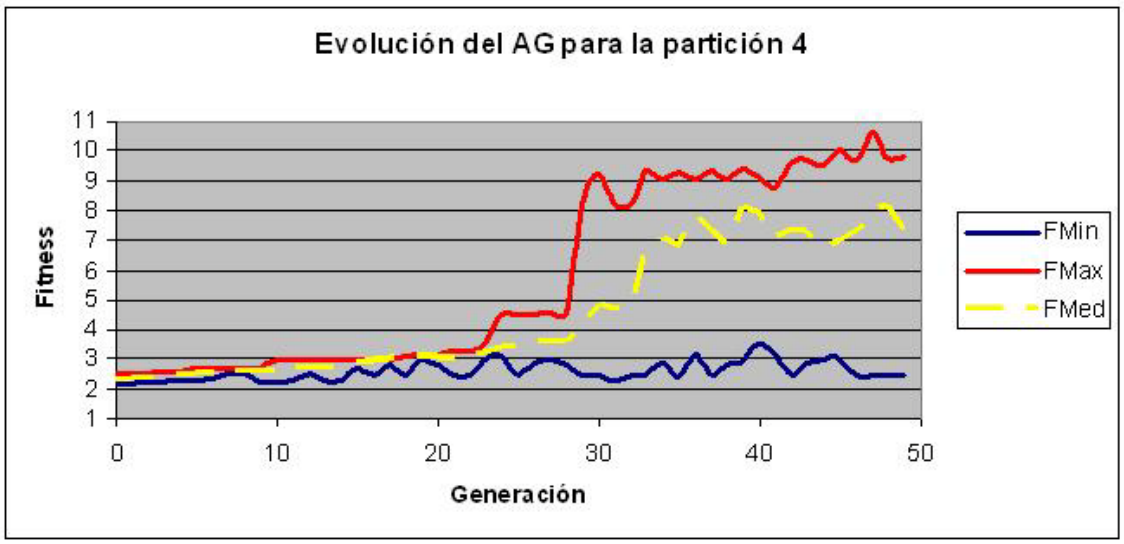


Figura 5.4.3-6: Evolución del AG para la partición 4. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

5.5.- DOMINIO DE CLASIFICACIÓN SINTÉTICO ALEATORIO

5.5.1 – Introducción.

El dominio de clasificación sintético aleatorio, ha sido generado de manera artificial, por lo que no está basado en datos reales como ocurría en el dominio de la diabetes (5.1).

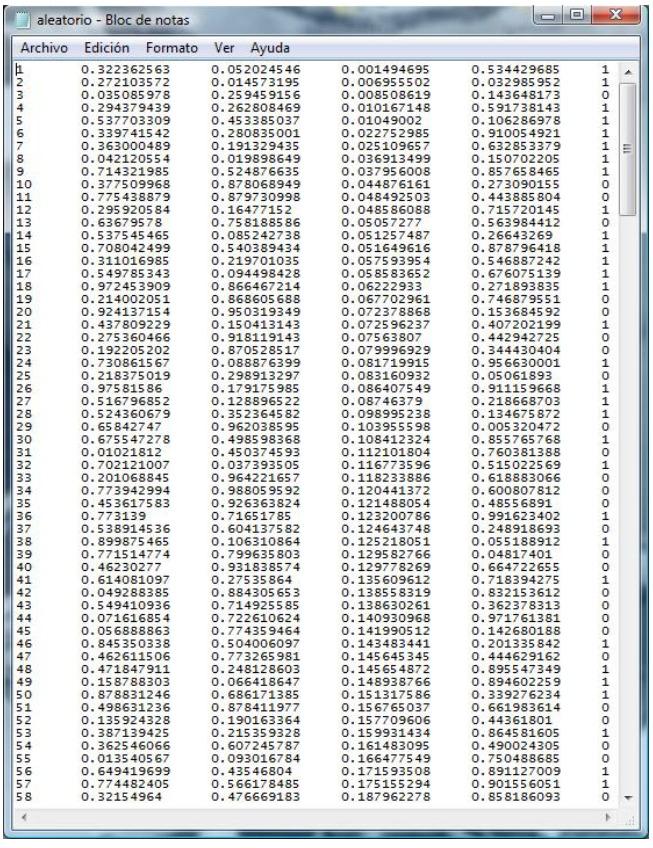
Es un dominio de clasificación (0 ó 1) que viene determinado por valores reales distribuidos en cuatro atributos (X_1, X_2, X_3, X_4).

Los ejemplos han sido generados aleatoriamente utilizando una distribución uniforme en el intervalo [0,1] para los atributos X_1 y X_2 y un intervalo [0,100] para los atributos X_3 y X_4.

Si X_1 es menos que X_2, podemos clasificar el ejemplo como de clase 1, en caso contrario será de clase 0. Por tanto, se deduce que los atributos X_3 y X_4 son irrelevantes.

Como el rango de atributos irrelevantes es muy grande, la exactitud de clasificación de KNN es bastante mala, alrededor del 50%.

A continuación (*figura 5.5.1.1-2*), se muestra un fragmento del fichero de entrada del dominio aleatorio.



1	0.322362563	0.052024546	0.001494695	0.534429685	1
2	0.272103572	0.014573195	0.006955502	0.032985952	1
3	0.035085978	0.259459156	0.008508619	0.143648173	0
4	0.294379439	0.262808469	0.010167148	0.591738143	1
5	0.537703309	0.453385037	0.01049002	0.106286978	1
6	0.339741542	0.280835001	0.022752985	0.910054921	1
7	0.363000489	0.191329435	0.025109657	0.632853379	1
8	0.042120554	0.019898649	0.036913499	0.150702205	1
9	0.714321985	0.524876635	0.037956008	0.857658465	1
10	0.377509968	0.878068949	0.044876161	0.273090155	0
11	0.775438879	0.879730998	0.048492503	0.443885804	0
12	0.295920584	0.16477152	0.048586088	0.715720145	1
13	0.63679578	0.758188586	0.05057277	0.563984412	0
14	0.537545465	0.085242738	0.051257487	0.26643269	1
15	0.708042499	0.540389434	0.051649616	0.878796418	1
16	0.311016985	0.219701035	0.057593954	0.546887242	1
17	0.549785343	0.094498428	0.058583652	0.676075139	1
18	0.972453909	0.866467214	0.06222933	0.271893835	1
19	0.214002051	0.868605688	0.067702961	0.746879551	0
20	0.924137154	0.950319349	0.072378868	0.153684592	0
21	0.437809229	0.150413143	0.072596237	0.407202199	1
22	0.275360466	0.918119143	0.07563807	0.442942725	0
23	0.192205202	0.870528517	0.079996929	0.344430404	0
24	0.730861567	0.088876399	0.081719915	0.956630001	1
25	0.218375019	0.298913297	0.083160932	0.05061893	0
26	0.97581586	0.179175985	0.086407549	0.911159668	1
27	0.516796852	0.128896522	0.08746379	0.218668703	1
28	0.524360679	0.352364582	0.098995238	0.134675872	1
29	0.65842747	0.962038595	0.103955598	0.005320472	0
30	0.675547278	0.498598368	0.108412324	0.855765768	1
31	0.01021812	0.450374593	0.112101804	0.760381388	0
32	0.702121007	0.037393505	0.116773596	0.515022569	1
33	0.201068845	0.964221657	0.118233886	0.618883066	0
34	0.773942994	0.988059592	0.120441372	0.600807812	0
35	0.453617583	0.926363824	0.121488054	0.48556891	0
36	0.773139	0.71651785	0.123200786	0.991623402	1
37	0.538914536	0.604137582	0.124643748	0.248918693	0
38	0.899875465	0.106310864	0.125218051	0.055188912	1
39	0.771514774	0.799635803	0.129582766	0.04817401	0
40	0.46230277	0.931838574	0.129778269	0.664722655	0
41	0.614081097	0.27535864	0.135609612	0.718394275	1
42	0.049288385	0.884305653	0.138558319	0.832153612	0
43	0.549410936	0.714925585	0.138630261	0.362378313	0
44	0.071616854	0.722610624	0.140930968	0.971761381	0
45	0.056888863	0.774359464	0.141990512	0.142680188	0
46	0.845350328	0.504006097	0.143483441	0.201335842	1
47	0.462611506	0.773265981	0.145645345	0.44462916	0
48	0.471847911	0.248128603	0.145654872	0.895547349	1
49	0.158788303	0.066418647	0.148938766	0.894602259	1
50	0.878831246	0.686171385	0.151317586	0.339276234	1
51	0.498631236	0.878411977	0.156765037	0.661983614	0
52	0.135924328	0.190163364	0.157709606	0.44361801	0
53	0.387139425	0.215359328	0.159931434	0.864581605	1
54	0.362546066	0.607245787	0.161483095	0.490024305	0
55	0.013540567	0.093016784	0.166477549	0.750488685	0
56	0.649419699	0.43546804	0.171593508	0.891127009	1
57	0.774482405	0.566178485	0.175155294	0.901556051	1
58	0.32154964	0.476669183	0.187962278	0.858186093	0

Figura 5.5.1-2: Fragmento del fichero de entrada del dominio de clasificación sintético aleatorio.

El dominio de clasificación sintético aleatorio tiene cuatro atributos o instancias, por ello las matrices obtenidas como solución durante la experimentación serán de dimensión 4.

5.5.2 - Experimentación con matrices de distancias diagonales.

A continuación veremos los resultados obtenidos de realizar para este dominio la primera prueba de experimentación, es decir obligando a que la matriz solución obtenida sólo pueda ser diagonal.

Como ya sabemos de la experimentación con dominios anteriores, tanto en soluciones con matrices diagonales como simétricas, realizaremos los experimentos calculando la fitness a partir del error de entrenamiento de la red.

En cualquiera de los casos vamos a realizar el estudio de la matriz obtenida comparando la tasa de aciertos obtenida en los conjuntos de test de cada una de las particiones de la validación cruzada.

La fitness que se va a emplear para hacer evolucionar el algoritmo genético (AG) se obtiene a partir del error de entrenamiento de la red. Una vez que termina de entrenarse se aplica la transformación logarítmica de ese error para obtener la fitness.

En el dominio de clasificación sintético aleatorio disponemos de 300 patrones, los cuales debido al uso de validación cruzada vamos a dividir en 4 particiones. Por lo tanto vamos a tener 75 patrones en cada partición y 225 patrones en la parte de aprendizaje para proceder a la validación de cada una de las particiones.

Los parámetros del sistema son los siguientes:

- La solución es una matriz de distancias diagonal
- Se emplea validación cruzada: 4 particiones.
- Se utilizan el 100% de los patrones del fichero de aprendizaje de cada partición para el entrenamiento de la red, a partir de los cuales se calcula la fitness.
- Número de generaciones: 50
- Tamaño de la población: 15
- Tamaño del torneo: 2
- Elitismo de 1 elemento
- Cruce de dos puntos
- Probabilidad de cruce: 0,7
- Probabilidad de mutación: 0,01
- Conversión binario a real con una precisión de 2 dígitos en la parte entera binaria y 3 en la parte fraccionaria (LongVal=6; Lpunto=3).

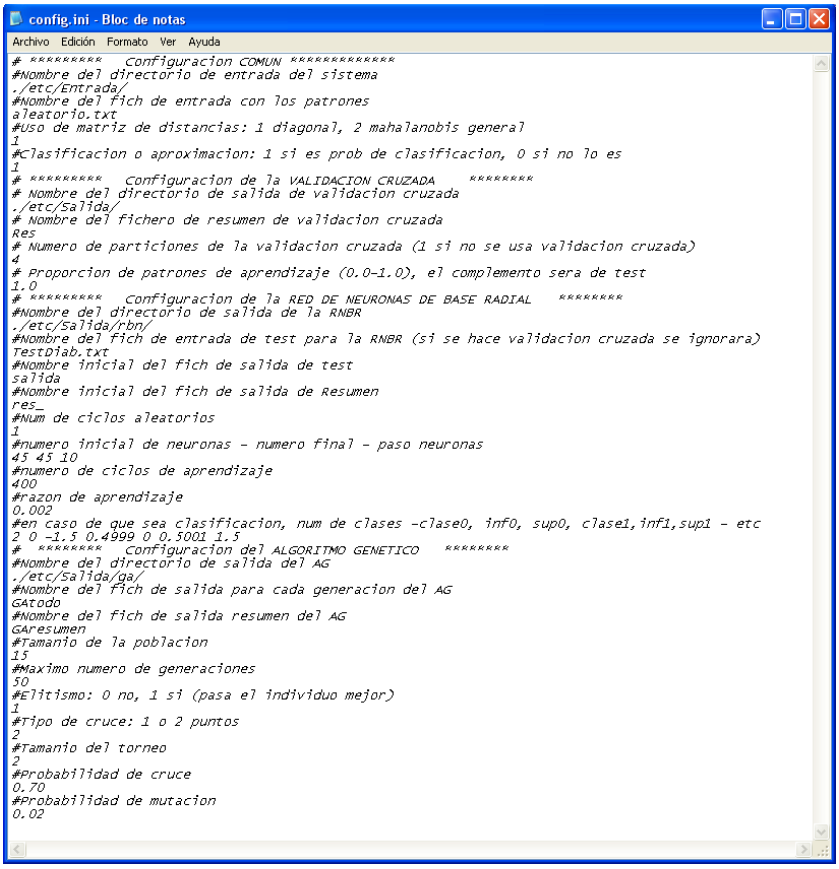


Figura 5.5.2-1: Fichero de configuración utilizado para la experimentación con el dominio sintético aleatorio limitando la solución a ser una matriz diagonal.

Una vez introducidos los parámetros en la aplicación *(figura 5.5.2-1)* procedemos a estudiar los resultados obtenidos:

Partición	Fitness con RNBR	Fitness con el sistema	Matriz solución obtenida
1	0,88158	0,98684	-3,6875 0 0 0 0 3,375 0 0 0 0 0 0 0 0 0 -0,1875
2	0,96053	0,96053	3,9375 0 0 0 0 3,375 0 0 0 0 -0,0625 0 0 0 0 0
3	0,90789	0,97368	-3,9375 0 0 0 0 3 0 0 0 0 0,0625 0 0 0 0 -0,3125
4	0,86842	0,98684	3,9375 0 0 0 0 -3,3125 0 0 0 0 0,3125 0 0 0 0 -0,125

Tabla 5.5.2-1: Resultados obtenidos para cada una de las particiones de validación cruzada restringiendo el resultado a matriz diagonal con fitness calculada a partir del error de entrenamiento.

En el gráfico *(figura 5.5.2-2)* podemos observar representada la tasa de acierto para cada una de las particiones. La última columna de la gráfica se corresponde con la tasa de acierto media:

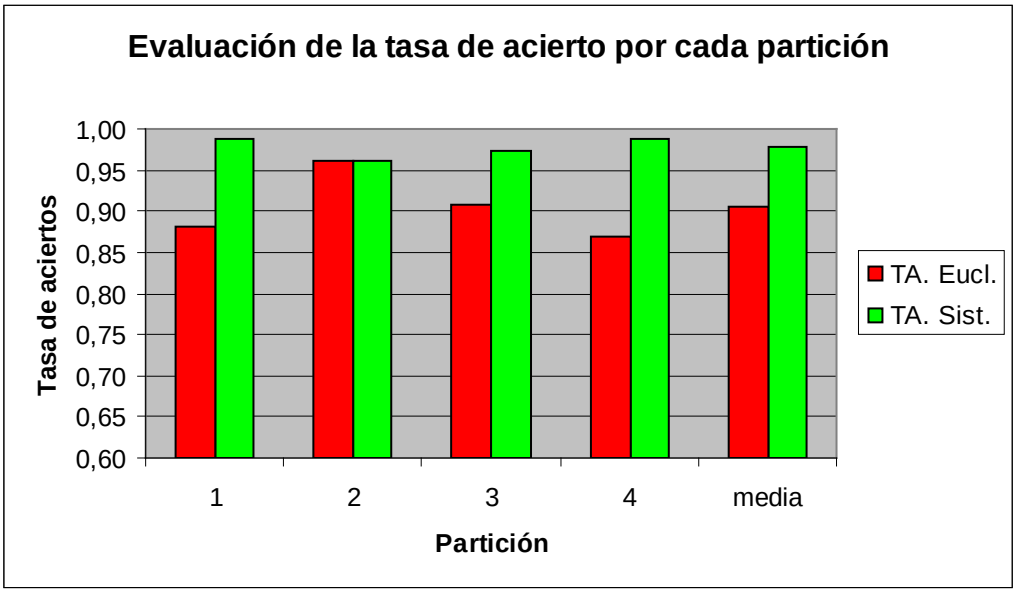


Figura 5.5.2-2: Evaluación de las particiones. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

Dada la tasa de aciertos media obtenida en el sistema *(0,9769)* y en la RNBR *(0,9046)* se puede observar como la tasa de aciertos obtenida por nuestro sistema es superior comparada con la obtenida por la Red Neuronal de Base Radial.

A excepción de la partición 2, en la cual las tasas de acierto son idénticas (0,9605), en el resto de particiones la tasa de acierto es bastante superior utilizando nuestro sistema que utilizando una RNBR clásica, por lo que se puede decir que en el caso del dominio sintético aleatorio con matriz obtenida como solución diagonal, nuestro sistema es mas óptimo que una red neuronal, llegando a obtener de medio un 98% de aciertos.

En las siguientes gráficas podemos observar la evolución de la fitness para cada una de las particiones:

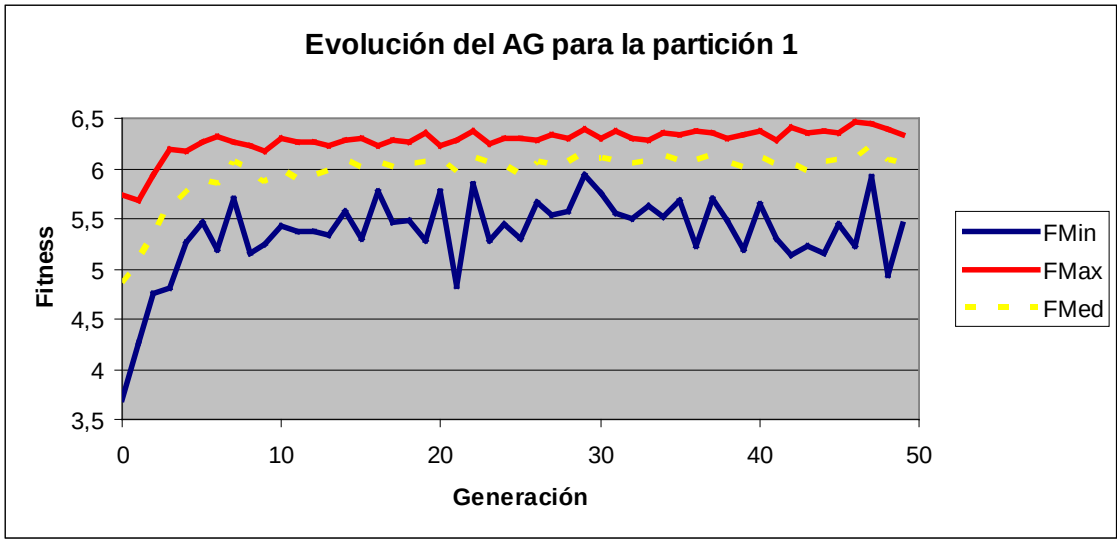


Figura 5.5.2-3: Evolución del AG para la partición 1. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

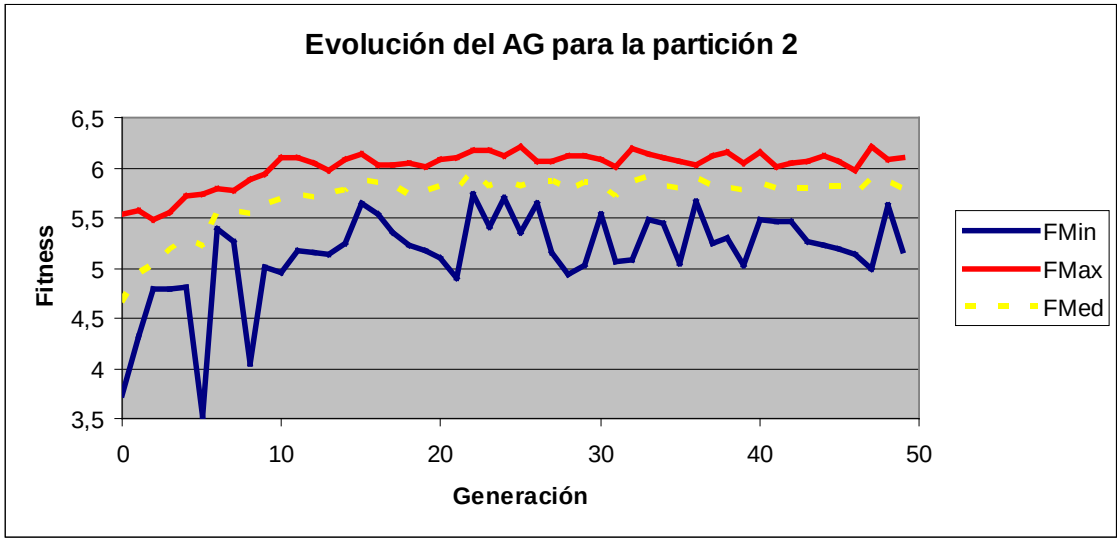


Figura 5.5.2-4: Evolución del AG para la partición 2. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

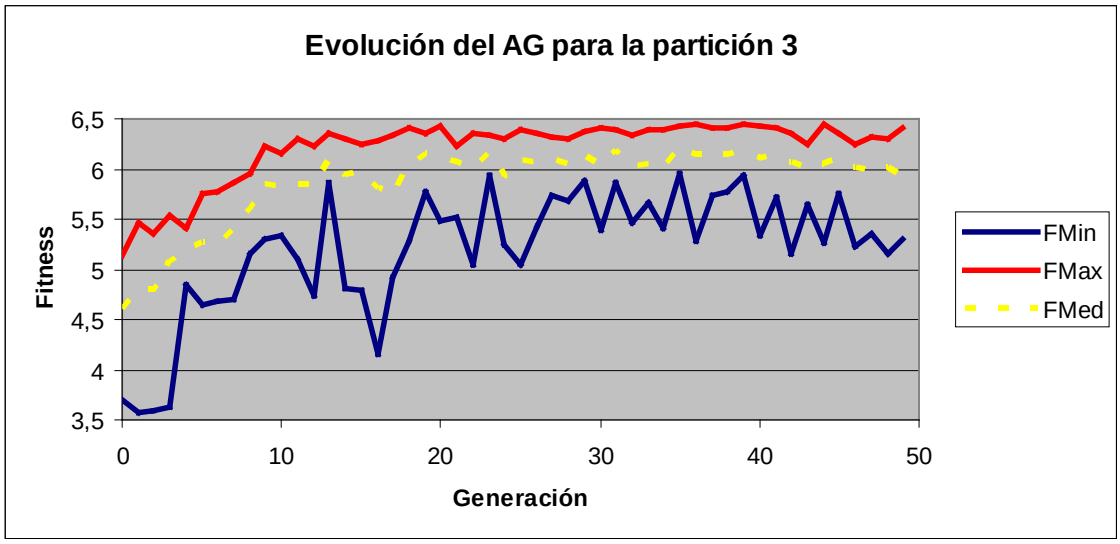


Figura 5.5.2-5: Evolución del AG para la partición 3. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

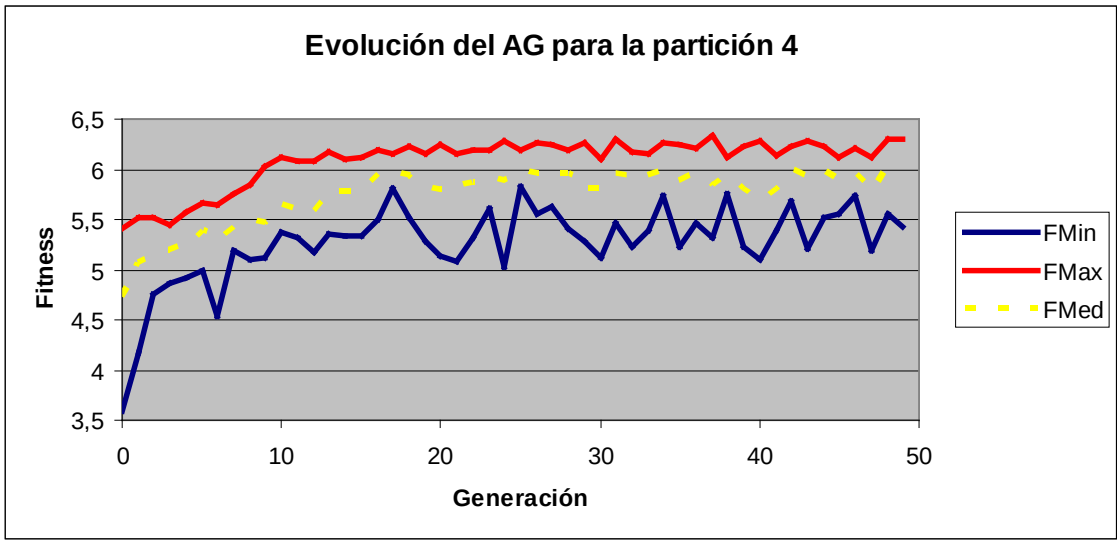


Figura 5.5.2-6: Evolución del AG para la partición 4. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

5.5.3 - Experimentación con matrices de distancias simétricas.

Para la siguiente prueba de experimentación sobre el dominio aleatorio simétrico se van a utilizar matrices simétricas.

Como en el caso anterior de experimentación en este dominio, con matrices diagonales, se realizará el experimento calculando la fitness a partir del error de entrenamiento de la red y se va a utilizar validación cruzada.

Recordemos que, para el dominio sintético aleatorio, disponemos de 300 patrones, los cuales debido al uso de validación cruzada vamos a dividir en 4 particiones. Por lo tanto vamos a tener 75 patrones en cada partición y 225 patrones en la parte de aprendizaje para proceder a la validación de cada una de las particiones.

Los parámetros del sistema son los siguientes:

- La solución es una matriz de distancias simétrica
- Se emplea validación cruzada: 4 particiones.
- Se utilizan el 100% de los patrones del fichero de aprendizaje de cada partición para el entrenamiento de la red, a partir de los cuales se calcula la fitness.
- Número de generaciones: 50
- Tamaño de la población: 15
- Tamaño del torneo: 2
- Elitismo de 1 elemento
- Cruce de dos puntos
- Probabilidad de cruce: 0,7
- Probabilidad de mutación: 0,01

- Conversión binario a real con una precisión de 2 dígitos en la parte entera binaria y 3 en la parte fraccionaria (LongVal=6; Lpunto=3).

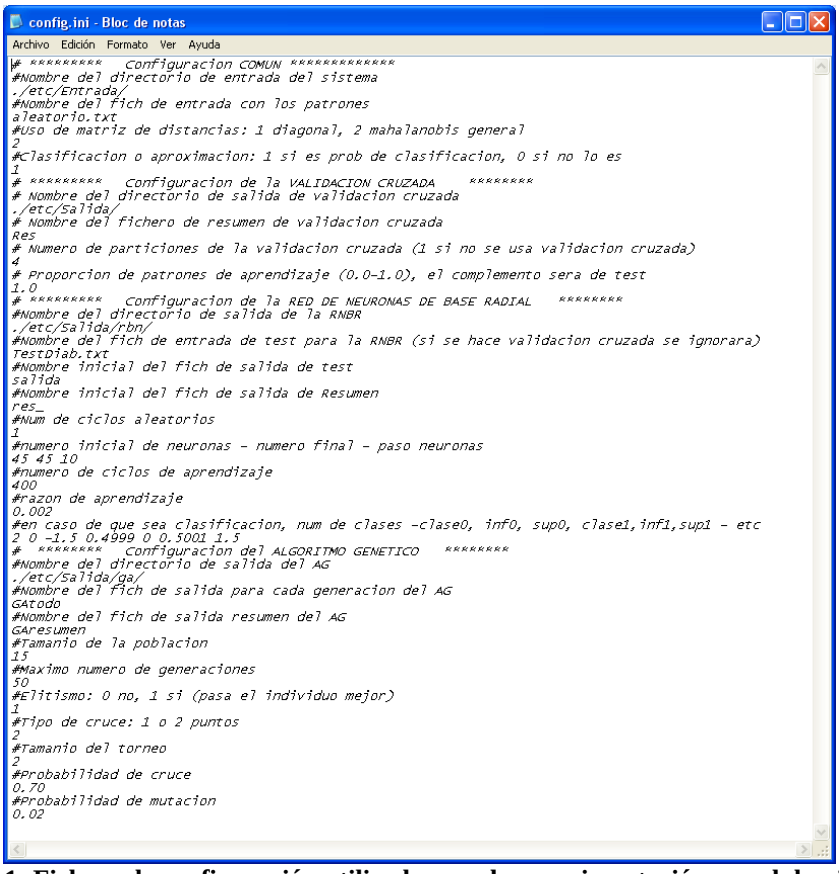


Figura 5.5.3-1: Fichero de configuración utilizado para la experimentación con el dominio sintético aleatorio limitando la solución a ser una matriz simétrica.

Una vez introducidos los parámetros en la aplicación *(figura 5.5.3-1)* procedamos a estudiar los resultados obtenidos:

Partición	Tasa de aciertos RNBR	Tasa de aciertos del sistema	Matriz solución obtenida
1	0,89474	0,60526	0,3125 -0,125 -3,5625 1,125 -0,125 0,625 3,25 -0,75 -3,5625 3,25 -1,875 0,625 1,125 -0,75 0,625 -0,6875
2	0,97368	0,97368	-3,5625 3,8125 -0,25 -0,1875 3,8125 -3,9375 -0,1875 -0,75 -0,25 -0,1875 0,4375 0,125 -0,1875 0 0,125 0,125
3	0,92105	0,90789	0 -0,0625 -2,25 0 -0,0625 0,625 1,875 -0,5625 -2,25 1,875 -0,6875 0,0625 0 -0,5625 0,0625 -0,4375
4	0,84211	0,84211	2,6875 -2,875 -0,25 -0,25 -2,875 2,9375 -0,25 -0,625 -0,25 -0,25 -0,875 0,625 -2,875 -0,625 0,625 -0,0625

Tabla 5.5.3-1: Resultados obtenidos para cada una de las particiones de validación cruzada restringiendo el resultado a matriz simétrica con fitness calculada a partir del error de entrenamiento para el dominio sintético aleatorio.

En el gráfico *(ver figura 5.5.3-2)* podemos observar representada la tasa de acierto para cada una de las particiones. Las últimas columnas de la gráfica se corresponden con la tasa de acierto media:

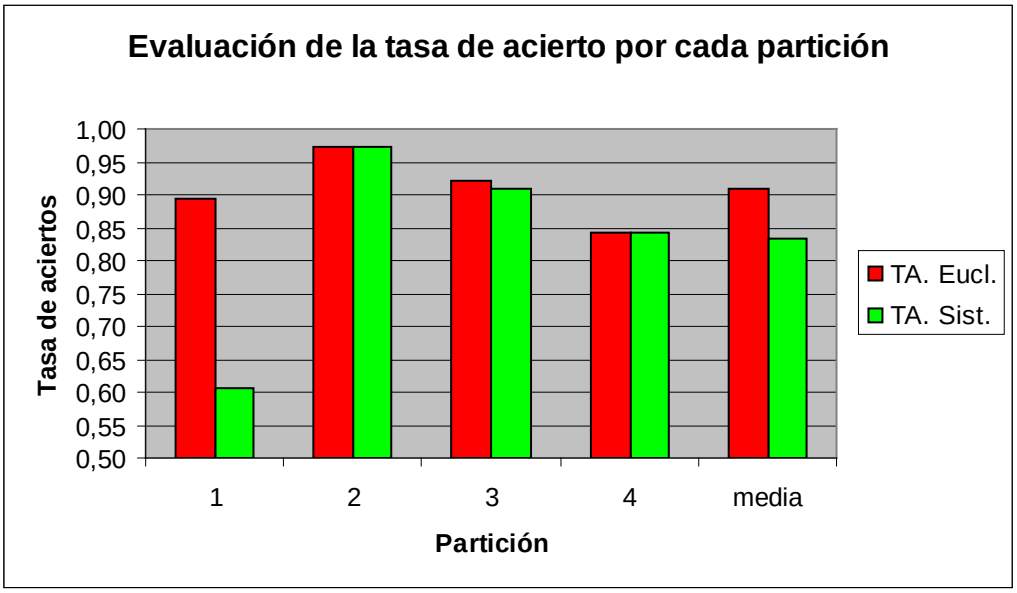


Figura 5.5.3-2: Evaluación de las particiones. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

Dada la tasa de aciertos media obtenida en el sistema *(0,8322)* y en la RNBR *(0,9078)* se puede observar como los resultados empeoran en relación a la utilización de matrices diagonales, siendo la tasa de aciertos del sistema muy inferior a la utilización de una red neuronal de base radial clásica.

Esto queda de manifiesto sobre todo en los resultados obtenidos por la tasa de aciertos del sistema en la primera partición (0,6052), donde es claramente muy inferior a la de una RNBR (0,8947) llegando a existir una diferencia de hasta un 30% menos.

En el resto de particiones, la tasa de aciertos prácticamente se iguala (0,98 y 0,84 obtenido en la segunda y cuarta partición tanto por el sistema como por la RNBR, siendo la tasa de aciertos del sistema ligeramente inferior a la de la RNBR en la tercera partición, 0,90 del sistema frente a un 0,92 de la red), pero debido a la gran diferencia existente en la primera partición, la media de la tasa de aciertos de las cuatro particiones es un 10% inferior en nuestro sistema que utilizando una RNBR.

Se puede incluir, que en el caso del dominio sintético aleatorio, con matrices simétricas, se obtiene un mayor número de aciertos a través de una red neuronal clásica que a través del sistema empleado en la experimentación.

En las siguientes gráficas podemos observar la evolución de la fitness para cada una de las particiones:

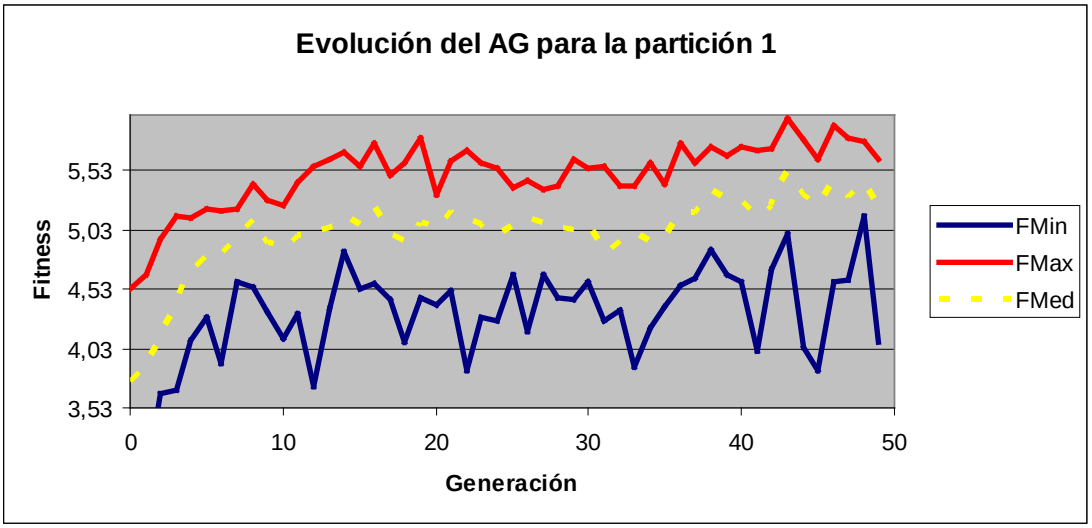


Figura 5.5.3-3: Evolución del AG para la partición 1. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

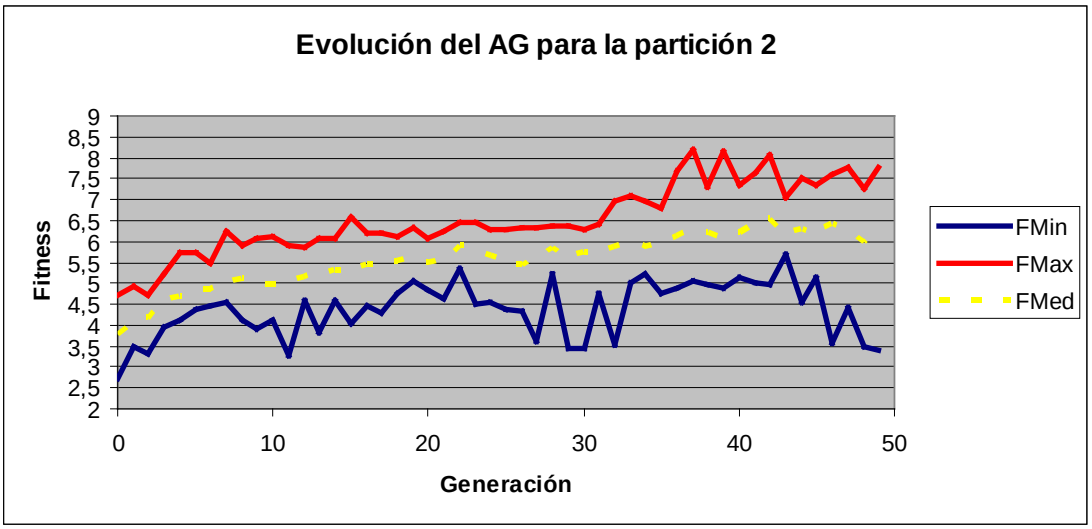


Figura 5.5.3-4: Evolución del AG para la partición 2. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

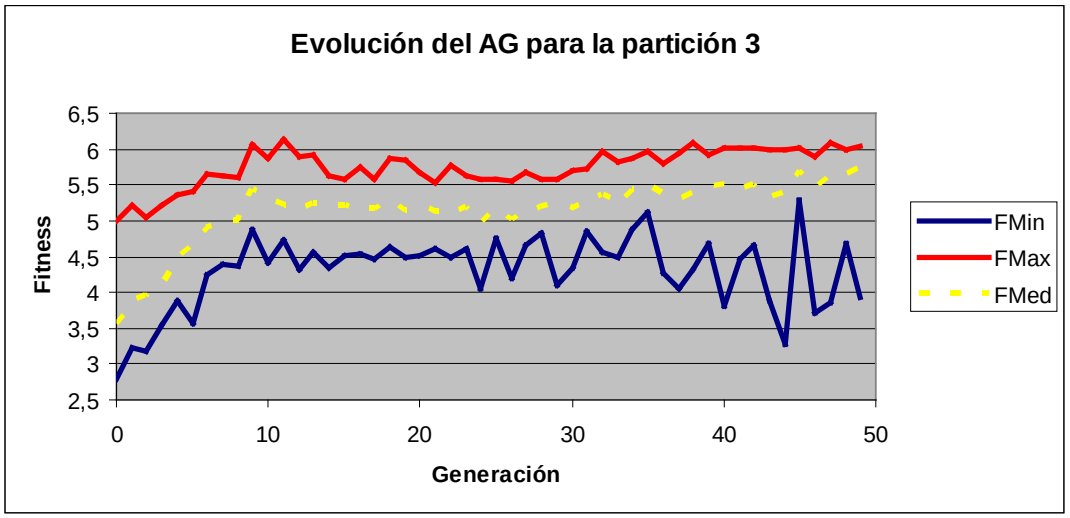


Figura 5.5.3-5: Evolución del AG para la partición 3. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

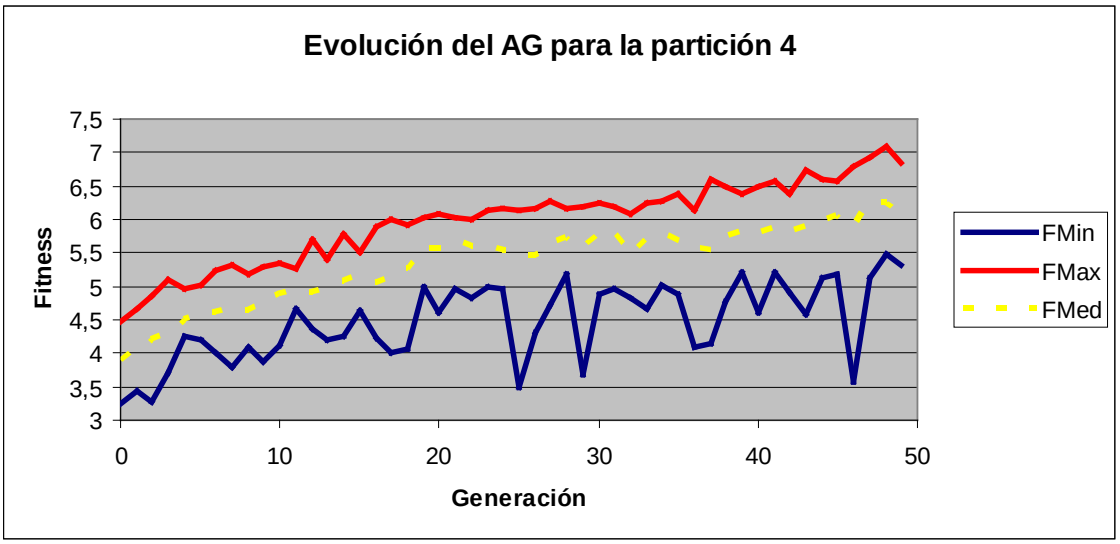


Figura 5.5.3-6: Evolución del AG para la partición 4. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

5.6.- DOMINIO DE CLASIFICACIÓN SINTÉTICO ALEATORIO GIRADO

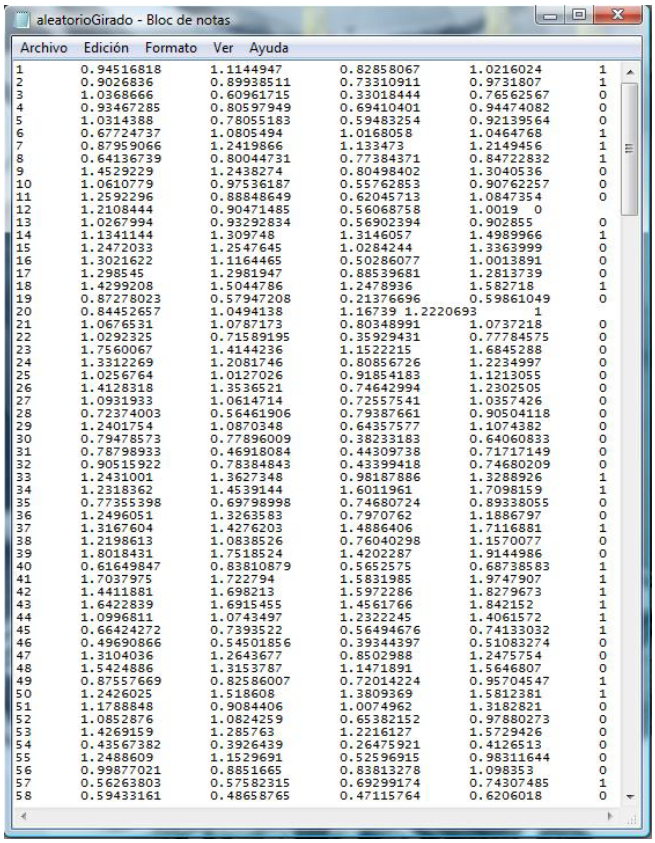
5.6.1 – Introducción.

El dominio de clasificación sintético aleatorio girado, es igual que el dominio aleatorio normal con el que hemos experimentado en el capítulo anterior (5.5).

Para generarlo se ha aplicado al dominio aleatorio normal, una matriz de giro aleatoria.

El dominio de clasificación sintético aleatorio girado tiene cuatro dimensiones, como se puede observar en la figura 5.6.1-2.

El número de neuronas utilizado para este domino es de 45 (*figuras 5.1.2-2 y 5.1.3-2).*


aleatorioGirado - Bloc de notas

Archivo Edición Formato Ver Ayuda

1	0.94516818	1.1144947	0.82858067	1.0216024	1
2	0.9026836	0.89938511	0.73310911	0.9731807	1
3	1.0368666	0.60961715	0.33018444	0.76562567	0
4	0.93467285	0.80597949	0.69410401	0.94474082	0
5	1.0314388	0.78055183	0.59483254	0.92139564	0
6	0.67724737	1.0805494	1.0168058	1.0464768	1
7	0.87959066	1.2419866	1.133473	1.2149456	1
8	0.64136739	0.80044731	0.77384371	0.84722832	1
9	1.4529229	1.2438274	0.80498402	1.3040536	0
10	1.0610779	0.97536187	0.55762853	0.90762257	0
11	1.2592296	0.88848649	0.62045713	1.0847354	0
12	1.2108444	0.90471485	0.56068758	1.0019 0	
13	1.0267994	0.93292834	0.56902394	0.902855	0
14	1.1341144	1.309748	1.3146057	1.4989966	1
15	1.2472033	1.2547645	1.0284244	1.3363999	0
16	1.3021622	1.1164465	0.50286077	1.0013891	0
17	1.298545	1.2981947	0.88539681	1.2813739	0
18	1.4299208	1.5044786	1.2478936	1.582718	1
19	0.87278023	0.57947208	0.21376696	0.59861049	0
20	0.84452657	1.0494138	1.16739 1.2220693	1	
21	1.0676531	1.0787173	0.80348991	1.0737218	0
22	1.0292325	0.71589195	0.35929431	0.77784575	0
23	1.7560067	1.4144236	1.1522215	1.6845288	0
24	1.3312269	1.2081746	0.80856726	1.2234997	0
25	1.0256764	1.0127026	0.91854183	1.1213055	0
26	1.4128318	1.3536521	0.74642994	1.2302505	0
27	1.0931933	1.0614714	0.72557541	1.0357426	0
28	0.72374003	0.56461906	0.79387661	0.90504118	0
29	1.2401754	1.0870348	0.64357577	1.1074382	0
30	0.79478573	0.77896009	0.38233183	0.64060833	0
31	0.78798933	0.46918084	0.44309738	0.71717149	0
32	0.90515922	0.78384843	0.43399418	0.74680209	0
33	1.2431001	1.3627348	0.98187886	1.3288926	1
34	1.2318362	1.4539144	1.6011961	1.7098159	1
35	0.77355398	0.69798998	0.74680724	0.89338055	0
36	1.2496051	1.3263583	0.7970762	1.1886797	0
37	1.3167604	1.4276203	1.4886406	1.7116881	1
38	1.2198613	1.0838526	0.76040298	1.1570077	0
39	1.8018431	1.7518524	1.4202287	1.9144986	0
40	0.61649847	0.83810879	0.5652575	0.68738583	1
41	1.7037975	1.722794	1.5831985	1.9747907	1
42	1.4411881	1.698213	1.5972286	1.8279673	1
43	1.6422839	1.6915455	1.4561766	1.842152	1
44	1.0996811	1.0743497	1.2322245	1.4061572	1
45	0.66424272	0.7393522	0.56494676	0.74133032	1
46	0.49690866	0.54501856	0.39344397	0.51083274	0
47	1.3104036	1.2643677	0.8502988	1.2475754	0
48	1.5424886	1.3153787	1.1471891	1.5646807	0
49	0.87557669	0.82586007	0.72014224	0.95704547	1
50	1.2426025	1.518608	1.3809369	1.5812381	1
51	1.1788848	0.9084406	1.0074962	1.3182821	0
52	1.0852876	1.0824259	0.65382152	0.97880273	0
53	1.4269159	1.285763	1.2216127	1.5729426	0
54	0.43567382	0.3926439	0.26475921	0.4126513	0
55	1.2488609	1.1529691	0.52596915	0.98311644	0
56	0.99877021	0.8851665	0.83813278	1.098353	0
57	0.56263803	0.57582315	0.69299174	0.74307485	1
58	0.59433161	0.48658765	0.47115764	0.6206018	0

Figura 5.6.1-2: Fragmento del fichero de entrada del dominio de clasificación sintético aleatorio girado.

5.6.2 - Experimentación con matrices de distancias diagonales.

A continuación veremos los resultados obtenidos de realizar para este dominio la primera prueba de experimentación, es decir obligando a que la matriz solución obtenida sólo pueda ser diagonal.

Como ya sabemos de la experimentación con dominios anteriores, tanto en soluciones con matrices diagonales como simétricas, realizaremos los experimentos calculando la fitness a partir del error de entrenamiento de la red.

En cualquiera de los casos vamos a realizar el estudio de la matriz obtenida comparando la tasa de aciertos obtenida en los conjuntos de test de cada una de las particiones de la validación cruzada.

La fitness que se va a emplear para hacer evolucionar el algoritmo genético (AG) se obtiene a partir del error de entrenamiento de la red. Una vez que termina de entrenarse se aplica la transformación logarítmica de ese error para obtener la fitness.

En el dominio de clasificación sintético aleatorio disponemos de 300 patrones, los cuales debido al uso de validación cruzada vamos a dividir en 4 particiones. Por lo tanto vamos a tener 75 patrones en cada partición y 225 patrones en la parte de aprendizaje para proceder a la validación de cada una de las particiones.

Los parámetros del sistema son los siguientes:

- La solución es una matriz de distancias diagonal
- Se emplea validación cruzada: 4 particiones.
- Se utilizan el 100% de los patrones del fichero de aprendizaje de cada partición para el entrenamiento de la red, a partir de los cuales se calcula la fitness.
- Número de generaciones: 50
- Tamaño de la población: 15
- Tamaño del torneo: 2
- Elitismo de 1 elemento
- Cruce de dos puntos
- Probabilidad de cruce: 0,7
- Probabilidad de mutación: 0,01
- Conversión binario a real con una precisión de 2 dígitos en la parte entera binaria y 3 en la parte fraccionaria (LongVal=6; Lpunto=3).

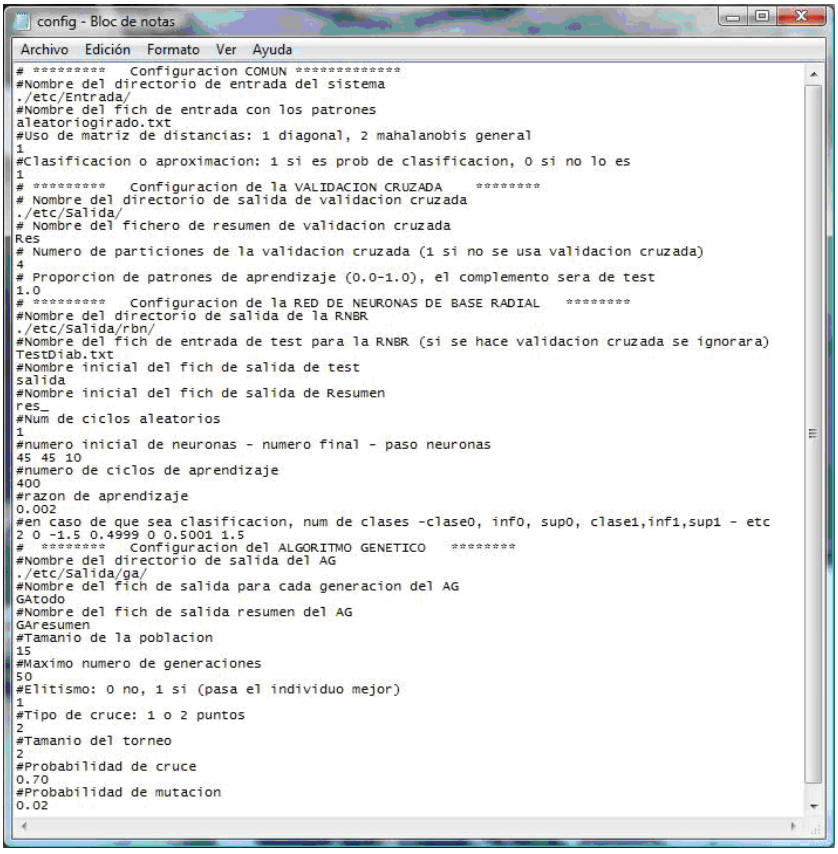


Figura 5.6.2-1: Fichero de configuración utilizado para la experimentación con el dominio sintético aleatorio limitando la solución a ser una matriz diagonal.

Una vez introducidos los parámetros en la aplicación *(figura 5.6.2-1)* procedemos a estudiar los resultados obtenidos:

Partición	Fitness con RNBR	Fitness con el sistema	Matriz solución obtenida
1	0,88158	0,98684	-3,6875 0 0 0 0 3,375 0 0 0 0 0 0 0 0 0 -0,1875
2	0,96053	0,96053	3,9375 0 0 0 0 3,375 0 0 0 0 -0,0625 0 0 0 0 0
3	0,90789	0,97368	-3,9375 0 0 0 0 3 0 0 0 0 0,0625 0 0 0 0 -0,3125
4	0,86842	0,98684	3,9375 0 0 0 0 -3,3125 0 0 0 0 0,3125 0 0 0 0 -0,125

Tabla 5.6.2-1: Resultados obtenidos para cada una de las particiones de validación cruzada restringiendo el resultado a matriz diagonal con fitness calculada a partir del error de entrenamiento.

En el gráfico *(figura 5.6.2-2)* podemos observar representada la tasa de acierto para cada una de las particiones. La última columna de la gráfica se corresponde con la tasa de acierto media:

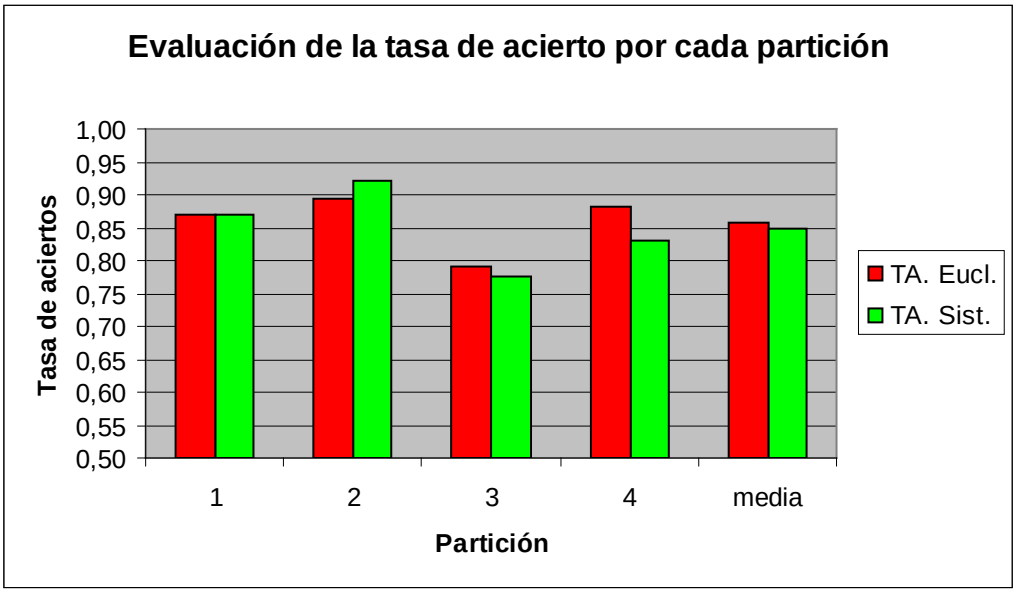


Figura 5.6.2-2: Evaluación de las particiones. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

Si se estudia la tasa de aciertos media obtenida por el sistema y por la red neuronal clásica, se llega a la conclusión que el número de aciertos obtenido por cada uno, prácticamente se igualan, siendo, por muy poco, mejores los obtenidos por la RNBR (0,8585 frente a 0,8486).

Analizando independientemente cada partición, se observa que en la primera partición, el número de aciertos obtenido tanto por el sistema como por la red son idénticos (0,8684).

En la segunda partición el número de aciertos del sistema (0,9210) es superior al conseguido por la RNBR (0,8947), pero en la tercera y cuarta partición es la red neuronal quien consigue un mayor número de aciertos (0,7894 y 0,8815 respectivamente) frente al sistema (0,7763 y 0,8289).

Se puede concluir, tras examinar los resultados obtenidos, que con el dominio sintético aleatorio girado y limitando a que la matriz solución sea diagonal, se obtienen buenos resultados tanto por la red neuronal como por el sistema (86% de aciertos).

En las siguientes gráficas podemos observar la evolución de la fitness para cada una de las particiones:

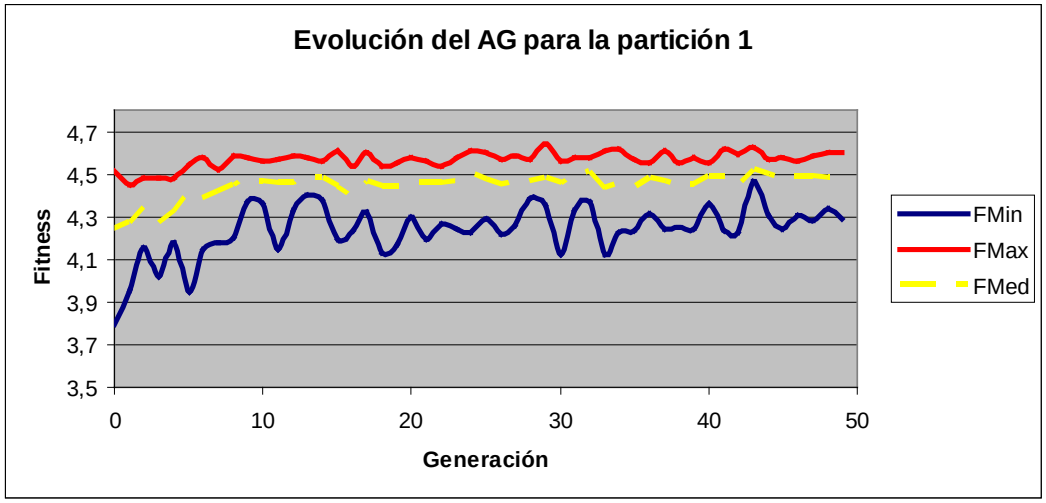


Figura 5.6.2-3: Evolución del AG para la partición 1. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

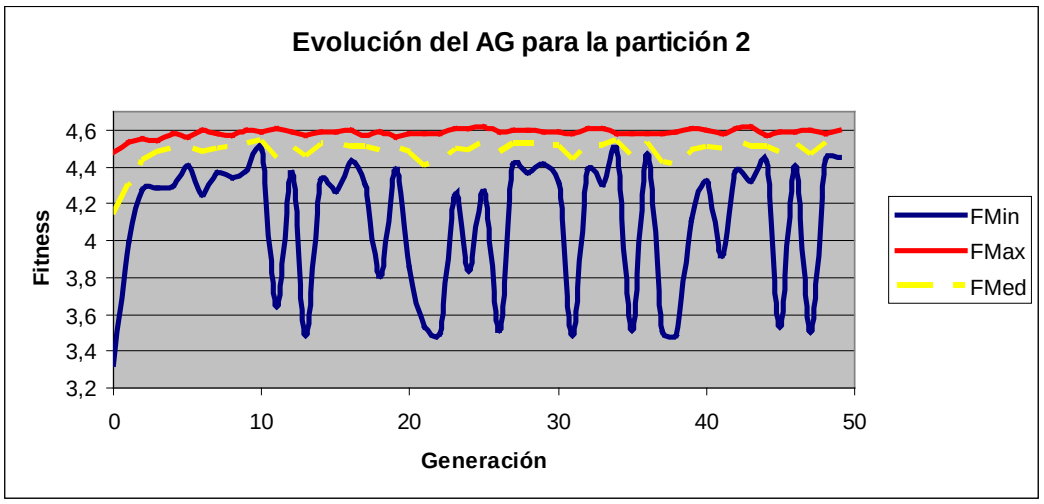


Figura 5.6.2-4: Evolución del AG para la partición 2. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

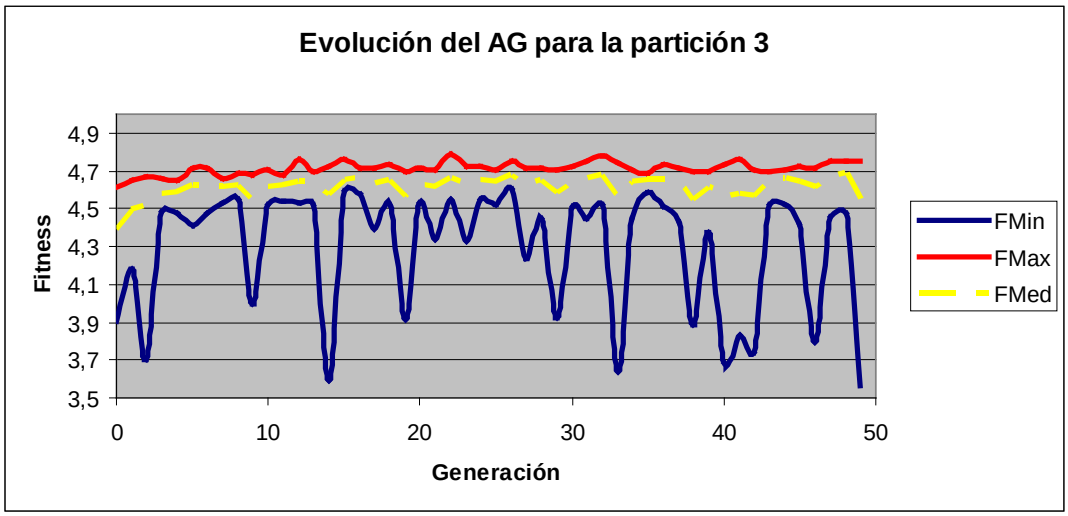


Figura 5.6.2-5: Evolución del AG para la partición 3. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

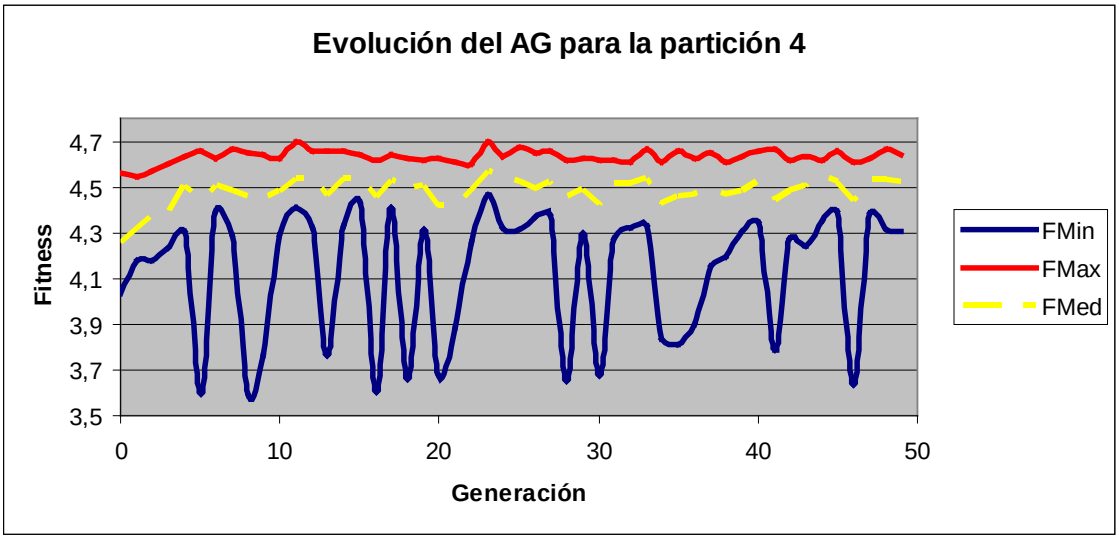


Figura 5.6.2-6: Evolución del AG para la partición 4. Matriz diagonal. Fitness calculada a partir del error de entrenamiento.

5.6.3 - Experimentación con matrices de distancias simétricas.

Para la siguiente prueba de experimentación sobre el dominio sintético aleatorio girado se van a utilizar matrices simétricas.

Como en el caso anterior de experimentación en este dominio, con matrices diagonales, se realizará el experimento calculando la fitness a partir del error de entrenamiento de la red y se va a utilizar validación cruzada, con 4 particiones.

La fitness que se va a emplear para hacer evolucionar el algoritmo genético (AG) se obtiene a partir del error de entrenamiento de la red. Una vez que termina de entrenarse se aplica la transformación logarítmica de ese error para obtener la fitness.

Recordemos que, para el dominio sintético aleatorio, disponemos de 300 patrones, los cuales debido al uso de validación cruzada vamos a dividir en 4 particiones. Por lo tanto vamos a tener 75 patrones en cada partición y 225 patrones en la parte de aprendizaje para proceder a la validación de cada una de las particiones.

Los parámetros del sistema son los siguientes:

- La solución es una matriz de distancias simétrica
- Se emplea validación cruzada: 4 particiones.
- Se utilizan el 100% de los patrones del fichero de aprendizaje de cada partición para el entrenamiento de la red, a partir de los cuales se calcula la fitness.
- Número de generaciones: 50
- Tamaño de la población: 15
- Tamaño del torneo: 2
- Elitismo de 1 elemento
- Cruce de dos puntos
- Probabilidad de cruce: 0,7

- Probabilidad de mutación: 0,01
- Conversión binario a real con una precisión de 2 dígitos en la parte entera binaria y 3 en la parte fraccionaria (LongVal=6; Lpunto=3).

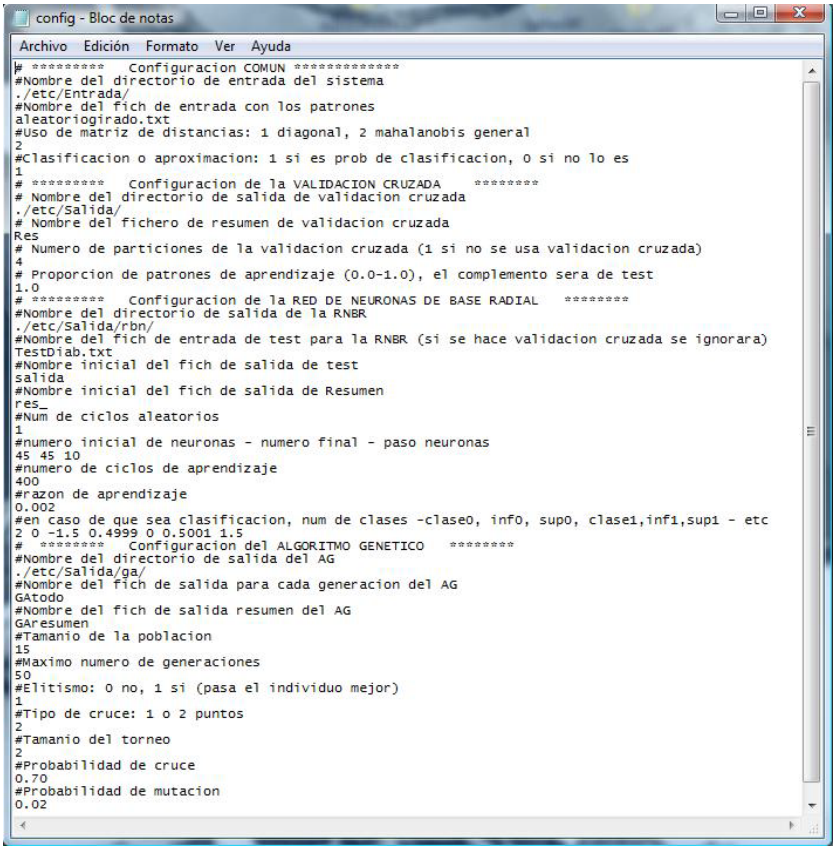


Figura 5.6.3-1: Fichero de configuración utilizado para la experimentación con el dominio sintético aleatorio limitando la solución a ser una matriz simétrica.

Una vez introducidos los parámetros en la aplicación *(tabla 5.6.3-1)* procedamos a estudiar los resultados obtenidos:

Partición	Tasa de aciertos RNBR	Tasa de aciertos del sistema	Matriz solución obtenida
1	0,86842	0,96053	0,8125 0,6875 1,625 -2,25 0,6875 1,0625 -2,0625 0,0625 1,625 -2,0625 1 -0,3125 -2,25 0,0625 -0,3125 2,4375
2	0,89474	0,90789	-1,8125 -1 0,0625 3 -1 -0,125 1,8125 0,0625 0,0625 1,8125 -0,25 -2 3 0 -2 -1,8125
3	0,78947	0,94737	-3,1875 1,4375 -3,25 3,8125 1,4375 -1,3125 1,8125 -1 -3,25 1,8125 -0,125 1,3125 3,8125 -1 1,3125 -3,6875
4	0,88158	0,90789	-0,9375 -0,0625 3,4375 -1,75 -0,0625 -0,125 -1,1875 0,9375 -1,75 -1,1875 -0,4375 -1,9375 -0,0625 0,9375 -1,9375 2

Tabla 5.6.3-1: Resultados obtenidos para cada una de las particiones de validación cruzada restringiendo el resultado a matriz simétrica con fitness calculada a partir del error de entrenamiento para el dominio sintético aleatorio girado.

En el gráfico *(ver figura 5.6.3-2)* podemos observar representada la tasa de acierto para cada una de las particiones. La última columna de la gráfica se corresponde con la tasa de acierto media:

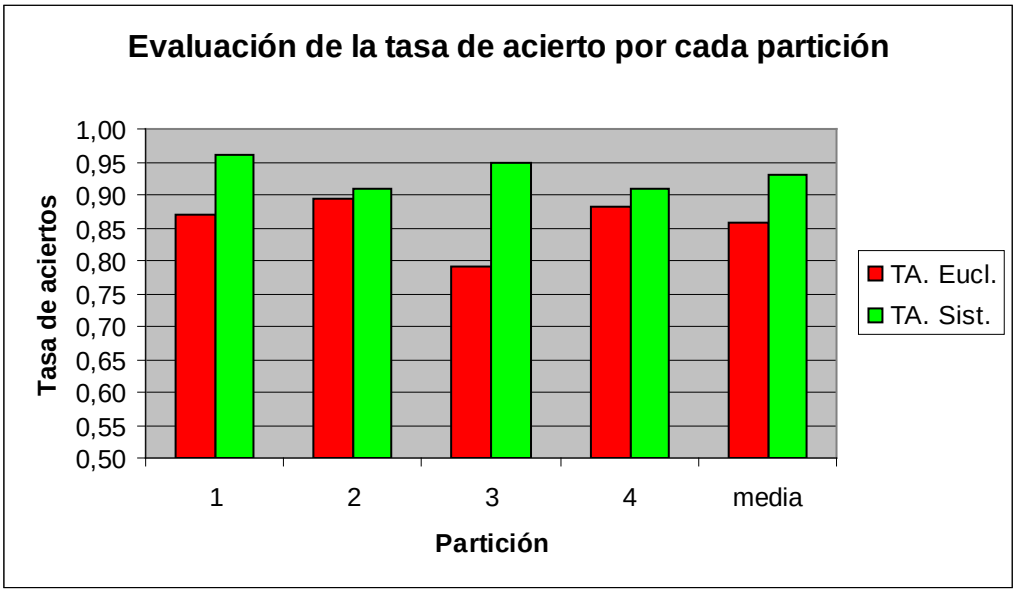


Figura 5.6.3-2: Evaluación de las particiones. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

Dada la tasa de aciertos media obtenida por el sistema *(0,9309)* y en la RNBR *(0,8684)* se puede observar como en el caso del dominio sintético aleatorio girado los aciertos obtenidos por el sistema mejoran de manera considerable los conseguidos por una RNBR clásica, consiguiendo el sistema un 7% aproximadamente más de aciertos de media.

Lo anterior se confirma, examinando los aciertos obtenidos partición a partición. En este caso en todas las particiones se han obtenido resultados mucho mejores que con una red neuronal.

Hay que destacar la gran diferencia, en relación a aciertos, que se da en la primera y tercera partición, en las cuales se alcanzan un 10% y un 16% más, respectivamente.

En la segunda y cuarta partición se igualan más los aciertos, siendo siempre ligeramente superior los obtenidos por el sistema.

En las siguientes gráficas podemos observar la evolución de la fitness para cada una de las particiones:

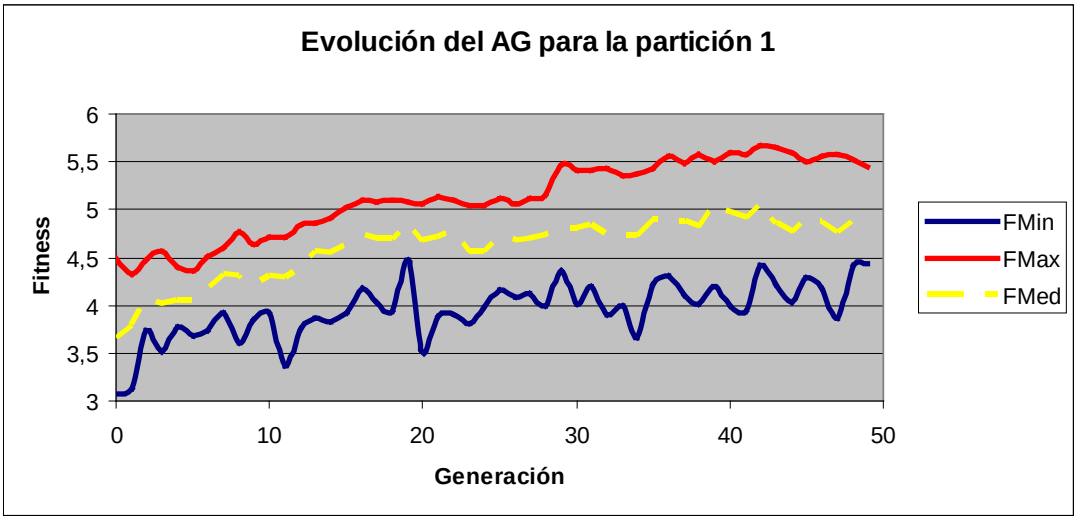


Figura 5.6.3-3: Evolución del AG para la partición 1. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

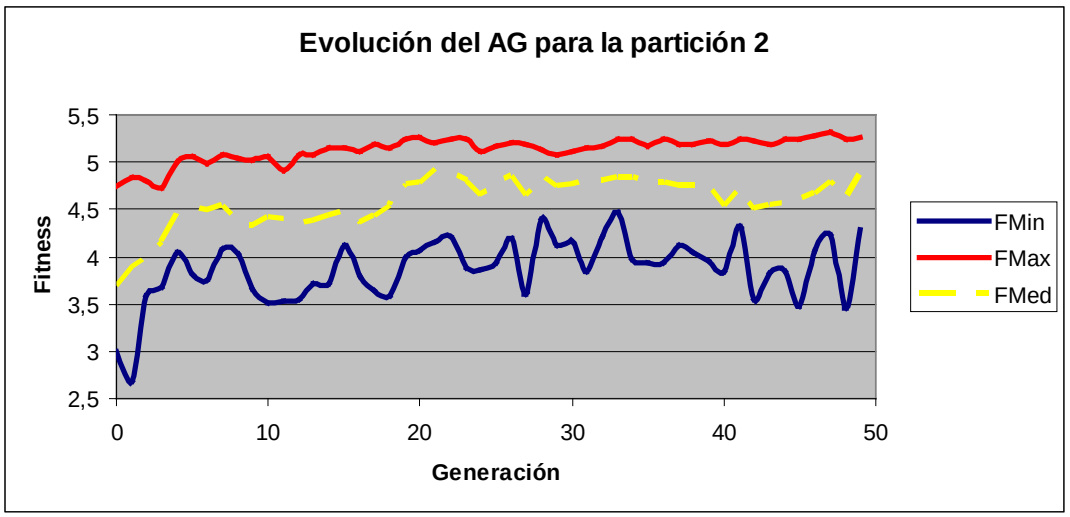


Figura 5.6.3-4: Evolución del AG para la partición 2. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

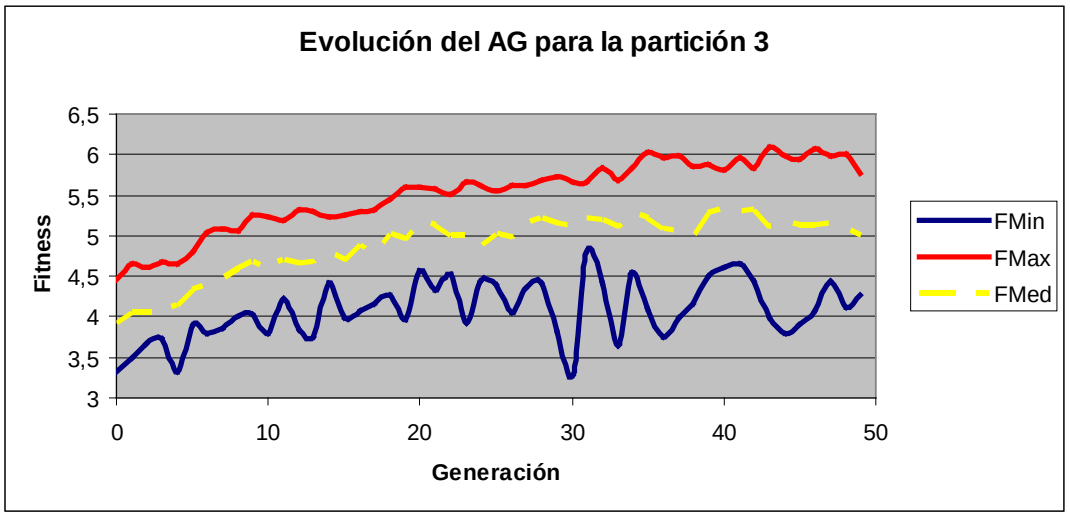


Figura 5.6.3-5: Evolución del AG para la partición 3. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

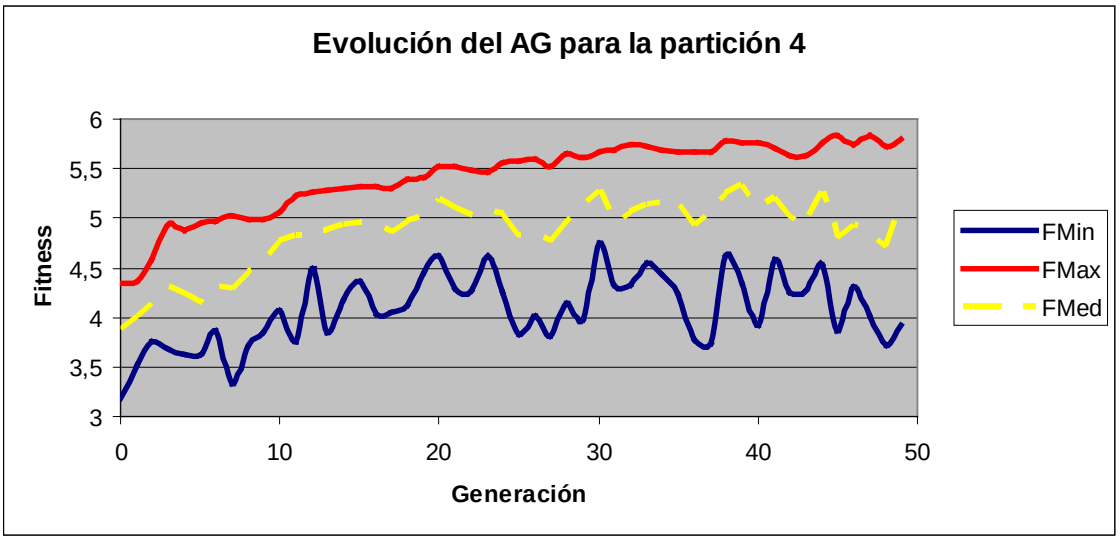


Figura 5.6.3-6: Evolución del AG para la partición 4. Matriz simétrica. Fitness calculada a partir del error de entrenamiento.

6. CONCLUSIONES Y FUTURAS LÍNEAS DE INVESTIGACIÓN

En este último capítulo del proyecto se va a realizar una recopilación de las conclusiones a las que se ha llegado en el capítulo de experimentación.

En primer lugar se expondrán las conclusiones a las que se ha llegado a partir de la experimentación realizada sobre el dominio de la diabetes, para proceder después con el resto de dominios.

Finalmente se incluirán las posibles líneas de investigación y las mejoras realizables a partir del trabajo realizado en este proyecto.

6.1 – Conclusiones

El objetivo del proyecto era evaluar el funcionamiento del sistema, encontrando la matriz de distancias más óptima, en comparación con una red neuronal de base radial utilizando para ello un software desarrollado previamente en proyectos anteriores.

El software se basa en algoritmos genéticos y estrategias evolutivas y permite seleccionar entre estos dos procesos evolutivos mediante un fichero único de configuración. En nuestro caso el 100% de la experimentación se ha realizado bajo la acción de los algoritmos genéticos.

6.1.1 – Análisis de los resultados obtenidos.

En general, podemos decir que los resultados obtenidos para los dominios de Rectas Ruido y Aleatorio Girado son bastantes mejores que los conseguidos para los dominios de Diabetes, Ripley y Aleatorio. Para todos los dominios se ha seguido el mismo plan de experimentación y los resultados obtenidos presentan un comportamiento diferente para cada uno de los dominios, además dependiendo de si la matriz solución es simétrica o diagonal.

Los resultados obtenidos permitiendo únicamente matrices de distancia diagonales como solución son mejores que los obtenidos permitiendo el uso de matrices de distancias simétricas. Al permitir el uso de matrices simétricas, el sistema realiza un peor ajuste, que queda reflejado en los resultados obtenidos por los dominios de la diabetes y aleatorio, donde el ajuste es mucho peor que usando una RNBR.

Comparando los valores de la matriz solución obtenidos utilizando matriz diagonal y simétrica, se observa que los valores de la diagonal son superiores en el caso que la matriz sea simétrica.

Cuando se ha empleado una matriz solución simétrica no se han encontrado elementos fuera de la diagonal principal que sean significativamente distintos de cero.

Las matrices diagonales, o bien obtienen resultados similares a la distancia euclídea, o bien son mejores (dominio Aleatorio).

Las matrices simétricas obtienen a veces resultados peores que la distancia euclídea (Diabetes y Aleatorio). Sin embargo, en los dominios artificiales donde se sabe que una matriz simétrica es necesaria (Aleatorio Girado y Rectas Ruido), los resultados de la matriz simétrica son mejores que los de la euclídea y los de la diagonal. Será por tanto necesario estudiar en el futuro porqué a pesar de funcionar como se espera en estos dos últimos dominios, las matrices simétricas obtienen malos resultados en otros dominios

6.1.2 – Resumen de los resultados obtenidos

DOMINIO	TIPO DE MATRIZ SOLUCIÓN	TASA DE ACIERTOS RNBR	TASA DE ACIERTOS SISTEMA	VALORACIÓN DEL SISTEMA
DIABETES	Diagonal	0,8110	0,8181	=
	Simétrica	**0,8201**	0,7581	X
RIPLEY	Diagonal	0,9001	0,8956	=
	Simétrica	0,8961	0,8977	=
RECTAS 45º	Diagonal	0,4342	0,3480	=
	Simétrica	0,4362	**0,9215**	✓
RECTAS 0º	Diagonal	0,4264	**0,9950**	✓
	Simétrica	0,4362	**0,9901**	✓
ALEATORIO	Diagonal	0,9046	**0,9769**	✓
	Simétrica	**0,9078**	0,8322	X
ALEATORIO GIRADO	Diagonal	0,8585	0,8486	=
	Simétrica	0,8684	**0,9309**	✓

Tabla 6.1.2-1: Resultados de la experimentación

6.2 – Futuras líneas de investigación

6.2.1 – Posibles mejoras a partir del trabajo realizado

Las posibles mejoras que se podrían llevar a cabo en función fundamentalmente de los resultados obtenido en la fase de experimentación se deben centrar en el perfeccionamiento del algoritmo de búsqueda de la matriz solución para que los resultados obtenidos por el sistema se acercasen a los obtenidos por una RNBR.

Se debería encontrar la forma de reducir el tiempo computacional empleado en cada experimento, puesto que en algunos casos empleando validación cruzada con 4 particiones el tiempo de procesamiento ha llegado hasta las 16 horas.

6.2.2 – Posibles líneas de investigación

- Además del uso de distancias de Mahalanobis, se podrían implementar otras funciones de distancias que serían empleadas para ponderar las variables de entrada de la RNBR.

- En este proyecto todos los experimentos han sido desarrollados sobre dominios de clasificación, en el futuro se podría realizar un proyecto que incluya dominios de aproximación, que ayuden a mejorar el sistema empleado.

- Estudiar por qué las matrices simétricas no funcionan bien en algunos casos y encontrar una solución a este problema.

- Se podrían desarrollar funciones en C++ que generaran de forma automática las gráficas de evolución de las tasas de aciertos y sería interesante la posibilidad de exportar los datos obtenidos para la matriz solución a Word o a Excel.

ANEXO A

ESQUEMAS EN LOS ALGORITMOS GENÉTICOS

Los esquemas son la forma de representar mediante una estructura estas similitudes entre cadenas y clases de cadenas.

Haciendo referencia, una vez más, a Holland, un esquema se puede considerar como una especie de plantilla en la cual se describen un subconjunto de cadenas con ciertas similitudes en ciertas posiciones de la misma.

Para que se entienda mejor, vamos a utilizar solo cadenas que contengan un alfabeto binario (0's y 1's). Para poder considerar los esquemas al alfabeto le debemos agregar un símbolo especial (*) que va a significar que no tiene importancia el símbolo del alfabeto que esté en esa posición particular, es decir, nos va a dar igual que sea un 1 o un 0. Por lo tanto, el alfabeto del que vamos a disponer para crear cadenas (esquemas) va a ser un alfabeto ternario (0's, 1's y *)

Alfabeto = { 0 , 1 , * }

La intención del uso de esquemas es la de clarificar si las cadenas tienen un patrón coincidente. Un esquema hará coincidir una cadena cuando si en cada posición del esquema donde hay un 1 coincide con un 1, un 0 coincide con un 0 y un * coincide con un 0 o con un 1. Veamos varios ejemplos para entenderlo.

- Dado el esquema *1111, coincidirá con dos cadenas:

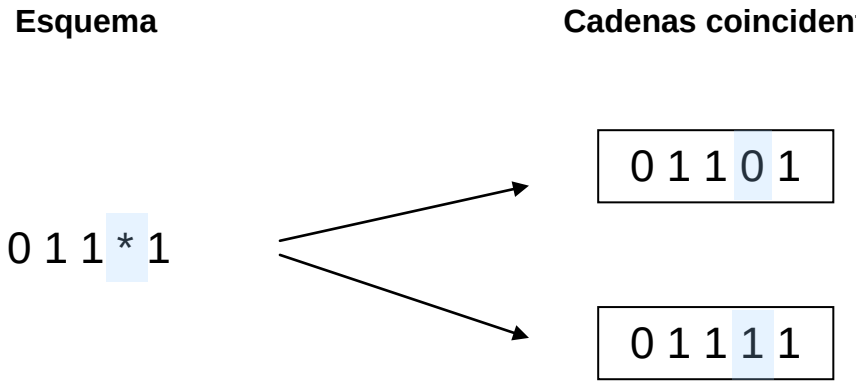


Figura A.1-1

- El esquema *100* describe un subconjunto de cuatro cadenas de longitud 5 y que se caracterizan por tener en la posición segunda un uno y en la posición tercera y cuarta un cero:

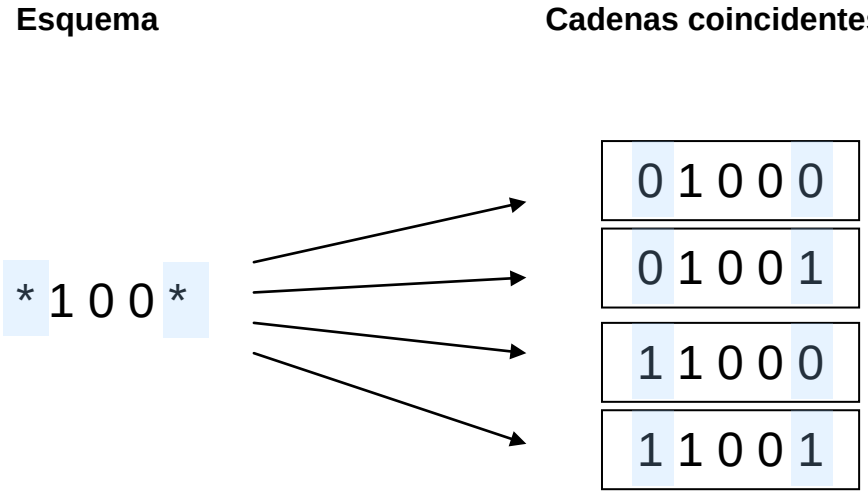


Figura A.1-2

- Como último ejemplo, la cadena 00*** coincide con cualquiera de las 8 cadenas de longitud 5, que comienzan por dos ceros.

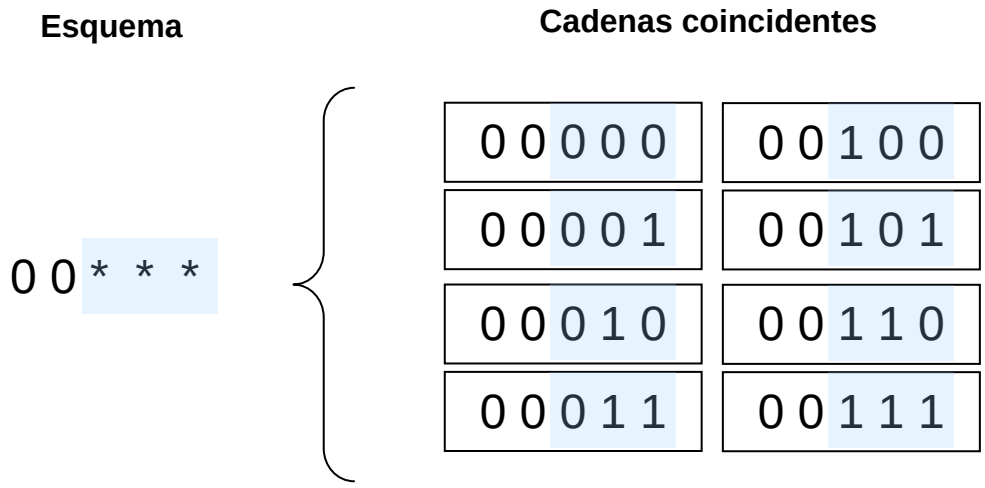


Figura A.1-3

De esta manera, el esquema nos proporciona una herramienta para hablar de similitudes bien definidas en cadenas de longitudes finitas y constituidas con un alfabeto finito.

Se debe entender que * solo es un meta-símbolo, es decir un símbolo que representa a otros y nunca se procesará explícitamente por el algoritmo genético.

El número de esquemas distintos que puede tener un alfabeto viene dado por la ecuación,

Número de esquemas = (k-l)

Siendo k, el número de caracteres del alfabeto y l, la longitud de la cadena.

Vamos a estudiar a continuación el efecto que produce la reproducción, el cruce y la mutación en el crecimiento o decaimiento de los esquemas de generación en generación.

El *efecto de la reproducción* en un esquema particular se deduce fácilmente. Cuanto mas adecuadas sean las cadenas en las que se encuentren, mayores probabilidades de selección tendrán los mismos.

El *efecto de cruce*, no va a dispersar un esquema si no lo corta, pero si lo alterará cuando esto suceda.

Por ejemplo, si tenemos los esquemas 1***0 y **11*, el primero será fácilmente alterable por el cruce, mientras que el segundo no lo será tanto utilizando este mecanismo.

El cruce tenderá a mantener los esquemas de longitud mas corta y éstos podrán propagarse de generación en generación si se encuentran en cadenas con un grado de adecuación alto.

El efecto de mutación en un esquema normalmente no lo descompondrá puesto que normalmente se utilizará una probabilidad muy baja de mutación

Como conclusión, podemos decir que los esquemas de longitud corta, a los que se les va a llamar bloques de construcción, en cadenas con un grado de adecuación alto, se propagarán de generación en generación de modo exponencial.

De este modo, los bloque de construcción, tienen un papel muy importante en el funcionamiento de los algoritmos genéticos.

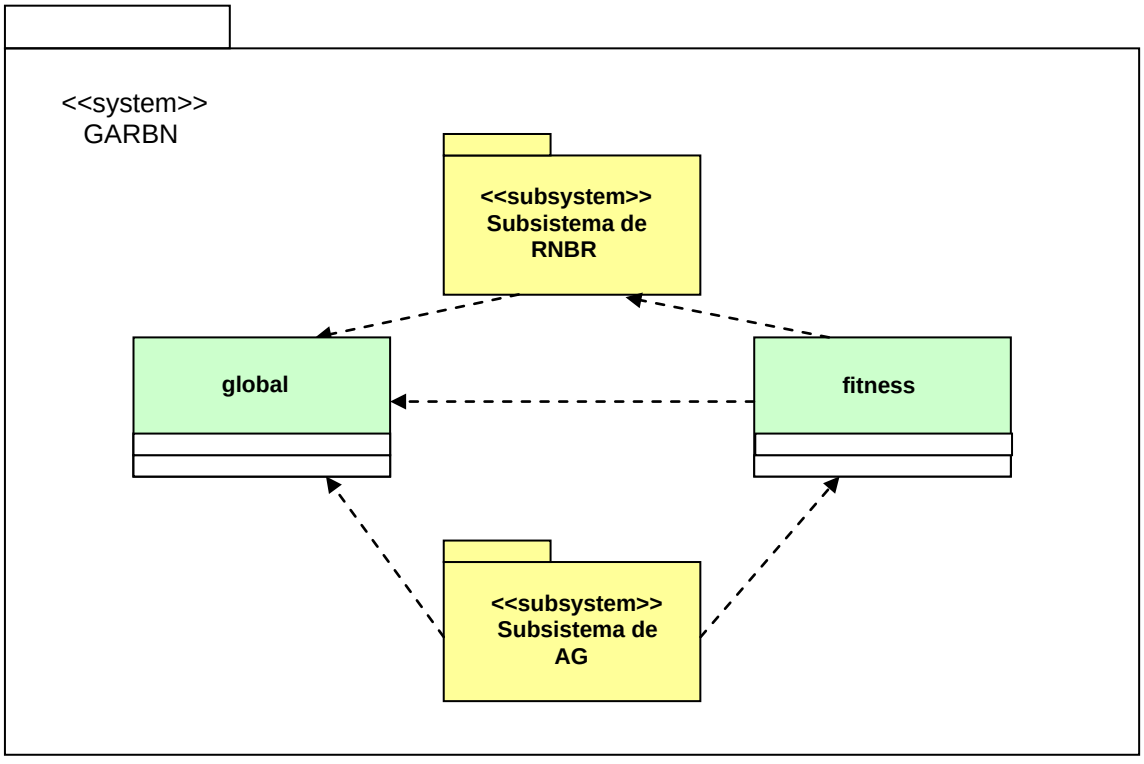
ANEXO B

ESTRUCTURA DEL SISTEMA

B.1. – Arquitectura del sistema

En este anexo se intenta dar una descripción a muy alto nivel de cómo funciona el sistema, dividiéndolo en dos subsistemas (RNBR y AG).

En el siguiente diagrama se puede ver esta división del sistema.



A continuación se explica de forma breve las funciones principales de los dos subsistemas:

- **Subsistema de RNBR**

  A este subsistema pertenecen tres clases (rbn, neurona y punto). En proyectos anteriores estas clases han sido modificadas paulatinamente para que sean capaces de trabajar con matrices de Mahalabobis.

- **Subsistema de AG (Algoritmo genético)**

  El subsistema del algoritmo genético está compuesto por dos clases (población y cromosoma), que sirven para controlar el funcionamiento del algoritmo

Además de estos dos subsistemas, en el sistema se utilizan dos clases que no pertenecen a ninguno de los dos y que tienen en común que solo pueden tener una instancia creada durante el funcionamiento del sistema. Estas clases son:

- **Clase global**

  En esta clase se encuentran todos los detalles de funcionamiento comunes y que pueden utilizarse en cualquier parte del programa.

  En ella, se definen los macros y tipos de datos utilizados en el programa, así como un repositorio de funciones comunes.

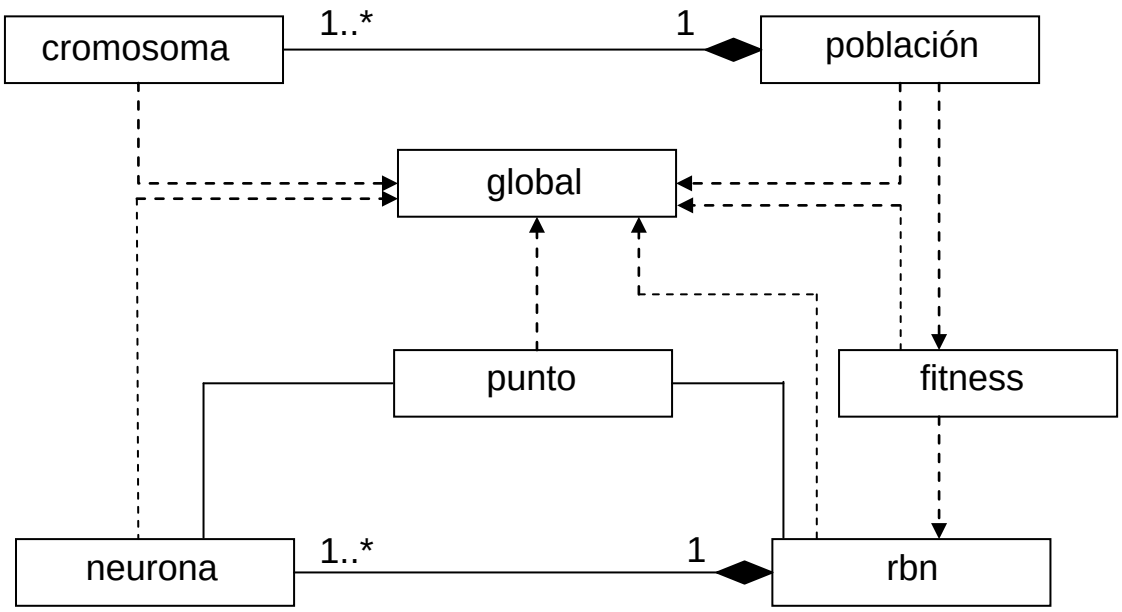
- **Clase fitness**

  Esta clase es utilizada para que el subsistema del algoritmo genético llame al subsistema de la red neuronal cuando sea necesario calcular la fitness utilizando la RNBR.

  Se puede considerar que esta clase es herencia del método de main del proyecto inicial, donde al principio solo se utilizaba el subsistema de la RNBR para entrenar la red con la distancia euclídea clásica.

B.2. – Diagrama de clases del sistema

Como se ha visto de forma breve en el apartado anterior, cada subsistema utiliza unas determinadas clases. A continuación se expone el diagrama de clases del sistema que se va a emplear para realizar la fase de experimentación:



El funcionamiento de la red neuronal lo controlan las clases rbn, neurona y punto. El algoritmo genético se encuentra implementado dentro de las clases población y cromosoma.

La clase global es utilizada por todas las clases con distintos fines y para calcular el grado de adecuación de cada una de las soluciones candidatas (cromosoma) mediante la RNBR, se emplea la clase fitness.

B.3. – Funcionamiento del sistema

En este apartado se va a desglosar el funcionamiento del sistema empleado, haciendo hincapié en la parte que explica todo lo relacionado con la mecánica del algoritmo genético.

B.3.1 – Función principal

A continuación se va a describir el diseño y la implementación que corresponde al método *main* que se encuentra en el fichero "mainGaRbn.cpp" de C++

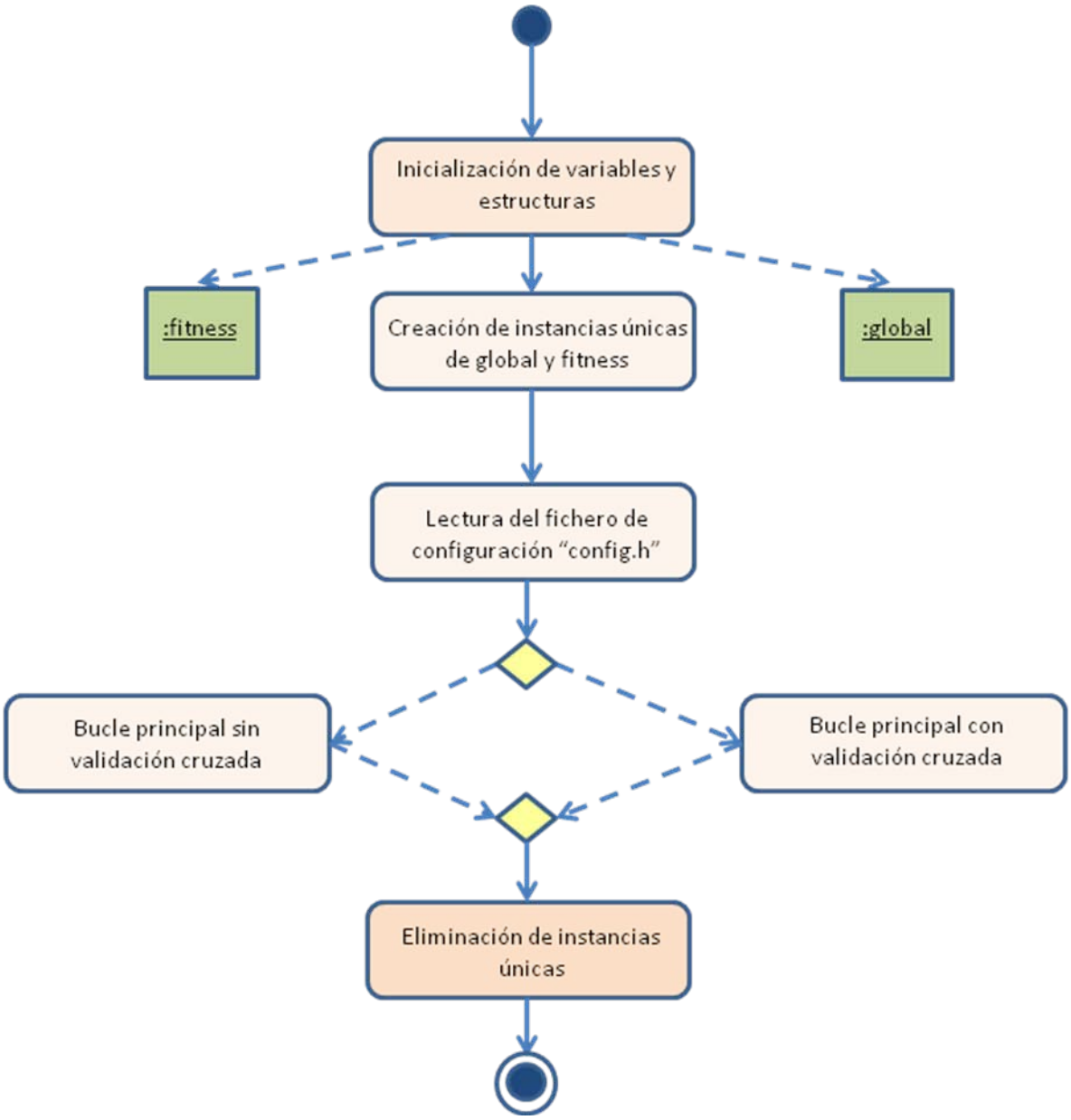
El método principal se encarga de crear y enviar los mensajes que sean necesarios a los demás objetos y así el programa vaya siguiendo su flujo de funcionamiento.

El **bucle principal** comienza con la ***inicialización de variables y estructuras***, que posteriormente se van a utilizar durante el funcionamiento del sistema, después de la inicialización de las variables y estructuras ***se crea una instancia de la clase global y otra para la clase fitness***, que van a poder ser utilizadas en cualquier parte del programa y a continuación se llevará a cabo la ***lectura del fichero de configuración "config.h"*** mediante el método global LeerConfiguración (TCONFIG*).

Después de leer el fichero de configuración, si el sistema utiliza un algoritmo genético (en este proyecto no están implementadas las estrategias evolutivas), el programa ***se meterá en un bucle principal sin o con validación cruzada***, dependiendo de lo configurado en el fichero config.ini.

Finalmente y antes de terminar el funcionamiento del programa, ***se eliminan las instancias únicas de las clase global y fitness***.

A continuación podemos observar el diagrama de actividad del flujo dl bucle principal:



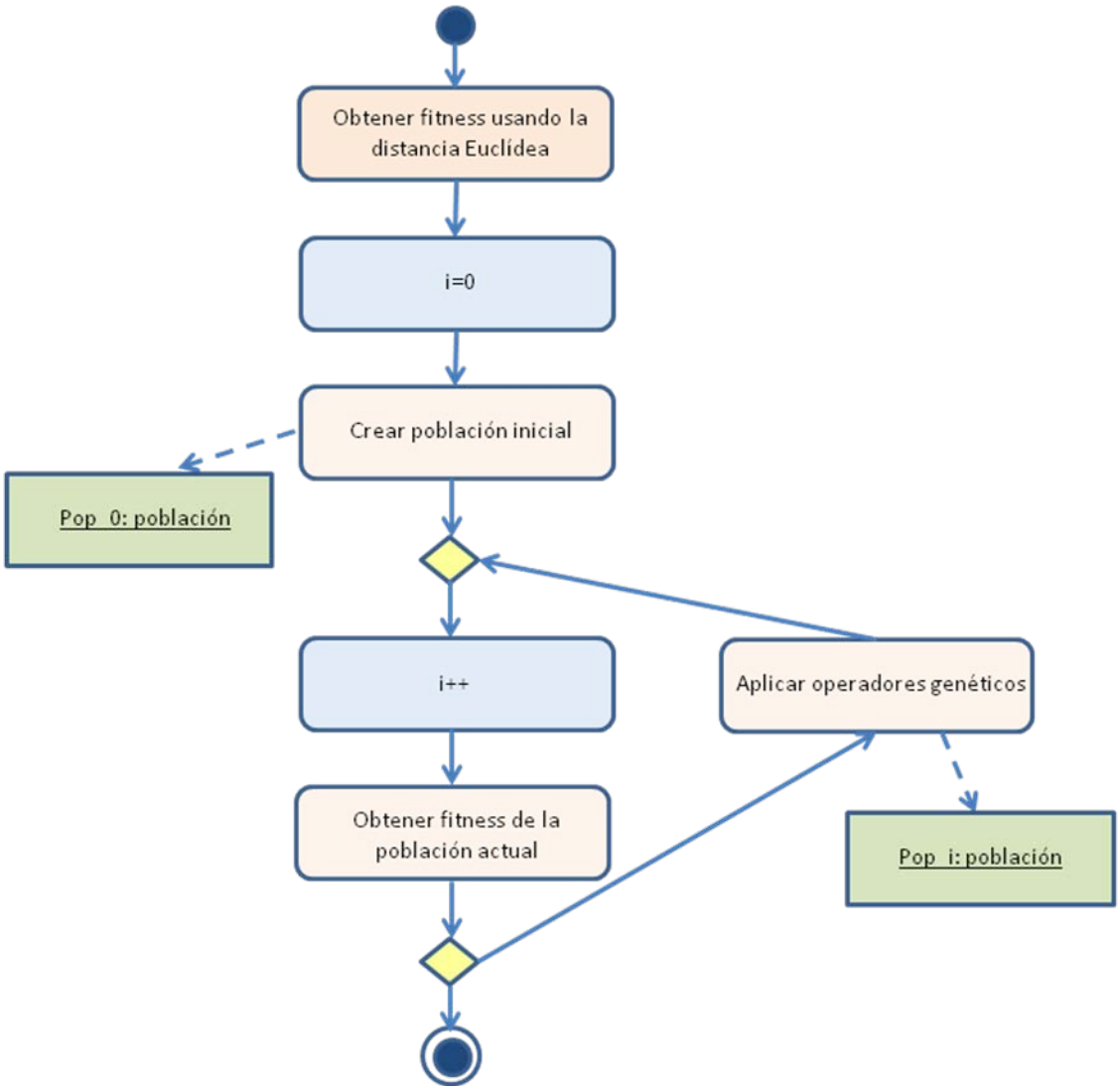
B.3.1.1 – Sub-bucle con algoritmo genético (AG)

El sub-bucle con algoritmo genético utiliza operadores genéticos. Detallemos a continuación los bucles principales sin validación cruzada y con validación cruzada:

B.3.1.2 - Sub-bucle sin validación cruzada.

El bucle comienza ***obteniendo la fitness usando la distancia euclídea***(la obtiene con los ficheros de entrenamiento y test de la RNBR que han sido indicados en el fichero config.h) para posteriormente poner el contador i a 0. A continuación ***se crea la población inicial*** (se crean los cromosomas inicializados aleatoriamente para que a partir de ellos vaya evolucionando el AG).

Una vez creada la población ***se obtiene la fitness de la población actual*** (para ello se calcula la fitness de todas las soluciones candidatas) y finalmente ***se aplican los operadores genéticos*** obteniendo una nueva generación y así repetir el proceso a partir de la obtención de la fitness a partir de la población actual.



B.3.1.3. - Sub-bucle con validación cruzada.

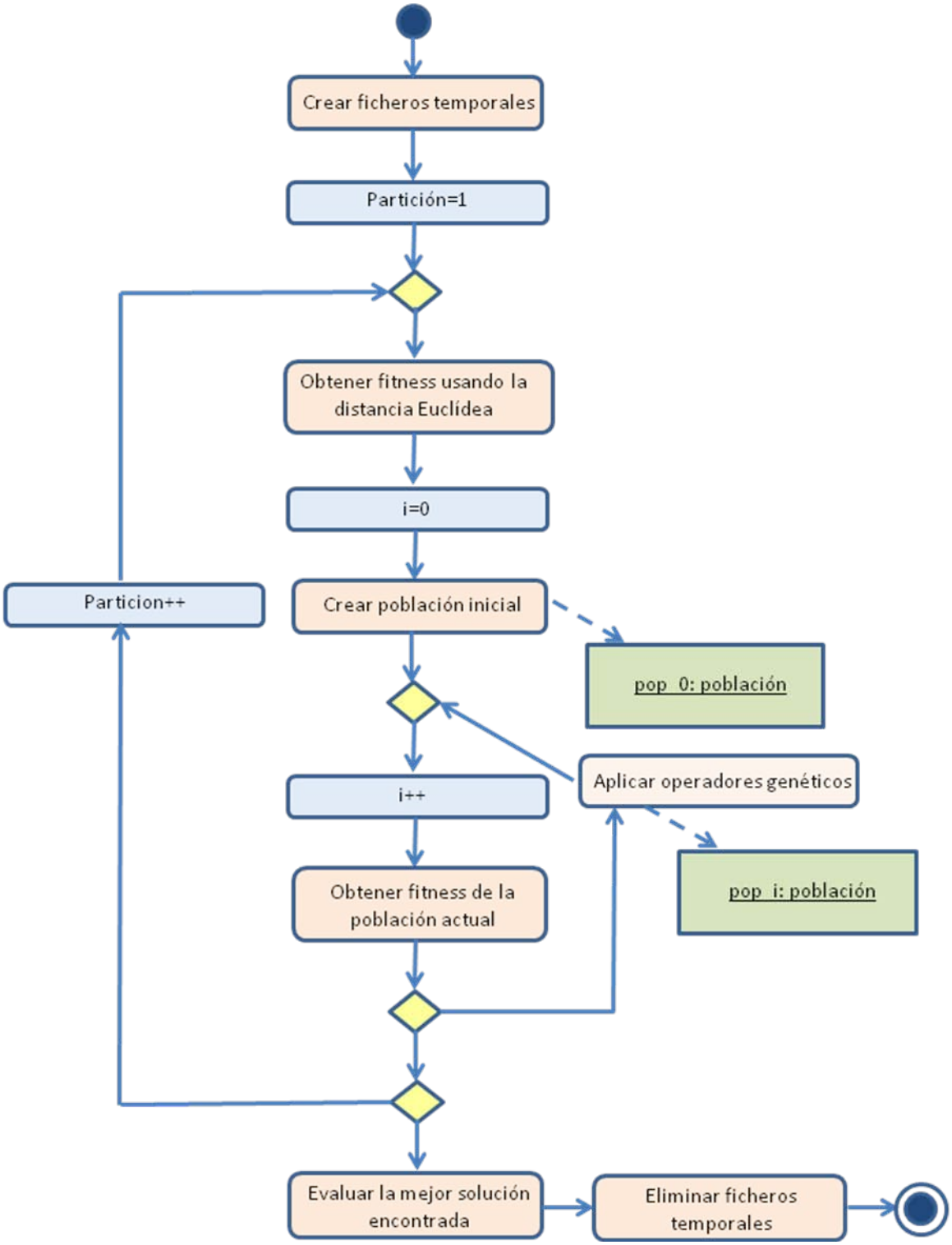
El bucle del algoritmo genético con validación cruzada comienza ***creando los ficheros temporales*** (primero se crean los ficheros para cada partición de la validación cruzada) y al igual que en el sub-bucle sin partición cruzada ***se obtiene la fitness utilizando la distancia euclídea*** (en este caso, al usar validación cruzada la fitness se obtiene a partir de los ficheros de aprendizaje y validación y se calculará a partir del error de validación de la RNBR al aplicarle los patrones reservados para la validación cruzada. En caso de que el problema fuese de clasificación, se tomará directamente la tasa de aciertos de aplicarles esos patrones a la red ya entrenada).

A continuación ***se crea la población inicial*** y ***se obtienela fitness de la población actual*** (se calculará la fitness de todas las soluciones candidatas. En este caso, se emplearán los ficheros de entrenamiento y test generados para la partición actual.).

Finalmente ***se aplican los operadores genéticos*** y ***se evalua la mejor solución encontrada***(se mira la fitness de la mejor solución encontrada aplicándole la porción de patrones reservados para validación cruzada).

Una vez evaluadas todas las particiones, ***se eliminan los ficheros temporales***.

En la siguiente página podemos observar el diagrama de actividad del flujo el sub-bucle con validación cruzada:


Crear ficheros temporales
Partición=1
Obtener fitness usando la distancia Euclídea
i=0
Particion++
Crear población inicial
pop_0: población
Aplicar operadores genéticos
i++
pop_i: población
Obtener fitness de la población actual
Evaluar la mejor solución encontrada
Eliminar ficheros temporales

B.3.2. – Clase global

La clase global se implementa dentro de los ficheros global.h y global.cpp.

En esta clase están todos los detalles acerca de los funcionamientos comunes y que pueden ser empleados en cualquier parte del programa. También, se definen los macros y los tipos de datos que se utilizan en el programa.

Esta clase solo tiene una instancia

A continuación se exponen las partes mas relevantes del diseño de esta clase para el funcionamiento del sistema:

Elementos comunes:

Son aquellos elementos contenidos en esta clase y que serán empleados por las demás clases del programa.

- *Inclusión de archivos*: Librerías que necesitan una o varias clases, por lo que, en su día cuando fue implementado, se utilizó la directiva de C++ #include.

- *Definición de macros*: Se definieron aquellos macros que son necesarios para el funcionamiento del programa. Entre ellos, existen algunos parámetros de configuración, junto con los ya existentes en el fichero 'config.ini'.

- *Definición de tipos:* En la clase global se definieron los tipos de datos comunes utilizado por el resto de clases

Creación de ficheros para la validación cruzada:

La instancia de la clase global se encarga de crear los ficheros temporales que son empleados en cada una de las particiones de la validación cruzada; cada partición tendrá cuatro ficheros de entrada diferentes:

- *Fichero de aprendizaje:* Este fichero contiene los datos con los que se entrenará la red. Está compuesta tanto por los patrones de entrenamiento como los de test de la red neuronal de base radial.

- *Fichero de validación*: Es la porción de patrones que se han reservado en cada partición para llevar a cabo la validación cruzada.

- *Fichero de entrenamiento*: Este fichero se construye a partir del de aprendizaje y contiene los patrones de entrenamiento de la RNBR.

- *Fichero de test*: Este fichero también se construye a partir del de aprendizaje y se diferencia del fichero de entrenamiento en que contiene los patrones de test de la RNBR.

El número de particiones que se empleará a la hora de realizar la experimentación con validación cruzada viene determinado por el valor que tome el parámetro numPart en la estructura TCONFIG que se encuentra en el fichero de configuración 'config.ini'.

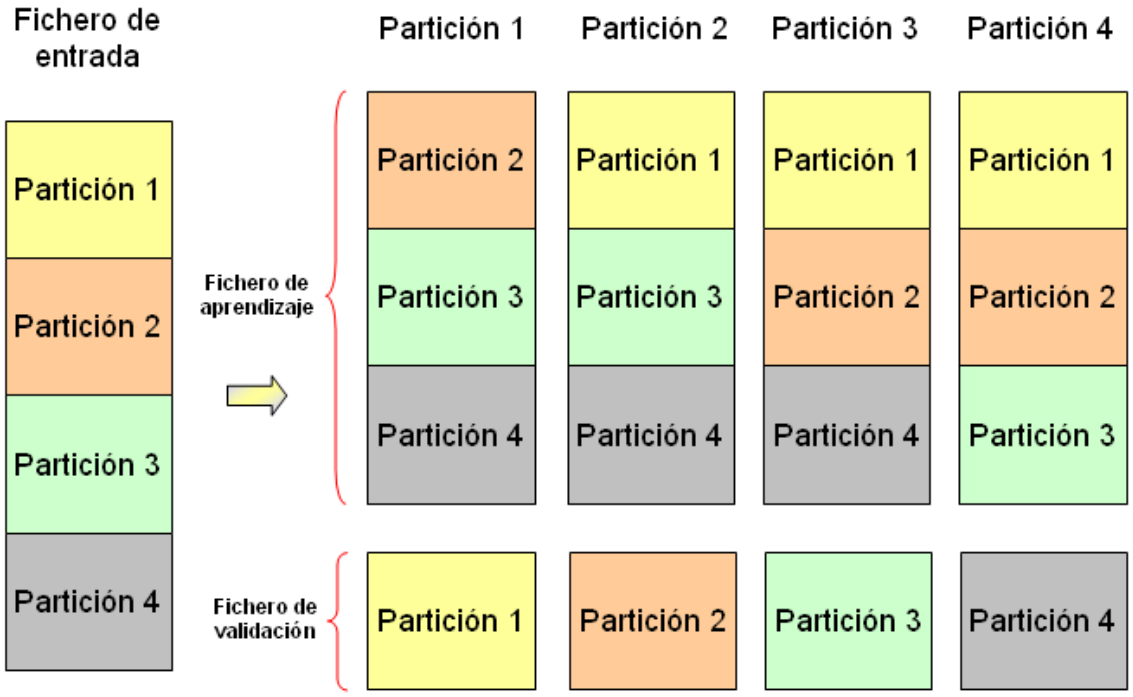
Este número de particiones determina el número de patrones que se encuentran en cada fichero de aprendizaje y de validación para cada una de las particiones.

El proceso consiste en dividirle número total de patrones disponibles en particiones con el mismo tamaño, siendo cada una de ellas la parte de validación de esa partición y el resto de los patrones pertenecerán al fichero de aprendizaje de esa partición.

Por ejemplo, si disponemos de un número total de 100 patrones y queremos experimentar con validación cruzada de 4 particiones, cada partición constará de 100/4= 25 patrones (número total de patrones entre el número de particiones).

Por lo tanto, para cada una de las particiones obtenidas se tendrá una parte de validación de 25 patrones y una parte de aprendizaje de 75 patrones.

A continuación, se verá de forma gráfica este ejemplo:



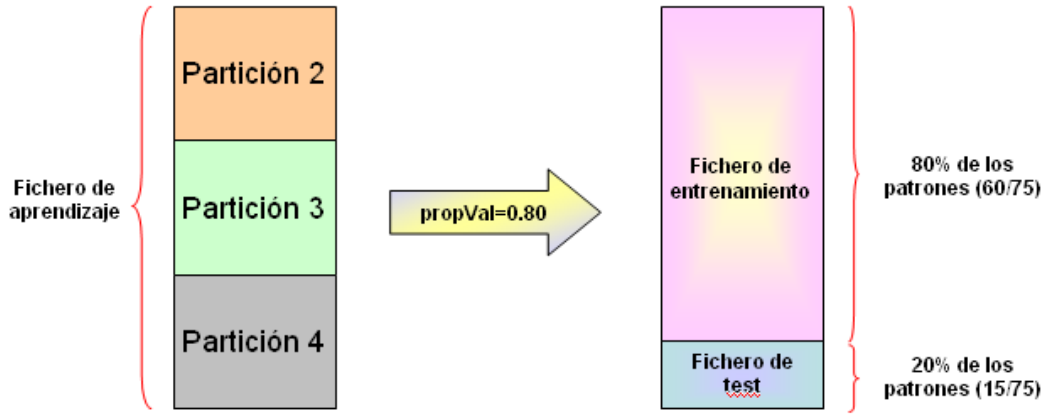
En este ejemplo, cada uno de los ficheros de validación tendrá ¼ de los patrones del fichero de entrada, es decir, 25 patrones de los 100 que lo componen.

Tras este proceso, el fichero de aprendizaje para cada una de las particiones debe ser distribuido en dos ficheros: fichero de entrenamiento y fichero de test que contendrán los patrones de entrenamiento y los patrones de test respectivamente que va a utilizar la red neuronal de base radial.

El número de patrones que se distribuirán a cada uno de los dos ficheros viene determinado por el valor del parámetro propVal en la estructura TCONFIG que también se encuentra en el fichero 'config.ini'.

Siguiendo con el ejemplo anterior en el que para cada partición, el fichero de entrenamiento está compuesto de 75 patrones, si el valor del parámetro propVal fuera 0.80, el fichero de entrenamiento tendría el 80% de los patrones (60) y el fichero de este el 20% restante(15).

Se expone a continuación, de forma gráfica, el ejemplo anterior para la partición 1.



B.3.3 – Clase Población

La clase población fue implementada dentro de los ficheros ‘población.h’ y ‘población.cpp’.

Es la clase de los objetos de tipo población (o generación) de posibles soluciones (cada una de ellas es un cromosoma).

A continuación se exponen las partes mas relevantes del diseño de esta clase para el funcionamiento del sistema:

Cálculo de la fitness de los elementos de la población

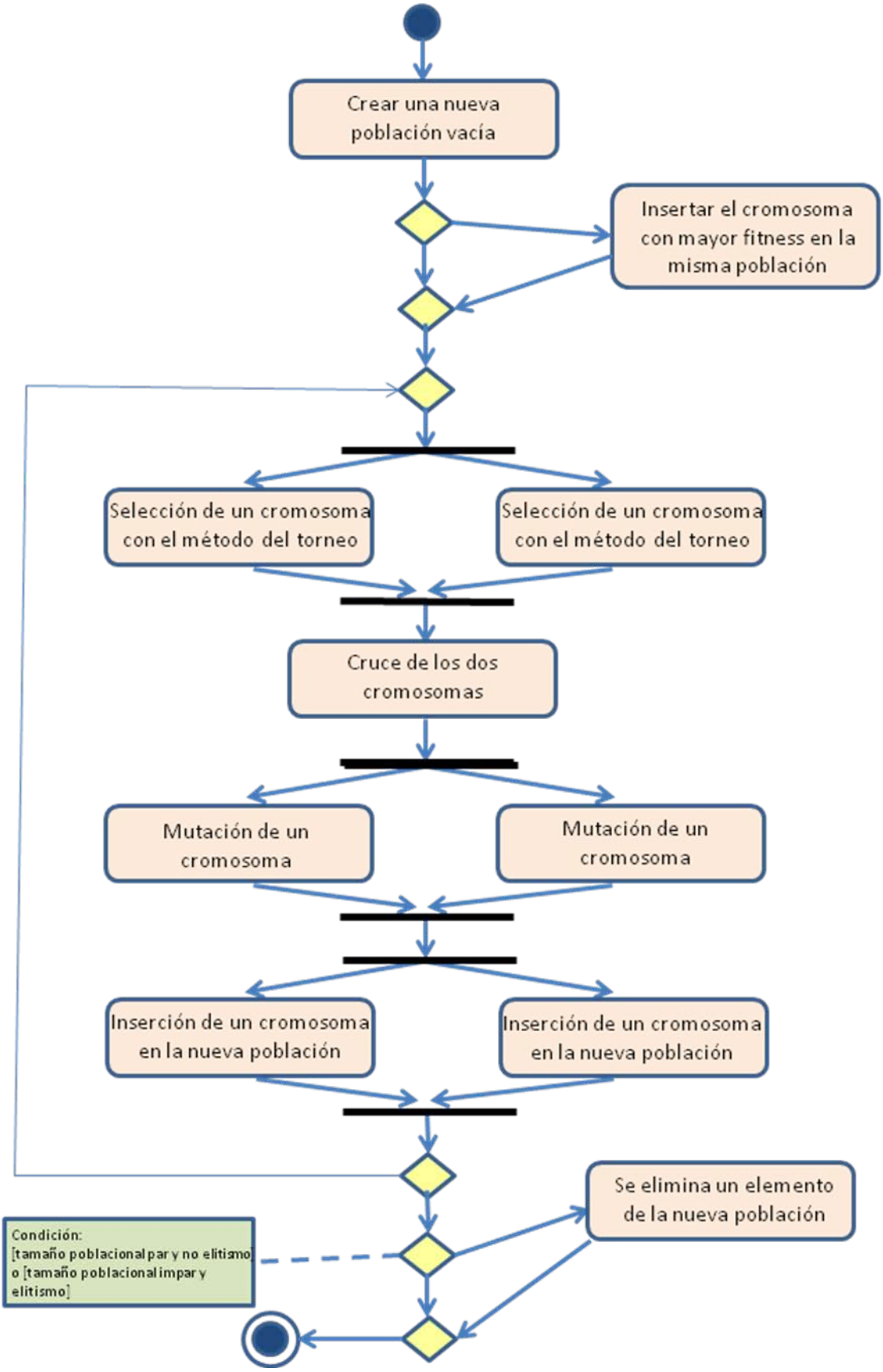
Para calcular la fitness de cada uno de lo elementos que conforman una población, el sistema que se implementó emplea los métodos población::calcularFitness (TCONFIG*) y población::calcularFitness(TCONFIF*,char*,char*), siendo este último el empleado en casa de que haya validación cruzada, proporcionando así a la RNBR, a través de estos dos parámetros, los nombres de los ficheros de entrenamiento y de test.

Estos métodos constan de un bucle muy simple que va ejecutando el método cromosoma::obtenerFitness(TCONFIG) o cromosoma::obtenerFitness (TCONFIG, char*. char*) dependiendo de si es usada la validación cruzada o no.

Reproducción de la población

Para pasar de una generación a la siguiente se utiliza el método población::reproducir (TCONFIG*) que utiliza y explota principalmente la actividad “Aplicar operadores genéticos”.

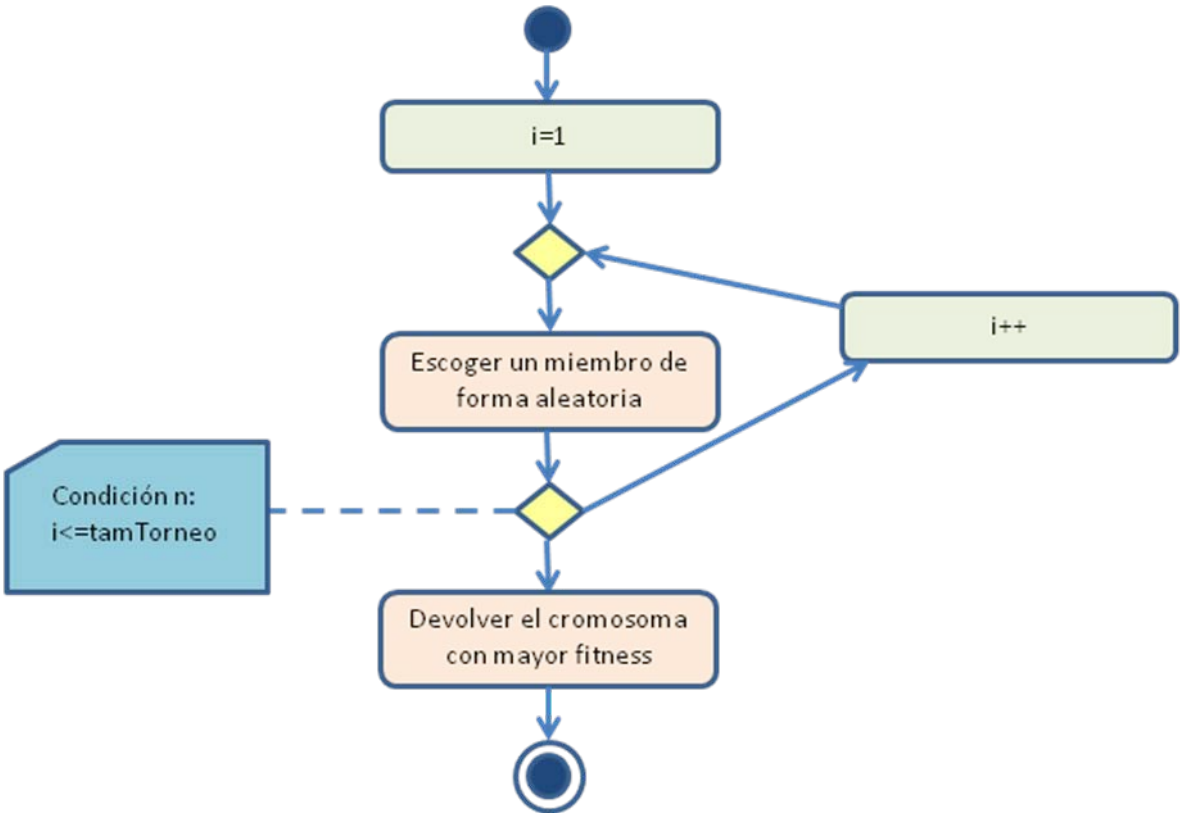
Se expone, a continuación, el diagrama de actividad que explica el flujo de la reproducción de una población.


Crear una nueva población vacía
Insertar el cromosoma con mayor fitness en la misma población
Selección de un cromosoma con el método del torneo
Selección de un cromosoma con el método del torneo
Cruce de los dos cromosomas
Mutación de un cromosoma
Mutación de un cromosoma
Inserción de un cromosoma en la nueva población
Inserción de un cromosoma en la nueva población
Se elimina un elemento de la nueva población
Condición:
[tamaño poblacional par y no elitismo]
o [tamaño poblacional impar y elitismo]

Método del torneo para la selección

El torneo es el método empleado por el sistema para realizar el proceso de selección. El algoritmo se implementó en el método población::seleccionarTorneo (int,cromosoma*).

A continuación se expone el método del torneo, de forma gráfica, mediante el diagrama de actividad del flujo de la selección de un elemento en una población.



Se especifica de forma mas detallada el funcionamiento de cada una de las actividades expuestas en el gráfico anterior.

- *Escoger un miembro de forma aleatoria*: entre todos los miembros de la población se selecciona de manera aleatoria uno de ellos. Este proceso es llevado a cabo por el sistema a través del método población::miembroAleatorio().
- *Devolver el cromosoma con mayor fitness*: se devuelve un puntero al cromosoma que tiene mayor fitness entre todos los que han sido seleccionados.

El tamaño del torneo se puede modificar y viene dado por el parámetro tamTorneo de la estructura TCONFIG, leído del fichero de configuración ‘config.ini’.

B.3.4. - Clase cromosoma.

La clase cromosoma fue implementada en los ficheros ‘cromosoma.h’ y ‘cromosoma.cpp’. Es la clase de los objetos tipo cromosoma, donde cada objeto representa una matriz de distancias como posible solución del problema.

B.3.4.1. - Representación de la solución:

Representación binaria

En el programa la matriz de distancias se guarda como una binaria con una longitud LONGVAL * DIM * DIM (LONGVAL y DIM son macros definidas en ‘global.h’).

El número de bits en cada celda será LONGVAL. Mientras que el número de bits de cada fila y en cada columna será LONGVAL * DIM.

Dependiendo de si el tipo de matriz es diagonal o simétrica, la estructura de la matriz solución será distinta:

- Si se utiliza una matriz diagonal todas las posiciones de la cadena que estén fuera de la diagonal principal serán cero, mientras que el resto de posiciones podrán tomar cualquier valor.
- Si por el contrario se utiliza un tipo de matriz simétrica (Mahalanobis) las LONGVAL * DIM posiciones de la primera fila y de forma análoga de la última columna serán distintas de cero, de la segunda fila serán distintas de cero las LONGVAL * (DIM-1) posiciones y así sucesivamente hasta que se complete la dimensión de la matriz de distancias.

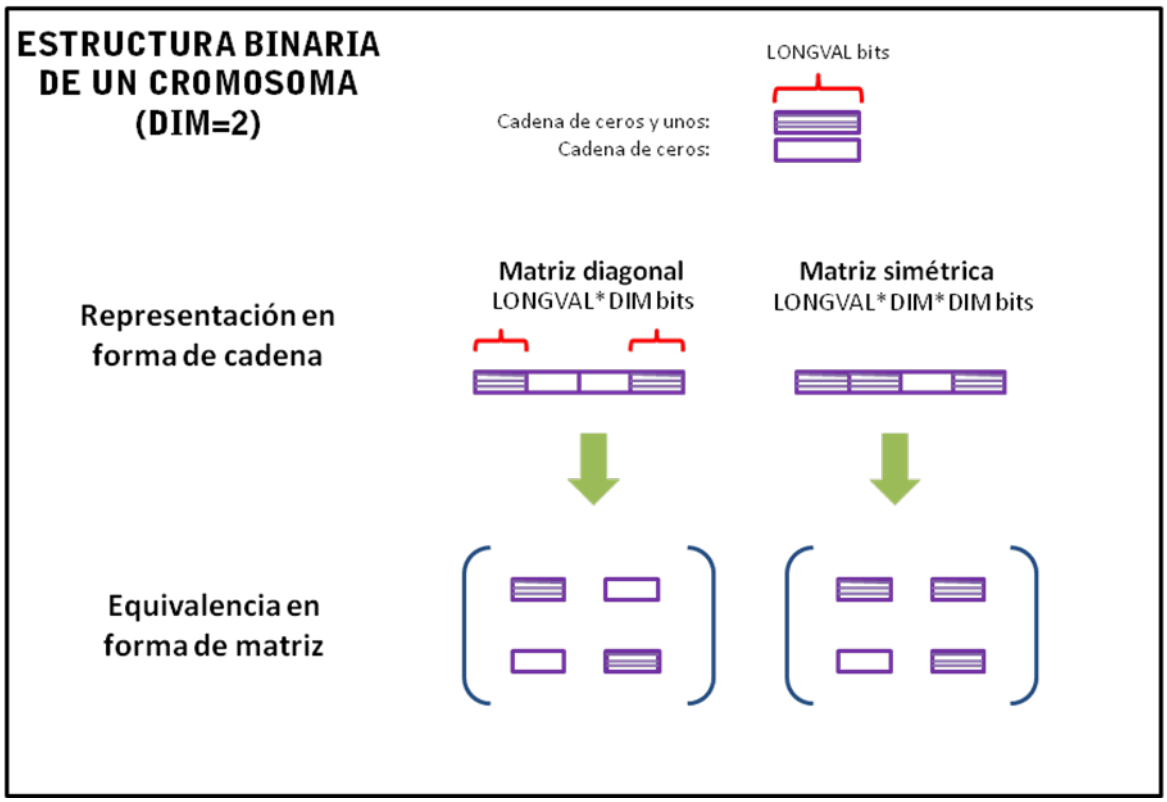
El valor del campo tipoDistancias de la estructura TCONFIG determinará si la matriz es diagonal o simétrica.

La variables tipoDistancias se leerá del fichero de configuración ‘config.ini’.

Las posiciones de la matriz simétrica que sean iguales a cero se completarán teniendo en cuenta la simetría de la matriz.

Esto se llevará a cabo al realizar la conversión a su equivalencia en valores reales.

La estructura nombrada anteriormente, se puede representar de la siguiente manera:

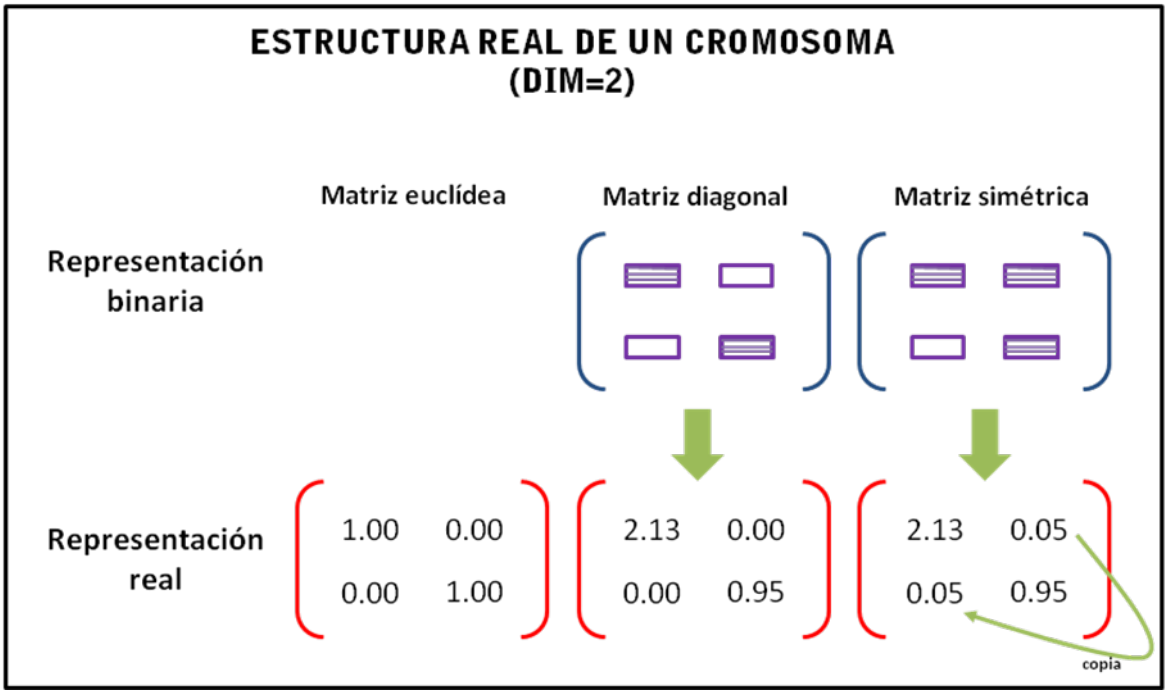


Representación real

El programa también guarda una matriz de DIM filas y DIM columnas (siendo DIM la dimensión de la matriz) con la representación real de la solución para así poder emplearla en la RNBR a la hora de calcular la fitness.

La matriz se guarda como un matriz [DIM][DIM] de números reales. Como la aplicación está desarrollada en C++ el tipo de datos será double.

Al igual que en el caso de representación binaria, esta matriz puede ser representada de manera gráfica:



Como se puede ver en la figura anterior, en el caso de que se use una matriz simétrica se copiarán los elementos por encima de la diagonal para conservar así la simetría de la matriz.

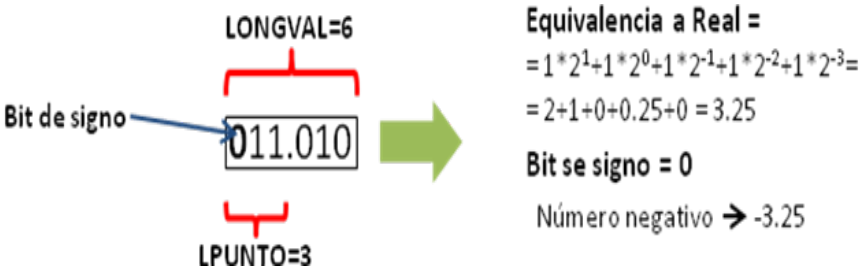
Paso de representación binaria a real

Para pasar de la representación de la cadena de LONGVAL*DIN*DIM valores binarios a la matriz de valores reales [DIM][DIM] se emplea el método cromosoma::pasarAReal().

La conversión se realiza empleando una notación de punto fijo con un bit de signo:

- El punto fijo es establecido con el parámetro LPUNTO (macro definida en el fichero 'global.h'), que representa la posición en la que se encontrará el punto binario, contando de izquierda a derecha (a partir del bit de signo).
- El bit de signo es situado en el más significativo, de manera que es cero si representa un número negativo y un uno en caso contrario.

En la siguiente figura se encuentra un ejemplo de aplicación de esta conversión:



B.3.4.2 - Operadores genéticos:

Reproducción

Este operador se lleva a cabo a nivel de la población, no a nivel de individuo, y el método empleado para realizar la selección de los individuos para reproducirlos es población:: seleccionarTorneo(int , cormosoma*)

Cruce

El operador de cruce es utilizado con el método cromosoma:: cruzar (cromosoma*, int, float, int). El operador cruce por un punto es llevado a cabo mediante el método cromosoma:: combinar(cromosoma*, int) y el operador cruce por dos puntos mediante cromosoma:: combinar2 (cromosoma*, int, int).

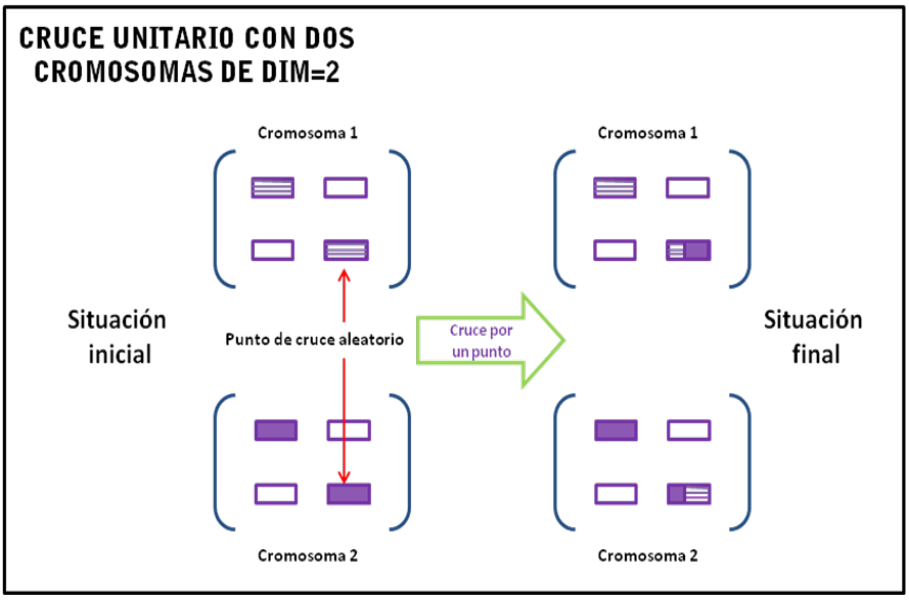
Las posiciones que son válidas en la matriz para poder llevar a cabo el cruce son aquellas cuyos valores puedan ser distintos de cero.

El sistema puede realizar tanto cruces por un punto como por dos y lo hace en función del parámetro tipoCruce de la estructura TCONFIG.

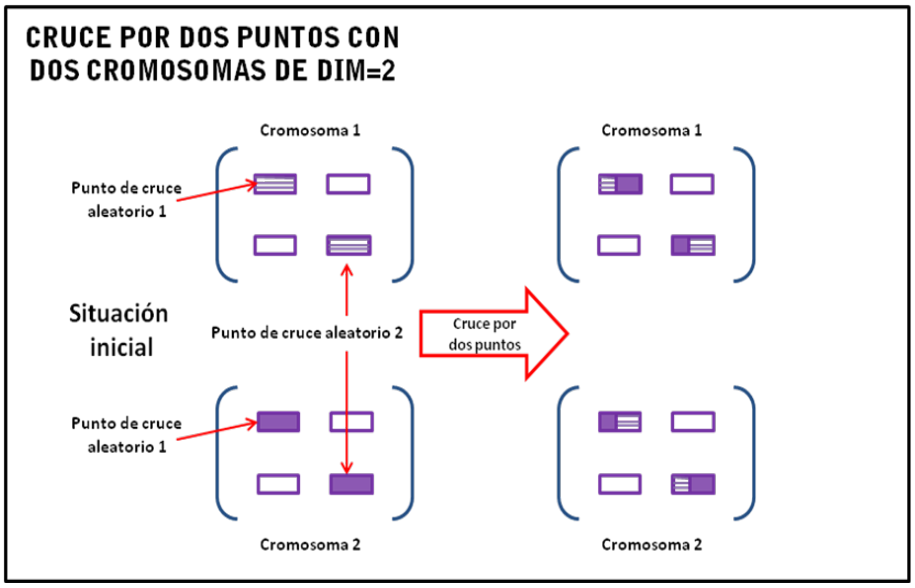
La probabilidad de que haya un cruce, viene determinada por el valor del parámetro pCrz que también está almacenado como campo de la estructura TCONFIG.

Tanto el parámetro tipoCruce como el pCrz son leídos del fichero de configuración “config.ini”.

A continuación se muestran gráficamente los cruces por un punto y por dos puntos en la representación binaria de dos matrices solución:



En el ejemplo gráfico que se presenta a continuación (Cruce por dos puntos) hay que tener en cuenta que se emplea una matriz diagonal, por lo que en el caso de que se utilizara matriz simétrica el cruce se podrá producir en las posiciones que son distintas de cero.

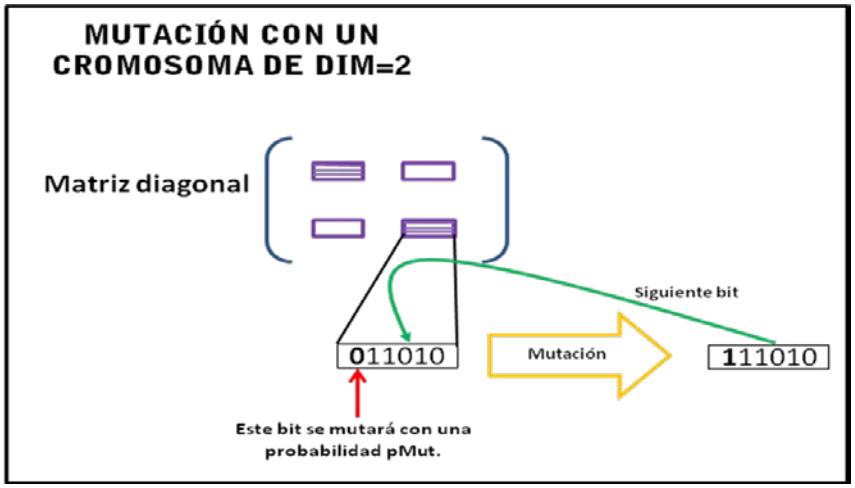


Mutación

El método que lleva a cabo el operador mutación es cromosoma:: mutar (float ,int). Como en el caso de cruce, las posiciones válidas de la matriz, para llevar a cabo una mutación son aquellas en las que la matriz pueda tener elementos distintos de cero y que dependen de si se trabaja con una matriz de distancias diagonal o simétrica.

El parámetro pMut que se encuentra en la estructura TCONFIG y que es leído del fichero de configuración "config.ini", es el que determina la probabilidad de que se produzca una mutación en cada bit.

A continuación se muestra gráficamente la mutación en una matriz diagonal:



B.3.4.3 - Cálculo de la fitness de un cromosoma:

Los métodos cromosoma::calcularFitness (TCONFIG*) y cromosoma::calcularFitness (TCONFIG*, char*, char*) son utilizados para calcular la fitness de cada cromosoma.

El método cromosoma:: calcularFitness (TCONFIG*, char*, char*) es utilizado en el caso de que se emplee validación cruzada, puesto que se tiene que proporcionar a la RNBR los nombres de los ficheros de entrenamiento y de test, que se generan automáticamente.

Ambos métodos son utilizados por los métodos población::calcularFitness (TCONFIG*) y población::calcularFitness (TCONFIG*, char*, char*) respectivamente para calcular la fitness de cada uno de los elementos de la población.

El funcionamiento de los mismos, consiste básicamente en obtener la instancia única de la clase fitness e invocar el método correspondiente,

Fitness::obtenerFitness (TCONFIG, double [DIM][DIM]) o
Fitness::obtenerFitness (TCONFIG, doublé [DIM][DIM], char*, char*)

, independientemente de que se use validación cruzada o no

B.3.5. – Clase fitness.

La clase fitness fue implementada en los ficheros ‘fitness.h’ y ‘fitness.cpp’. Como la clase global, esta clase solo tiene una instancia.

Esta clase contiene las funciones necesarias para calcular la fitness con la RNBR.

B.3.5.1. - Obtención de la fitness utilizando la red

Existen cuatro métodos distintos que se pueden emplear para obtener la fitness utilizando la RNBR.

1. *Fitness::obtenerFitness(TCONFIG)*

   Para obtener la fitness, se utiliza la RNBR usando la distancia euclídea común y sin utilizar validación cruzada.

2. *Fitness::obtnerFitness(TCONFIG, char*, char*,int)*

   Se utiliza la RNBR para obtener la fitness usando la distancia euclídea común y utilizando validación cruzada. En este caso se le deben pasar los nombres de los ficheros de entrenamiento y test a la RNBR.

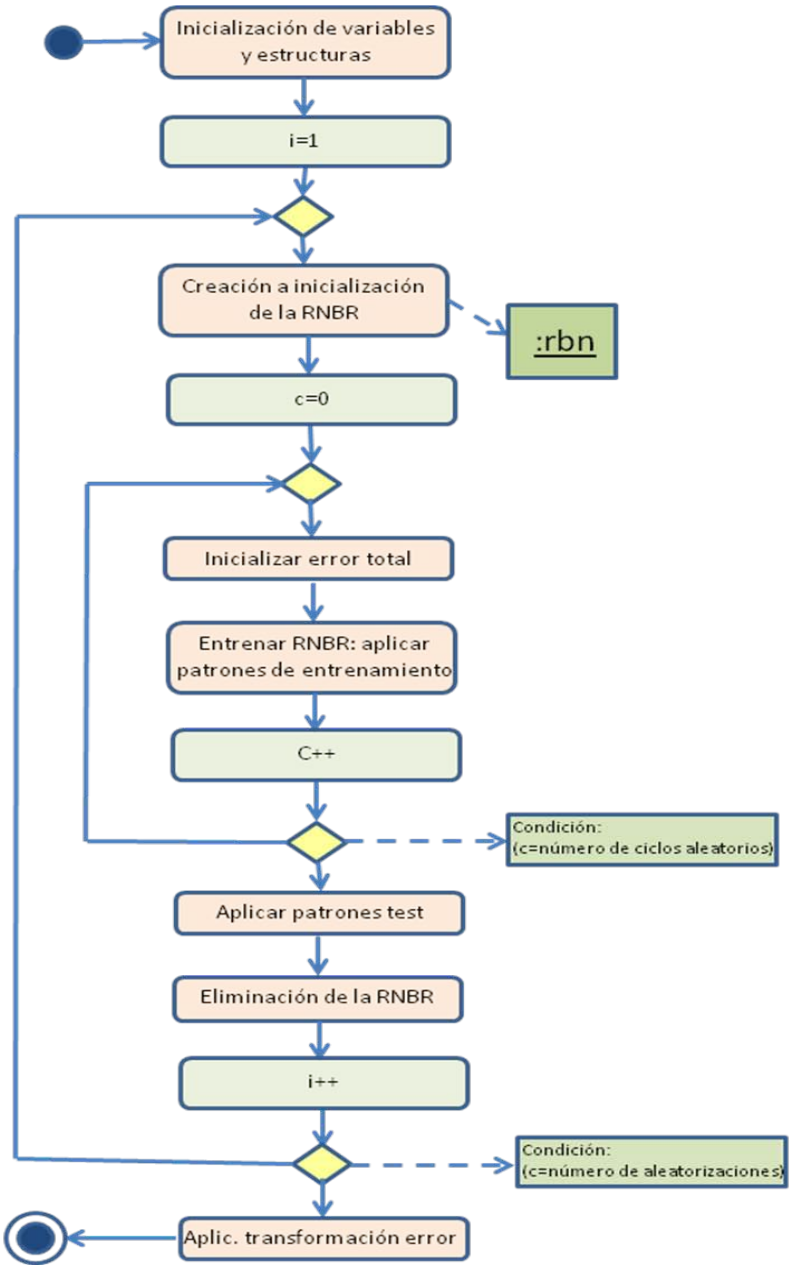
3. *Fitness::obtenerFitness(TCONFIG, doublé [DIM][DIM])*

   Se utiliza la RNBR para obtener la fitness usando la distancia de Mahalanobis y sin emplear validación cruzada. Como parámetros se le pasa la matriz de distancias independientemente que sea simétrica o diagonal.

4. *Fitness::obtenerFitness (TCONFIG, doublé [DIM][DIM],char*,char*,int)*

   La RNBR se utiliza en este caso para obtener la fitness usando la distancia de Mahalanobis y utilizando validación cruzada. Como parámetros se le pasa la matriz de distancias ya sea simétrica o diagonal. También, en este caso, se le pasa como parámetros los nombres de los ficheros de entrenamiento y test a la RNBR.

A continuación se expone el diagrama de secuencia que representa el funcionamiento básico de estos métodos:



B.3.6 – Clase rbn.

En la clase rbn fue implementado el algoritmo de aprendizaje híbrido estándar que se emplea en RNBR, en este sistema se emplean distancias euclídeas.

La clase rbn fue implementada en los ficheros ‘rbn.h’ y ‘rbn.cpp’. Es la clase de los objetos que contienen la RNBR.

B.3.6.1 – Modificación del entrenamiento de la red

El método de entrenamiento de la RNBR rbn::entrenar(FILE*,FILE*,int) va a ser modificado con el uso de una matriz de distancias, ya que hay que pasarle a este método los valores de esta matriz para poder calcular las funciones de activación de las neuronas de la capa oculta utilizando esta métrica.

El método modificado será rbn::entrenar(FILE*,FILE*,int,doublé[DIM][DIM]). Y en el entrenamiento en lugar de hacer la llamada a rbn::activarFunciones(), se invocará a rbn::activarFunciones(doublé[DIM][DIM]) pasando los valores de la matriz de distancias.

B.3.6.2. – Modificación de la validación de la red

Al igual que ocurre con el entrenamiento, también es necesario cambiar el método de validación para aplicar los patrones de test a la red, utilizando funciones de activación que tengan en cuenta la matriz de distancias.

El método de validación de la RNBR aplicarTest (FILE*, FILE*, doublé*, doublé*, int*, int*) se va a ver modificado con el uso de una matriz de distancias euclídeas generalizada.

El método modificado será aplicarTest (FILE*, FILE*, doublé*, doublé*, int*, int*, doublé[DIM][DIM]).

Y al igual que ocurría en el entrenamiento, para el cálculo de la activación, en vez de invocar a rbn::activarFunciones(), se invoca rbn::activarFunciones(doublé[DIM][DIM]) pasando los valores de la matriz de distancias.

B.3.6.3. - Modificación del cálculo de la activación de la red

Se ha debido modificar la función de activación radial gaussina utilizando la distancia euclídea, para poder utilizar la distancia de Mahalanobis.

En este caso ya no se tendrá una función de activación radial gaussiana, sino una función de activación elíptica, dependiendo de la configuración de la matriz de distancias que se aplique.

En el caso del uso de distancias euclídeas clásicas se emplea el método rbn::activarFunciones(), mientras que se ha debido añadir otro método rbn::activarFunciones (doublé[DIM][DIM]) al que se le pasa la matriz de distancias.

Estos métodos serán invocados, para cada neurona de la capa oculta de la RNBR, al método que devuelve la activación: neurona::activacion() si se usa la distancia euclídea clásica, y neurona::activacion(doublé[DIM][DIM]) si se usan distancias euclídeas generalizadas.

B.3.7. – Clase neurona.

La clase neurona fue implementada en los ficheros ‘neurona.h’ y ‘neurona.cpp’. Es la clase de los objetos neurona que forman la capa oculta de la RNBR.

B.3.7.1. - Modificación del cálculo de la función de activación

Al igual que se ha visto anteriormente al hablar de la clase rbn, para la clase neurona se ha debido cambiar también el cálculo de la función de activación radial gaussiana de las neuronas de la capa oculta de la RNBR.

En el caso de que se utilice la distancia euclídea clásica, se usa el método neurona:: activación() en el cual se implementa la función radial gaussiana utilizando la distancia euclídea clásica, lo que da lugar a:

$$y_j = e^{-r_j^2/2\sigma_j^2}$$

$$r_j^2 = \|x - c_j\|^2 = \sum_{i=1}^{n} (x_i - c_{ij})^2$$

Por el contrario, cuando se emplean distancias euclídeas generalizadas, se le debe pasar la matriz de distancias al método neurona::activación(double [DIM][DIM]), quedando la ecuación modificada del siguiente modo:

$$r_j^2 = (x - c_j)^T M^T M (x - c_j)$$

Donde:

M es la matriz de Mahalanobis

Para llevar a cabo el cálculo de la distancia del centro c al punto x, en caso de utilizar la distancia euclídea se usará el método punto::euclídea(punto*), mientras que si se emplea una matriz de Mahalanobis el método empleado será punto:: mahalanobis (punto*, doublé [DIM][DIM]).

B.3.8. - Clase punto.

La clase punto fue implementada en los ficheros ‘punto.h’ y ‘punto.cpp’. Es la clase de los objetos que representan los puntos necesarios para el funcionamiento de la RNBR.

B.3.8.1. - Cálculo de la distancia euclídea

Al tener que calcular la distancia de Mahalanobis además de la distancia euclídea clásica, se ha tenido que desarrollar un nuevo método.

El método que se desarrolló para calcular la distancia euclídea entre un punto y otro es punto::euclídea(punto*), mientras que para calcular la distancia euclídea generalizada es mahalanobis(punto*, doublé[DIM][DIM]).

Este método implementa el cálculo de la distancia de Mahalanobis utilizando una matriz M. La implementación es válida tanto para una matriz diagonal como simétrica.

Con este método se calcula de igual manera la distancia entre el centro y otro punto para calcular la activación de la neurona.

ANEXO C

BIBLIOGRAFÍA

[Anderberg, 73] Anderberg, M. *Cluster Analysis for Applications*. New York: Academic Press, 1973.

[Baker, 85] Baker, J.E., *Adaptative selection methods for genetic algorithms*. Ed, J. J. Grefensette, Proceedings of the First International Conference on Genetic Algorithmas and Their Applications, Eribaum, 1985.

[Bellman, 61] Bellman R., *Adaptative Control Processes*: A Guied Tour, Princeton, NJ: Princeton University Press, 1961.

[Bethke, 80] Bethke, A.D., *Genetic Algorithms as Function Optimizers*. Tesis doctoral Universidad de Michigan, 1980.

[Bromead, 88] Bromead, D.S., y D. Lowe. *Multivariable functional interpolation and adaptative networks, en Complex System, vol. 2*, pp. 321-355, 1988

[Caruana, 88] Caruana, R.A. y Schaffer J.D. *Representation and hidden bias: Gray vs. binary coding for genetic algorithms*. Proceedings of the Fifth International Conference of Machine Learning. Morgan Kaufmann, 1988.

[Chen, 92] Chen, S., B. Mulgrew, y S. McLaughlin. *Adaptative Bayesian feedback equalizer based on a radial basis function network*, IEEE International Conference on Communications, vol. 3, pp. 1267-1271, Chicago, 1992.

[Darwin, 59] Darwin, C., *On the origin of the species*. Ed. John Murray, 1859

[De Jong, 75] De Jong, K. A., *An Análisis of the Behavior of a Class of Genetic Adaptative Systems*. Tesis Doctoral. Universidad de Michigan, Ann Arbor, 1975.

[De Jong, 93] De Jong, K. A. y Sarma J., *Generation gaps revisited.* En L.D. Whitley, Foundation of Genetic Algorighms 2. Morgan Kaufmann, 1993.

[De la Maza, 91] De la Maza M. y Tidor B., *Boltzmann Weighed Selection Improves Performance of Genetic Algorithms*. A. I. Memo 1345, MIT Artificial Intelligence Laboratory, 1991.

[De la Maza,93] De la Maza M. y Oidor. B., *An analisis of selection procedures with particular attention paid to proporcional and Boltzmann selection*., Ed. S. Forrest, Proceedings of the Fifth International Conferenceon Genetic Algorithms. Morgan Kaufmann, 1993.

[Duda, 73] Duda, R.O. y P.E. Hart Pattern *Classification and Scene Análisis*, New York, Wiley, 1973.

[Fogel, 66] Fogel,D.B., Owens, A.J., y Walsh, M. J. *Artificial Intelligence through Simulated Evolution*. Wiley, 1966.

[Forrest, 85] Forrest S., Scalling *fitness in the genetic algorithm*. Documentation for PRISIONERS DILEMMA and NORMS Programs That use the Genetic Algorithm. Unpublished manuscript, 1985.

[García, 55] García,O. *Utilización de distancias de Mahalanobis en redes de neuronas de base radial mediante algoritmos genéticos*, PFC, 2004

[Goldberg, 89] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[Goldberg, 90] Goldberg, D. E., *A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing*. Complex System 4: 445-460, 1990.

[Goldberg, 91] Goldberg, D. E., *A comparative analysis of selection schemes used in genetic algorithms*. En G, Rawlins, Foundations of Genetic Algorithms. Morgan Kaufmann, 1991.

[Hartigan, 75] Hartigan J.A. *Clustering Algorithms*. John Wiley & Sons, pag. 12, 1975

[Hebb,49] Hebb, D.O. The *organization of Behaviour: A Neuropsychological Theory*, New York: Wiley, 1949

[Holland, 75] Holland, J.H., *Adaptation in Natural and Artificial System.* University of Michigan Press, 1975.

[Hush, 93] Hush, D. R, Horne, B. G. *Progress in supervised neural networks. What's new since Lippmann?*. IEEE Signal Proc. Mag., 8-38, enero 1993.

[Janikov, 91] Janikow, C. Z., y Mchalewicz, Z., *An experimental comparison of binary and floating point representations in genetic algorithms*. Eds. R.K. Belew y L. B.

Booker., Proceedings of the Fourth International Conference of Genetic Algortihms, 1991.

[Kohonen, 89] Kohonen T. *Self-Organization and Associative Memory*. 3th edition, Springer-Verlag, 1989.

[Kohonen, 90] Kohonen T. *The Self-Organization Map*. Proc of the IEEE, 78, 9, pp-1464-1480, 1990.

[Lee, 92] Lee, S. *Supervised learning with Gaussian potentials*. En [Kosko, 92], pp 189-227,1992.

[Lippmann, 89] Lippmann, R.P. *Pattern Classification using neural networks, IEEE Communications Magazine, vol 27*, pp. 47-64, 1989.

[McCulloch, 43] McCulloch, W.S., y Pitts, W. *A logical calculus of the ideas inmmanent in nervious activity, en Bulletin of Mathematical Biophysics, vol. 5*, pp. 115-333, 1943

[Minsky, 69] Minsky, M.L. y S.A. Papert. *Perceptrons*, Cambridge. MIT Press, 1969.

[Moody, 89] J.E. Moody y C. J. Darken, *Fast Learning in Networks of Locally-Tuned Proccessing Units*, en Neural Computation, 1, pp. 281-294, 1989.

[Müller, 90] Müller, B., Reinherd, J. Neural Networks. *An Introduction*, Springer-Verlag, 1990.

[Newmann, 58] Von-Newmann, J. The *computer and The Brain*, New Haven. Yale University Press, 1958.

[Prügel-Bennett, 94] Prügel-Bennet, A., y Shapiro, J. L., *An análisis of genetic algoritms using statistical mecanics*. Physical Review Letters 72, no. 9: 1305-1309, 1994

[Rechemberg, 73] Rechemberg, I. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.

[Ripley, 25] Ripley, B.D.: *Pattern Recognition and Neural Networks*. Cambridge: Cambridge, University Press, 1996.

[Rossenblatt, 58] Rossenblatt, F., *The Perceptron: A probabilistic model for information storage and organization in the brain*, en Psycological review, vol. 65, pp. 386-408, 1958.

[Rumelhart, 86] D. E. Rumelhart, G.E. Hinton, R. J. Williams Learning *Internal Representations by error propagation*, en D.E. Rumerlhart y J. L. McClelland, en eds., Parallel Distributed Processing: Exploration in the Microstructure of cognition. Bradford Books / MIT Press, Cambridge, MA, 1986.

[Ruppert, 94] Ruppert, D., y Wand, M. P. *Multivariate locally weighted least squares regression*. The Annuals of Staticstics, 22(3), pp. 1346-1370, 1994.

[Syswerda, 89] Syswerda, G., *Uniform croosover in genetic algorithms*. Ed. J. D. Schaffer, Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufmann, 1991.

[Syswerda, 91] Syswerda, G., *A study of reproduction in generational and steady-state genetic algorithms*. Ed. G. Rawlins, Foundations of Genetic Algorithms. Morgan Kaufmann, 1991.

[Wand, 93] Wand, M.P., y Jones, M. C., *Kernel Smoothing*. Chapman y Hall, London, 1994

[Wright, 91] Wright, A. H., *Genetic Algorithms for real parameter optimization*. Ed. G. Rawlins, Foundation of Genetic Algorithms. Morgan Kauffmann, 1991.